TRABAJO FIN DE GRADO

SUPPORT VECTOR REGRESSION: PROPIEDADES Y APLICACIONES

Realizado por: Juan José Martín Guareño

Supervisado por: Dr. Rafael Blanquero Bravo y
Dr. Emilio Carrizosa Priego

FACULTAD DE MATEMÁTICAS

DEPARTAMENTO DE ESTADÍSTICA E INVESTIGACIÓN OPERATIVA

Índice general

Abstract

Statistical learning plays a key role in many areas of science, finance and industry. In particular, supervised learning plays a key role in the fields of statistics, data mining and artificial intelligence, intersecting with areas of engineering and other disciplines.

Mathematical optimization has played a crucial role in supervised learning. Techniques from very diverse fields within mathematical optimization have been shown to be useful. Support Vector Machine (SVM) and Support Vector Regression (SVR) are ones of the main exponents as application of the mathematical optimization to supervised learning. SVM and SVR are state of the art methods for supervised learning and regression. These geometrical optimization problems can be written as convex quadratic optimization problems with linear constraints, in principle solvable by any nonlinear optimization procedure.

In this work we analyze SVMs and SVRs: how the problems are obtained and expressed in a manageable way. On the one hand, we describe the techniques used by the algorithms of supports vectors dedicated to the classification, in linear and nonlinear cases. On the other hand, we focus on the theoretical development of the techniques in the field of support vector regression. We pay more attention to the nonlinear case, where the algorithm of support vector shows its full potential, using a kernel function to calculate a nonlinear approximation function.

Finally we bring these theoretical procedures into practice with the help of the statistical language and environment R.

Capítulo 1

Introducción

El aprendizaje estadístico desempeña un papel clave en muchas áreas de la ciencia, las finanzas y la industria. Algunos ejemplos de problemas de aprendizaje son: predecir si un individuo que ha tenido un accidente de tráfico volverá a tener otro accidente, en base a su conducción; predecir el valor de una acción en seis meses desde el momento del estudio, sobre la base de las medidas de rendimiento de la empresa y los datos económicos; o identificar los factores de riesgo para una enfermedad, basándose en variables clínicas y demográficas.

En particular, el aprendizaje supervisado es una parte importante del aprendizaje estadístico, con gran influencia en los campos de la minería de datos y la inteligencia artificial. Partiendo de un conjunto $\mathcal{C}$, constituido por ejemplos de una población específica de los cuales se conocen una serie de variables, una vez determinada la variable respuesta y las variables explicativas, se buscan procedimientos para predecir el valor de la variable respuesta, conocido el valor de las variables explicativas de un futuro ejemplo de dicha población. En el caso de que la variable respuesta fuese cualitativa estaríamos ante un problema de clasificación, y si la variable respuesta fuese cuantitativa tendríamos un problema de regresión.

La optimización matemática es crucial en el aprendizaje supervisado, donde han demostrado ser útiles técnicas precedentes de muy diversos campos dentro de la misma. Las máquinas de vectores soporte (SVM) y la regresión de vectores soporte (SVR) son unos de los principales exponentes de la aplicación de la optimización matemática al aprendizaje supervisado.
donde han demostrado ser útiles técnicas precedentes de muy diversos campos.

Tanto para el caso de las (SVM) como (SVR), el objetivo es realizar la predicción a partir de un problema de optimización geométrica que se puede escribir como un problema de optimización cuadrático convexo con restricciones lineales, en principio resoluble mediante cualquier procedimiento de optimización no lineal.

En este documento se muestra cómo podemos obtener dichos problemas y las transformaciones necesarias para facilitar su resolución.

En el capítulo 2 describiremos las técnicas utilizadas por los algoritmos de vectores soportes dedicados a la clasificación, en los casos Lineal y No Lineal.

El capítulo 3 se centra en el desarrollo teórico de las técnicas de los vectores soporte en el ámbito de la regresión. Se presenta una primera parte donde se realiza el estudio de la función de regresión a partir de una función lineal y una segunda parte más interesante donde el algoritmo de los vectores soporte muestra todo su potencial, con el uso de una función kernel para el cálculo de una función de aproximación no lineal.

En el capítulo 4 llevaremos estos procedimientos teóricos a la práctica con la ayuda del programa estadístico *R*.

Capítulo 2

Support Vector Machines

Las máquinas de vectores soporte (SVM, del inglés Support Vector Machines) tienen su origen en los trabajos sobre la teoría del aprendizaje estadístico y fueron introducidas en los años 90 por Vapnik y sus colaboradores, [1]. Aunque originariamente las SVMs fueron pensadas para resolver problemas de clasificación binaria, actualmente se utilizan para resolver diversos tipos de problemas, por ejemplo, la regresión, en el cual nos centraremos más adelante. De hecho, desde su introducción, han ido ganando un merecido reconocimiento gracias a sus sólidos fundamentos teóricos.

La idea es seleccionar un hiperplano de separación que equidiste de los ejemplos más cercanos de cada clase para, de esta forma, conseguir lo que se denomina un margen máximo a cada lado del hiperplano. Además, a la hora de definir el hiperplano, sólo se consideran los ejemplos de entrenamiento que distan del hiperplano la distancia margen. Estos ejemplos reciben el nombre de *vectores soporte*.

2.1. SVM Caso Lineal Separable

Dado un conjunto separable de ejemplos $\mathcal{C} = \{(x_1, y_1) ..., (x_n, y_n)\}$, donde $x_i \in \mathbb{R}^d$ e $y_i \in \{+1, -1\}$ se puede definir un hiperplano de separación, como una función lineal que es capaz de separar dicho conjunto:

$$D(x) = (w_1 x_1 + ... + w_d x_d) + b = \langle w, x \rangle + b\,,$$

donde $w_i \in \mathbb{R} \quad \forall i = 1, ..., d$ y $b \in \mathbb{R}$.

Dicho hiperplano deberá cumplir las siguientes desigualdades:

$$\begin{aligned} \langle w, x_i \rangle + b \geq 0 \qquad & \text{si} \quad y_i = +1 \quad \forall i = 1, ..., n \\ \langle w, x_i \rangle + b \leq 0 \qquad & \text{si} \quad y_i = -1 \quad \forall i = 1, ..., n \end{aligned}$$

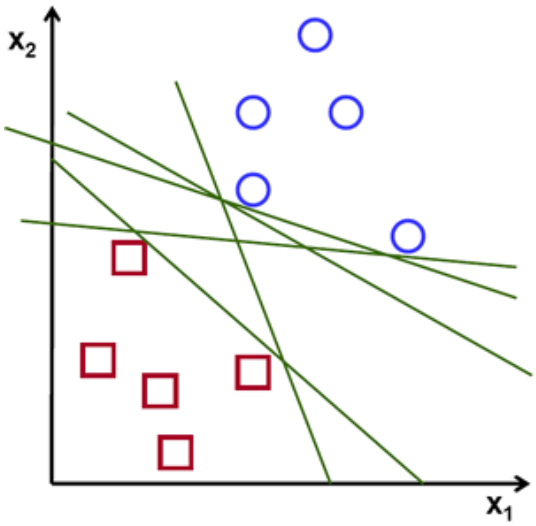
O escrito de forma más compacta de la siguiente forma:

$$y_i \left(\langle w, x_i \rangle + b \right) \geq 0 \quad \forall i = 1, ..., n$$

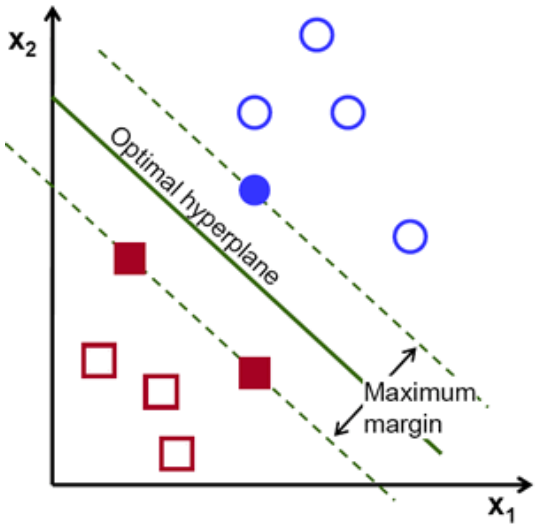
Es decir,

$$y_i D(x_i) \geq 0 \quad \forall i = 1, ..., n \qquad (1)$$

El hiperplano que permite separar las dos clases no suele ser único. La selección de un hiperplano de entre todos los posibles hiperplanos de separación se realizará a partir del concepto de *margen*, que se define como la distancia mínima entre dicho hiperplano y el ejemplo más cercano a cada clase; se denotará por τ.



(a) En esta representación apreciamos la NO unicidad de solución.



(b) Mientras que en esta, una vez impuesto el margen máximo, el hiperplano es único.

Por geometría se sabe que la distancia entre el hiperplano y un ejemplo x' viene dada por

$$\frac{| D(x') |}{\| w \|}$$

De la forma que hemos definido el *margen* y considerando la desigualdad (1) debe cumplirse:

$$\frac{y_i D(x_i)}{\| w \|} \geq \tau \quad \forall i = 1, ..., n$$

De la expresión anterior se deduce que encontrar el hiperplano óptimo, es equivalente a encontrar el valor de w que maximiza el margen. Existen infinitas soluciones de la forma:$\lambda(\langle w, x \rangle + b)$, con $\lambda \in \mathbb{R}$, que representan el mismo hiperplano.

La desigualdad anterior se puede escribir de la forma equivalente:

$$y_i D(x_i) \geq \tau \| w \| \quad \forall i = 1, ..., n$$

De esta forma, para limitar el número de soluciones a una sola, fijamos arbitrariamente el producto de τ y la norma de w a la unidad.

$$\tau \| w \| = 1$$

De esta forma concluimos que maximizar el margen es equivalente a minimizar la norma de w.

Si a esto le imponemos las condiciones mencionadas anteriormente lo podemos escribir de forma equivalente como un problema de programación cuadrática de la siguiente forma:

$$\begin{aligned} \text{mín} \quad & \frac{1}{2} \parallel w \parallel^2 \\ \text{s.a} \quad & y_i \left(\langle w, x_i \rangle + b \right) - 1 \geq 0 \quad i = 1, ..., n \end{aligned}$$

cuyo problema dual asociado quedaría:

$$\begin{aligned} \text{máx} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{s.a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \qquad\qquad i = 1, ..., n \end{aligned}$$

2.2. SVM Caso Lineal No Separable

En los problemas reales encontrar un conjunto con dos clases totalmente separables es escasamente probable, entre otras cosas por la existencia de *ruido* en los datos. La idea para tratar con este tipo de casos con *ruido* es introducir un conjunto de variables reales y positivas, *variables artificiales*, ξ_i, $i = 1, ..., n$, de forma que permitan algunos ejemplos no separables, es decir:

$$y_i \left(\langle w, x_i \rangle + b \right) \geq 1 - \xi_i \qquad \xi_i \geq 0, \quad i = 1, ..., n$$

De acuerdo con la expresión anterior los ejemplos con variable artificial nula corresponden a ejemplos separables, mientras que los que tengan asociada una variable artificial positiva son los llamados *ruido*, mal clasificados. La función a optimizar en este caso debe incluir estas variables artificiales de forma que controle el error en la clasificación.

De esta forma el nuevo problema de optimización a resolver nos quedaría:

$$\begin{aligned} \text{mín} \quad & \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{n} \xi_i \\ \text{s.a} \quad & y_i \left(\langle w, x_i \rangle + b \right) + \xi_i - 1 \geq 0 \quad i = 1, ..., n \\ & \xi_i \geq 0 \qquad\qquad i = 1, ..., n, \end{aligned}$$

donde C es una constante positiva a determinar de la que puede depender el clasificador.

El hiperplano definido tras resolver este problema se denomina hiperplano de separación de margen blando o *soft margin*. En el caso perfectamente separable el hiperplano obtenido se le denomina hiperplano de separación duro o *hard margin*.

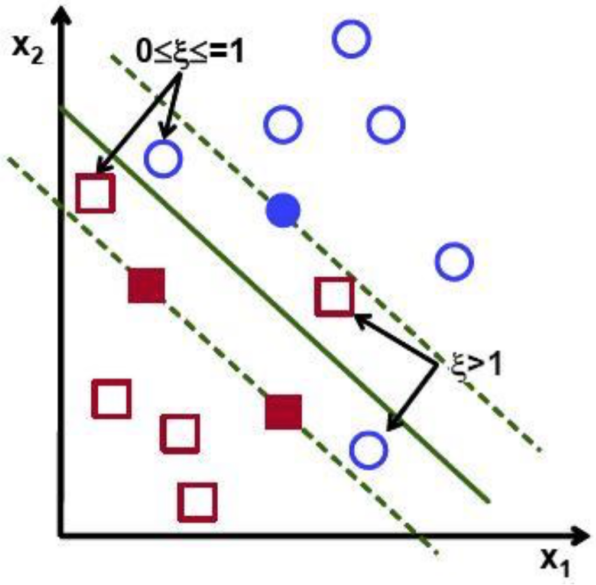


Figura 2.2: Ejemplo de problema no separable linealmente.

El problema dual asociado a este nuevo problema quedaría:

$$\begin{aligned} \text{máx} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ \text{s.a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \qquad\qquad i = 1, ..., n \end{aligned}$$

2.3. SVM Caso No Lineal

Es muy común en los problemas reales clasificar un conjunto donde los ejemplos no suelen ser separables. Aunque podemos utilizar el enfoque descrito en la sección anterior, sería deseable poder usar estrategias no lineales. Para resolver la clasificación mediante SVM de este tipo de conjuntos, se realiza la inmersión no lineal de nuestro conjunto en un espacio de dimensión mayor donde sí sean separables, y en este nuevo espacio buscar el hiperplano de separación. Este espacio se denomina espacio de características.

Sea $\Phi : \mathbb{X} \to \mathcal{F}$ la función que hace corresponder a cada punto de entrada x un punto en el espacio de características $\mathcal{F}$. Lo que pretendemos es encontrar el hiperplano de separación en este nuevo espacio $\mathcal{F}$. Este hiperplano en el espacio de características se transforma en una función no lineal que separa nuestro conjunto en el espacio original de entradas.

De esta forma, la función de decisión en el espacio de características viene dada por:

$$D(x) = \langle w, \Phi(x) \rangle + b$$

El problema primal asociado a este caso es similar al descrito previamente con la salvedad de que el hiperplano de separación se determina en el espacio de características, es decir:

$$\begin{aligned} \text{mín} \quad & \frac{1}{2} \parallel w \parallel^2 + C \sum_{i=1}^{n} \xi_i & (2) \\ \text{s.a} \quad & y_i \langle w, \phi(x_i) \rangle + \xi_i - 1 \geq 0 \qquad i = 1, ..., n \\ & \xi_i \geq 0 \qquad i = 1, ..., n \end{aligned}$$

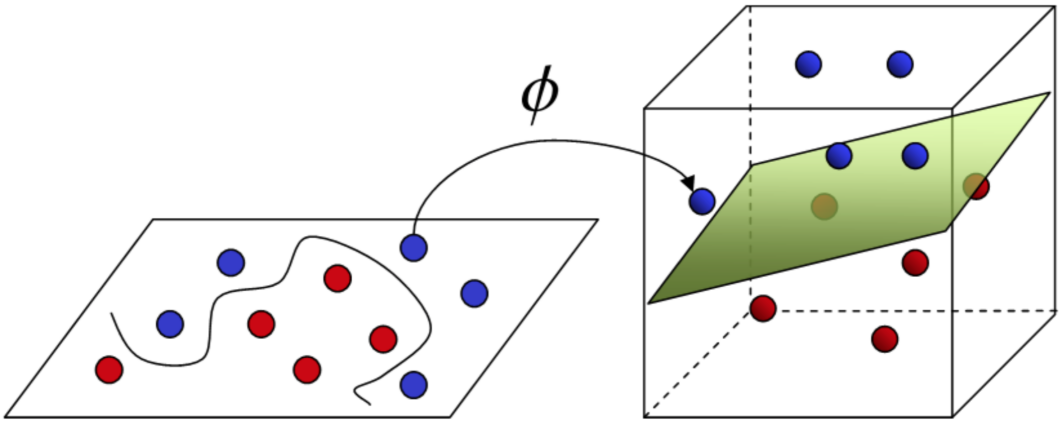


Figura 2.3: Ejemplo gráfico de una posible función Φ.

La complejidad del problema (2) depende de la dimensión del espacio de características, que suele ser alta. Por ello se considera el problema dual asociado, cuya complejidad depende del número de ejemplos.

En este caso el dual quedaría:

$$\begin{aligned} \text{máx} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \Phi(x_i), \Phi(x_j) \rangle \\ \text{s.a} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C \qquad i = 1, ..., n \end{aligned}$$

En el problema anterior se observa que no es necesario conocer las componentes ϕ_i de la función de inversión, sino sólo los productos escalares $\langle \Phi(x_i), \Phi(x_j) \rangle$. Para resolver este tipo de problemas se usa el *truco del kernel*, donde:

$$K(x_i, x_j) = \langle \Phi(x_i), \Phi(x_j) \rangle.$$

De esta forma podemos construir una función de decisión en el espacio original sin usar explícitamente la función de inversión Φ.

En el siguiente capítulo se tratará de manera más detallada el concepto de kernel en el contexto del SVR.

Capítulo 3

Support Vector Regression

Dado el buen rendimiento de las máquinas de vectores soporte para la clasificación, ¿por qué no aplicar esta teoría y estos conocimientos en otros ámbitos? La técnica de los vectores soporte es una herramienta universal para resolver problemas de estimación de funciones multidimensionales. Dado el buen resultado en el ámbito de la clasificación se plantea un problema similar para abordar la regresión.

En este caso, la idea es seleccionar el hiperplano regresor que mejor se ajuste a nuestro conjunto de datos de entrenamiento. Ahora no disponemos de clases para separar. La idea se basa en considerar una distancia margen ε, de modo que esperamos que todos los ejemplos se encuentren en una *banda* o *tubo* entorno a nuestro hiperplano, es decir, que disten una cantidad menor de ε del hiperplano. A la hora de definir el hiperplano sólo se consideran los ejemplos que disten más de ε de nuestro hiperplano. En este caso esos ejemplos serán los considerados como vectores soporte.

3.1. Funciones de Pérdida

La idea en este apartado para la obtención de la función de decisión óptima, es añadir unas variables reales y positivas que cuantifiquen el error cometido entre la aproximación y el valor real de cada ejemplo de nuestro conjunto.

En la clasificación mediante SVM se usa el margen para determinar la separación entre las dos clases de puntos. A mayor margen, mayor seguridad de que estamos ante un hiperplano de separación bueno. En la regresión nosotros no estamos interesados en la separación de puntos, pero pretendemos que la función determinada esté lo más próxima posible a los puntos. El análogo de lo que sería para clasificación el margen es la formación de una *banda* o *tubo* alrededor de la verdadera función de regresión. Los puntos no contenidos dentro del tubo se identifican con la positividad estricta de las variables artificiales asociadas.

Buscamos una función de regresión, en principio lineal, es decir, de la siguiente forma:

$$f(x) = (w_1 x_1 + ... + w_d x_d) + b = \langle w, x \rangle + b,$$

donde $w_i \in \mathbb{R} \quad \forall i = 1, ..., d$ y $b \in \mathbb{R}$.

Lo que pretendemos es definir una función de pérdida que ignore los errores asociados con los puntos que caen dentro de una banda que está a una cierta distancia de la función de regresión lineal. Es decir, si el punto verifica $|y - f(x)| \leq \varepsilon$ la función pérdida deberá anularse. Los ejemplos que disten exactamente ε de nuestra función se denominarán vectores soporte.

Siguiendo esta estrategia nosotros podemos definir las siguientes funciones de pérdida:

- $L_1^{\varepsilon}(y, f(x)) = \text{máx } \{0, |y - f(x)| - \varepsilon\}$
- $L_2^{\varepsilon}(y, f(x)) = \text{máx } \{0, (y - f(x))^2 - \varepsilon\}$

Dichas funciones son denominadas funciones de pérdida ε-insensible lineal y cuadrática respectivamente. En lo que sigue centraremos el estudio de la SVR tomando como función perdida la función de pérdida ε-insensible lineal, se puede hacer un estudio similar para la otra función de pérdida.

3.2. Funcional de Riesgo

Hasta ahora el algoritmo SV para la regresión puede parecer bastante extraño y casi no relacionado con otros métodos existentes de estimación de la función de regresión. Sin embargo una vez expresado con notación matemática más estándar se observa la conexión.

Para más simplicidad vamos a tener solo en cuenta solo al caso lineal, sabiendo que podría extenderse al caso no lineal utilizando el método del kernel.

Supongamos que disponemos de un conjunto $\mathcal{C} = \{(x_1, y_1), ..., (x_n, y_n)\}$, donde $x_i \in \mathbb{R}^d$ e $y_i \in \mathbb{R}$. Asumimos que este conjunto de datos se ha obtenido a partir de cierta función de probabilidad $P(x, y)$. Nuestro objetivo será encontrar una función f que minimice el funcional:

$$R[f] = \int c(x, y, f(x))\, dP(x, y)$$

Vamos a denotar como $c(x, y, f(x))$ a la función que penaliza los errores de estimación. Cuando nos refiramos a esta función lo haremos como función *coste*.

Basándonos en los datos empíricos y dado que no conocemos la medida de probabilidad, solo podemos usar los datos de S para la estimación de una función que minimiza $R[f]$. Una posible aproximación consiste en la sustitución de la integración por la estimación empírica para conseguir el llamado funcional de riesgo empírico:

$$R_{emp}[f] := \frac{1}{n}\sum_{i=1}^{n} c(x_i, y_i, f(x_i))$$

Un primer intento sería encontrar la función $f := \text{argmin}_{f\in H}(R_{emp}[f])$ para alguna hipotética clase H. Sin embargo si la clase H es muy amplia, hay una gran cantidad de ejemplos, cuando se trata de unos datos en espacios de muy alta dimensión esto puede no ser una buena idea, ya que se dará sobreajuste y por tanto malas propiedades de generalización. Por lo tanto le añadimos un termino de control de capacidad, que en el caso del algoritmo SV es $\|w\|^2$, lo que conduce al funcional de riesgo generalizado:

$$R_{reg}[f] := \frac{\lambda}{2}\|w\|^2 + R_{emp}[f]$$

donde $\lambda > 0$ es llamada la constante de regularización.

La pregunta que nos surge ahora es, ¿ Qué función de coste debemos usar en R_{reg} ? La función coste utilizada para el caso descrito de SVR es:

$$c(x, y, f(x)) = L_1^{\varepsilon}(y, f(x)) = |y - f(x)|_{\varepsilon},$$

con $|z|_{\varepsilon} = \text{máx}\{\ 0,\ |z| - \varepsilon\}$.

Las funciones de coste del tipo $|y - f(x)|_{\varepsilon}^{p}$ con $p > 1$ pueden no ser deseable, ya que el aumento superlineal conduce a una pérdida de las propiedades de robustez del estimador (véase ejemplos en [5]). En estos casos la derivada de la función coste puede crecer sin límite. Para $p < 1$ tendríamos una función de coste no convexa.

Para el caso $c(x, y, f(x)) = L_2^{\varepsilon}(y, f(x))$, es decir, para la función $\varepsilon-$insensible cuadrática, tendríamos una aproximación en forma de mínimos cuadráticos que, a diferencia de nuestra función de coste estándar del SV, conduce a una inversión de la matriz en lugar de un problema de programación cuadrática.

Por un lado vamos a pretender evitar el uso de funciones complejas ya que esto puede conducir a problemas de optimización difíciles, por otro lado podríamos usar una función de coste que se adapte mejor a los datos. Bajo el supuesto de que las muestran fuesen generadas por una dependencia funcional subyacente, más un ruido aditivo, es decir: $y_i = f_v(x_i) + \xi_i$ con densidad $p(\xi)$, la función de coste óptima un sentido de probabilidad máxima sería:

$$c(x, y, f(x)) = -\log p(y - f(x))$$

Algunos ejemplos de funciones de coste asociadas a modelos de densidad conocidos son:

	Funciones de Coste
ε-insensitive	$c(x, y, f(x)) = \lvert y - f(x)\rvert_\varepsilon$
Laplace	$c(x, y, f(x)) = \lvert y - f(x)\rvert$
Gauss	$c(x, y, f(x)) = (y - f(x))^2$
Huber	$c(x, y, f(x)) = \begin{cases} \frac{1}{2\sigma}(y - f(x))^2 & \text{si } \lvert y - f(x)\rvert \leq \sigma \\ \lvert y - f(x)\rvert - \frac{\sigma}{2} & \text{si } \lvert y - f(x)\rvert > \sigma \end{cases}$
Polinomial	$\frac{1}{p}\lvert y - f(x)\rvert^p$
Polinomial a trozos	$c(x, y, f(x)) = \begin{cases} \frac{1}{p\sigma^{p-1}}(y - f(x))^p & \text{si } \lvert y - f(x)\rvert \leq \sigma \\ \lvert y - f(x)\rvert - \sigma\frac{p-1}{p} & \text{si } \lvert y - f(x)\rvert > \sigma \end{cases}$

Cuadro 3.1: Tabla correspondiente a algunas funciones de coste asociadas a funciones de densidad comunes.

Las funciones coste deben ser funciones convexas. En caso de que una función no sea convexa habría que encontrar una aproximación convexa con el fin de hacer frente a la situación de manera eficiente. El requisito de convexidad se pide porque queremos asegurar la existencia y unicidad de un mínimo en los problemas de optimización, y que los óptimos locales sean globales [2].

3.3. Caso Lineal

Vamos a considerar en este caso un conjunto de igual estructura que en el caso de SVM, modificando el posible valor de la variable respuesta.

Sea el conjunto $\mathcal{C} = \{(x_1, y_1), ..., (x_n, y_n)\}$, donde $x_i \in \mathbb{R}^d$ e $y_i \in \mathbb{R}$. Así pues podríamos plantear el problema de forma análoga al caso de SVM pero con las mencionadas diferencias como:

$$\begin{aligned} \text{mín} \quad & \frac{1}{2} \parallel w \parallel^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{s.a} \quad & y_i - (\langle w, x_i\rangle + b) \leq \varepsilon + \xi_i \quad i = 1, ..., n \\ & (\langle w, x_i\rangle + b) - y_i \leq \varepsilon + \xi_i^* \quad i = 1, ..., n \\ & \xi_i \geq 0, \quad \xi_i^* \geq 0 \qquad\qquad i = 1, ..., n. \end{aligned}$$

La constante $C > 0$ determina el equilibrio entre la regularidad de f y la cuantía hasta la cual toleramos desviaciones mayores que ε. Consideraremos ξ_i y ξ_i^* las variables que controlan el error cometido por la función de regresión al aproximar el i-ésimo ejemplo.

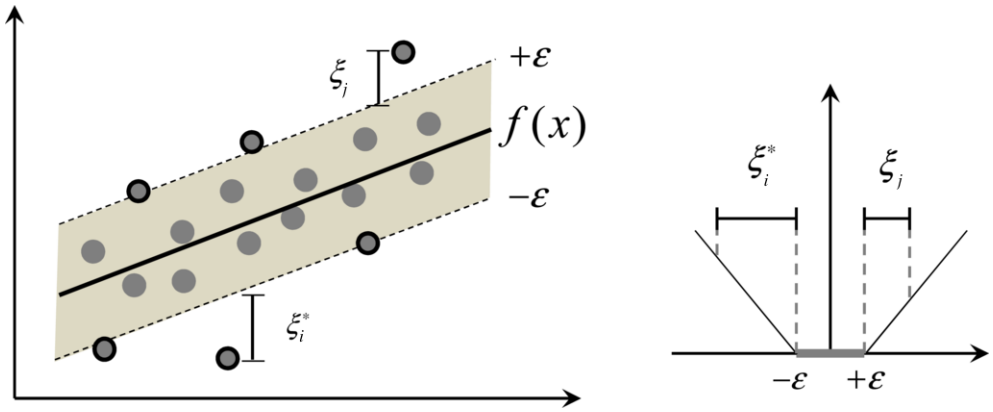


Figura 3.1: Ahora la idea es englobar a todos los ejemplos en una *banda* en torno a nuestra función de predicción f.

Así pues un valor muy grande de la constante C, en el caso límite ($C \to \infty$) estaríamos considerando que el conjunto está perfectamente representado por nuestro hiperplano predictor ($\xi_i \to 0$). Por contra, un número demasiado pequeño para C permitiría valores de ξ_i elevados, es decir, estaríamos admitiendo un número muy elevado de ejemplos mal representados.

Tras la obtención del problema primal pasamos a plantear el problema dual asociado. Para ello determinamos la función lagrangiana. La idea clave es construir una función de Lagrange con la función objetivo y las restricciones correspondientes, mediante la introducción de un conjunto de variables duales. Nuestro objetivo es obtener el problema dual a partir de las condiciones de dualidad fuerte [6]. Así definimos $L := L(w, b, \xi, \xi^*, \alpha, \alpha^*, \eta, \eta^*)$, donde w, b, ξ, ξ^* son las variables originales del problema y α, α^*, η, η^* son las variables duales asociadas a las restricciones. Esta función tiene un punto de silla respecto a las variables del primal. Una demostración de esto se puede encontrar en [6]. Con esto,

$$\begin{aligned} L \quad := \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) - \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \langle w, x_i\rangle + b) \\ & - \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle w, x_i\rangle - b) - \sum_{i=1}^{n}(\eta_i\xi_i + \eta_i^*\xi_i^*) \end{aligned}$$

lo que buscamos es hallar el:

$$\max_{\alpha,\alpha^*,\eta,\eta^*}\{\min_{w,b,\xi,\xi^*} L(w, b, \xi, \xi^*, \alpha, \alpha^*, \eta, \eta^*)\}.$$

Se entiende que las variables del dual son positivas. Ademas se deduce de la

condición de punto silla que las derivadas parciales respecto de las variables del primal deberían anularse para el óptimo.

$$\begin{aligned}
\delta_b L = &\quad \sum_{i=1}^{n}(\alpha_i^* - \alpha_i) &= 0 \qquad (3)\\
\delta_w L = &\quad w - \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\ x_i &= 0 \quad \Rightarrow w = \sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\ x_i\\
\delta_{\xi_i} L = &\quad C - \alpha_i - \eta_i &= 0 \quad \Rightarrow \eta_i = C - \alpha_i \geq 0\\
\delta_{\xi_i^*} L = &\quad C - \alpha_i^* - \eta_i^* &= 0 \quad \Rightarrow \eta_i^* = C - \alpha_i^* \geq 0
\end{aligned}$$

Como la única restricción tanto para η como para η^* es ser positivas, podemos expresarlas en función de α_i y α_i^*. Sustituyendo las ecuaciones anteriores en la función de Lagrange obtenemos:

$$\begin{aligned}
\max_{\alpha_i,\alpha_i^*} \ \{ & \frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j\rangle + C\sum_{i=1}^{n}(\xi_i + \xi_i^*)\\
& - \sum_{i=1}^{n}\alpha_i(\varepsilon + \xi_i - y_i + \langle\left(\sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\ x_i\right), x_i\rangle + b)\\
& - \sum_{i=1}^{n}\alpha_i^*(\varepsilon + \xi_i^* + y_i + \langle\left(\sum_{i=1}^{n}(\alpha_i^* - \alpha_i)\ x_i\right), x_i\rangle - b)\\
& - \sum_{i=1}^{n}((C - \alpha_i)\xi_i + (C - \alpha_i^*)\xi_i^*)\ \}
\end{aligned}$$

Simplificando la expresión anterior e imponiendo la restricción (3) aun no utilizada, obtenemos el siguiente problema de optimización dual:

$$\begin{aligned}
\text{máx} \quad & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j\rangle - \varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*)\\
& \text{s.a} \qquad \sum_{i=1}^{n}(\alpha_i - \alpha_i^*) = 0\\
& \qquad\qquad \alpha_i, \alpha_i^* \in [0, C]
\end{aligned}$$

A partir de estas expresiones, también estaríamos extrayendo una expresión de nuestra función de predicción:

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\langle x_i, x\rangle + b$$

De esta forma estaríamos obteniendo la función buscada sin depender la resolución del problema de la dimensión en la que se encuentra nuestros ejemplos de entrada y pasaría a depender únicamente de los vectores soporte.

Para completar nuestra función de regresión deberíamos calcular b para ello usamos las condiciones de complementariedad de holgura de Karush-Kuhn-Tucker

(KKT) [9]. Estas establecen que en la solución óptima el producto entre las variables de holgura y las restricciones duales deben anularse, es decir:

$$\alpha_i(\varepsilon + \xi_i - y_i + \langle w, x_i\rangle + b) = 0$$
$$\alpha_i^*(\varepsilon + \xi_i^* + y_i - \langle w, x_i\rangle - b) = 0$$

$$(C - \alpha_i)\xi_i = 0$$
$$(C - \alpha_i^*)\xi_i^* = 0,$$

esto nos permite deducir varias conclusiones. En primer lugar, solo los ejemplos (x_i, y_i) tales que $\alpha_i = 0$ ó $\alpha_i^* = 0$ estarían fuera de nuestro *tubo* o *banda* construido. Por otra parte $\alpha_i\alpha_i^* = 0$, es decir, no se pueden activar a la vez las dos variables duales asociadas a un mismo ejemplo. Por último en los casos que $\alpha_i, \alpha_i^* \in (0, C)$ tendríamos que la variable ξ_i, ξ_i^* correspondiente se debe anular, por lo que podríamos despejar el valor de b de las primeras restricciones.

$$b = y_i - \langle w, x_i\rangle - \varepsilon \quad \text{si} \quad \alpha_i \in (0, C)$$
$$b = y_i - \langle w, x_i\rangle + \varepsilon \quad \text{si} \quad \alpha_i^* \in (0, C)$$

3.4. Caso No Lineal

De igual forma que nos planteamos la búsqueda de la función de regresión como una función lineal, podíamos plantearnos el caso que quisiésemos una aplicación más general.

Consideramos el mismo conjunto de antes: $\mathcal{C} = \{(x_1, y_1), ..., (x_n, y_n)\}$, donde $x_i \in \mathbb{R}^d$ e $y_i \in \mathbb{R}$. Sea $\Phi : \mathbb{X} \to \mathcal{F}$ la función que hace corresponder a cada punto de entrada x un punto en el espacio de características $\mathcal{F}$, donde $\mathcal{F}$ es un espacio de Hilbert. Este espacio de características puede ser de dimensión elevada o incluso infinita.

Nuestro objetivo va a ser trasladar nuestros ejemplos a este nuevo espacio de características y aquí encontrar la función que mejor aproxime las imágenes de nuestro conjunto.

En este caso la función que estamos buscando será de la siguiente forma:

$$f(x) = \langle w, \Phi(x)\rangle + b$$

Plantearíamos el problema primal al igual que en el caso lineal a diferencias que en este caso no nos depende directamente de los ejemplos del conjunto sino de sus imágenes por una cierta función ϕ.

$$\begin{aligned} \text{mín} \quad & \frac{1}{2} \parallel w \parallel^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) \\ \text{s.a} \quad & y_i - \langle w, \phi(x_i)\rangle \leq \varepsilon + \xi_i \quad i = 1, ..., n \\ & \langle w, \phi(x_i)\rangle - y_i \leq \varepsilon + \xi_i^* \quad i = 1, ..., n \\ & \xi_i \geq 0, \quad \xi_i^* \geq 0 \qquad\quad i = 1, ..., n \end{aligned}$$

Como ya hemos dicho anteriormente, la complejidad de este problema va a depender de la dimensión en la que se encuentren nuestros ejemplos, y en este caso, tras ser transformados por la función ϕ, estos ejemplos podrían tener una dimension extremadamente alta, lo que complicaría en exceso la resolución de este problema primal.

Así pues, procederemos a construir el problema dual asociado. Como ya lo transformamos para el caso lineal, por analogía podríamos deducir el dual buscado.

$$\begin{aligned} \text{máx} \quad & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i-\alpha_i^*)(\alpha_i-\alpha_i^*)\langle\phi(x_i),\phi(x_j)\rangle - \varepsilon\sum_{i=1}^{n}(\alpha_i+\alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i-\alpha_i^*) \\ \text{s.a} \quad & \sum_{i=1}^{n} y_i(\alpha_i-\alpha_i^*) = 0 \\ & \alpha_i, \alpha_i^* \in [0, C] \end{aligned}$$

3.4.1. Truco del Kernel

¿Podríamos resolver este problema sin llegar a conocer explícitamente la función ϕ? La respuesta es que sí.

Tras plantear el problema dual, observamos que la función objetivo solo depende del producto interno de las imágenes de nuestros ejemplos. El algoritmo del truco del kernel es ampliamente utilizado en los algoritmos de cálculo de productos internos de la forma $\langle\Phi(x), \Phi(x')\rangle$ en el espacio de características $\mathcal{F}$.

El truco consiste en que, en lugar de calcular estos productos internos en $\mathcal{F}$, lo que realmente utilizamos computacionalmente, debido a su posible alta dimensionalidad, es definir una función *kernel*, $K : X \times X \to \mathbb{R}$ que asigna a cada par de elementos del espacio de entrada X, un valor real correspondiente al producto escalar de las imágenes de dichos elementos en el nuevo espacio $\mathcal{F}$, es decir,

$$K(x, x') = \langle\Phi(x), \Phi(x')\rangle$$

donde $\Phi : X \to \mathcal{F}$. Los primeros en aplicarlo para este tipo de problemas fueron Cortes y Vapnik en [1].

Ejemplo:

Consideramos un espacio de entrada bidimensional, $X \subseteq \mathbb{R}^2$ y sea Φ la función definida de la siguiente forma:

$$\Phi : \mathbf{x} = (x_1, x_2) \longmapsto \Phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2) \in \mathcal{F} = \mathbb{R}^3$$

A la hora de evaluar el producto interno en el espacio $\mathcal{F}$ nos queda:

$$\begin{aligned} \langle\phi(x), \phi(z)\rangle &= \langle(x_1^2, x_2^2, \sqrt{2}x_1x_2), (z_1^2, z_2^2, \sqrt{2}z_1z_2)\rangle \\ &= x_1^2z_1^2 + x_2^2z_2^2 + 2x_1x_2z_1z_2 \\ &= (x_1z_1 + x_2z_2)^2 \\ &= \langle x, z\rangle^2 \end{aligned}$$

Luego en este caso la función *kernel* sería:

$$K(x, z) = \langle x, z \rangle^2$$

Problema a resolver aplicando el kernel:

Una vez hayamos fijado el *kernel* a utilizar pasaríamos a resolver el problema de la forma:

$$\begin{aligned} \text{máx} \quad & -\frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i - \alpha_i^*)(\alpha_i - \alpha_i^*)K(x_i, x_j) - \varepsilon\sum_{i=1}^{n}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) \\ \text{s.a} \quad & \sum_{i=1}^{n} y_i(\alpha_i - \alpha_i^*) = 0 \\ & \alpha_i, \alpha_i^* \in [0, C], \end{aligned}$$

y la función de predicción:

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)K(x_i, x) + b$$

3.4.2. Matriz de Gram

Dado un conjunto de puntos $\mathcal{C} = \{x_1, ..., x_l\}$ la matriz de Gram se define como G cuyas entradas son: $g_{ij} = \langle x_i, x_j \rangle$. Si usamos una función *kernel* K para evaluar los productos internos en el espacio de características con la función característica ϕ. Las entradas asociadas a la matriz de Gram son: $k_{ij} = \langle \phi(x_i), \phi(x_j) \rangle$. En estos casos la matriz también es conocida como matriz kernel y presenta la forma característica:

$$K := \begin{pmatrix} K(x_1, x_1) & K(x_1, x_2) & \cdots & K(x_1, x_l) \\ K(x_2, x_1) & K(x_2, x_2) & \cdots & K(x_2, x_l) \\ \vdots & \vdots & \ddots & \vdots \\ K(x_l, x_1) & K(x_l, x_2) & \cdots & K(x_l, x_l) \end{pmatrix}$$

Para el caso que usemos una función kernel, esta matriz es simétrica por construcción y además es semidefinida positiva, es decir, si consideramos el caso general de la matriz kernel tenemos:

$$k_{ij} = K(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle, \text{con } i, j = 1, ..., l.$$

Dado $\alpha \in \mathbb{R}^l$ cualquiera se tiene:

$$\begin{aligned} v'\, K\, v &= \sum_{i,j=1}^{l} v_i\, v_j\, K_{ij} = \sum_{i,j=1}^{l} v_i\, v_j \langle \phi(x_i), \phi(x_j) \rangle \\ &= \left\langle \sum_{i,j=1}^{l} v_i\, \phi(x_i), \sum_{i,j=1}^{l} v_j\, \phi(x_j) \right\rangle \\ &= \left\| \sum_{i,j=1}^{l} v_i\, \phi(x_i) \right\|^2 \geq 0 \end{aligned}$$

3.4.3. Función Kernel

Un producto interno en el espacio de características tiene asociado un kernel equivalente en el espacio de entradas:

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

Hemos visto cómo construir una matriz de evaluaciones, aplicando la función núcleo a todos los pares un conjunto de entradas. La matriz resultante es semidefinida positiva.

Ahora vamos a introducir un método alternativo para demostrar que una función candidata a ser kernel, de verdad lo es. Esto representa una de las herramientas teóricas necesarias para la creación de nuevos y más complejos núcleos a partir de otros más simples. Una de las observaciones más importantes es la relación con las matrices semidefinidas positivas. Como hemos visto, la matriz kernel evaluada para cualquier conjunto de datos es semidefinida positiva.

Dada esta peculiaridad de las funciones kernel nos lleva a definir lo siguiente: una función $K : X \times X \to \mathbb{R}$ se dice finitamente semidefinida positiva, si es simétrica y las matrices formadas con cualquier conjunto finito del espacio X son semidefinidas positivas. Esta definición no requiere que X sea siquiera un espacio vectorial.

Teorema 3.1 *Sea una función $K : X \times X \to \mathbb{R}$, ya sea continua o con dominio finito. Se puede descomponer en:*

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

con cierta función característica ϕ en un espacio de Hilbert $\mathcal{F}$ aplicado a los argumentos y seguido de la evaluación del producto interno en $\mathcal{F}$ si y solo si K es finitamente semidefinida positiva.

Una prueba de este resultado la podemos encontrar en [14].

Dada una función K que satisfaga la propiedad anterior, es decir, que sea finitamente semidefinida positiva, llamamos al correspondiente espacio $\mathcal{F}_k$ como el espacio de Hilbert reproducido por el kernel, del inglés (RKHS).

3.4.4. Núcleo de Mercer

Ahora sí estamos en las condiciones óptimas para pasar al teorema de Mercer como consecuencia del análisis anterior.

El teorema de Mercer se utiliza generalmente para la construcción del espacio de características para un kernel válido. Dado que ya hemos logrado esto con la construcción (RKHS), nosotros no necesitamos el teorema de Mercer en sí mismo. Lo incluimos ya que define el espacio de características en términos de un vector de características explícito, en lugar de utilizar el espacio de funciones de nuestra construcción (RKHS).

Teorema 3.2 *Sea X un conjunto compacto en $\mathbb{R}^n$. Supongamos que K es una función continua y simétrica tal que el operador*

$$\begin{aligned} T_k &: L_2(X) \longrightarrow L_2(X) \\ (T_k f)(\cdot) &= \int_X K(\cdot, x)\, f(x)\, dx \qquad \text{es positivo.} \end{aligned}$$

Esto es: $$\int_{X\times X} K(x,z)\, f(x)\, f(z)\, dx\, dz \geq 0 \quad \forall\, f \in L_2(X),$$

entonces nosotros podemos desarrollar $K(x,z)$ en una serie uniformemente convergente, en $X \times X$, en términos de unas funciones ϕ_j, satisfaciendo $\langle \phi_j, \phi_i \rangle = \delta_{ij}$

$$K(x,z) = \sum_{j=1}^{\infty} \phi_j(x)\phi_j(z)$$

Además la serie $\sum_{j=1}^{\infty} \|\phi_j\|^2_{L_2(X)}$ es convergente.

La prueba de este resultado la podemos encontrar en [14].

3.4.5. Matriz Kernel

Dado un conjunto de datos de entrenamiento $C_e = \{x_1, ..., x_l\}$ y una función kernel $K(\cdot,\cdot)$ nosotros introducimos facilmente la matriz de Gram o en este caso también llamada matriz kernel con las entradas $K_{ij} = H(x_i, x_j)$. En la sección anterior hemos visto que si la función K es un kernel válido la matriz kernel asociada a cada conjunto de entrenamiento C_e tiene que ser semidefinida positiva. Lo que llamabamos ser la función finitamente semidefinida positiva (FPSF).

Este hecho nos permite manipular los kernels sin tener en cuenta necesariamente el espacio de características correspondiente. A condición de que mantengamos la propiedad, tenemos garantizado un kernel válido, es decir, que existe un espacio de características para la cual es la función núcleo correspondiente. Este razonamiento sobre la medida de similitud implica que la función del núcleo puede ser más natural que la realización de una construcción explícita de su espacio de características.

La modularidad intrinseca de máquinas kernel también significa que cualquier función kernel se puede utilizar siempre que produzca matrices simétricas y semidefinidas positivas, y podemos aplicar cualquier algoritmo de núcleo, siempre y cuando podamos aceptar como entrada una matriz adecuada a nuestro conjunto de datos.

En vista de nuestra caracterización de los núcleos, en cuanto a la propiedad (FPSF), queda claro que la matriz del núcleo es el ingrediente básico en la teoría de los métodos con kernels. Esta matriz contiene toda la información disponible para llevar a cabo la etapa de aprendizaje, con la única excepción de las etiquetas de salida en el caso de aprendizaje supervisado. Vale la pena tener en cuenta que es solo a través de la matriz kernel que el algoritmo de aprendizaje obtiene la

información acerca de la elección del espacio de características o modelo, y de hecho los propios datos de entrenamiento.

La propiedad (FPSF) también puede utilizarse para justificar etapas de procesamiento intermedias destinadas a mejorar la representación de los datos, y por lo tanto el rendimiento general del sistema a través de la manipulación de la matriz kernel antes de se pase a la máquina de aprendizaje.

Otro importante aspecto de la caracterización de los núcleos válidos en términos de la propiedad (FPSF) es que es la misma que debe cumplir para cualquier tipo de entradas. No se exige que las entradas deban ser vectores reales, por lo que caracterización se aplica sea cual sea el tipo de los datos, ya sean secuencias, estructuras discretas, imágenes, series de tiempo etc. Siempre que la matriz kernel correspondiente a cualquier conjunto de información sea semidefinida positiva, el núcleo calcula el producto interno después de la proyección de los pares en algún espacio de características.

3.4.6. Construcción del Núcleo

La caracterización de las funciones del núcleo y las matrices kernel de los apartados anteriores no es solo útil para decidir si un determinado candidato es un kernel válido. Una de sus principales consecuencias es que puede ser utilizado para justificar una serie de normas para la manipulación y combinación de kernel, fácil de obtener, más complejos y más útiles. En otras palabras este tipo de operaciones en uno o más núcleos se pueden realizar y preservar la propiedad (FPSF). Diremos que la clase de kernels es cerrada bajo tales operaciones. Estos incluyen operaciones sobre las funciones y operaciones directamente en la matriz del núcleo. En tanto que se pueda garantizar que el resultado de una operación mantiene la propiedad (FPSF) seguirá incorporando los datos en un espacio de características modificado por la operación elegida.

Vamos a probar que los kernels satisfacen una serie de propiedades de clausura. Estas operaciones la utilizaremos para crear nuevos kernels más complejos y útiles a partir de kernels más sencillos.

Teorema 3.3 *Dados $K_1 : X \times X \longrightarrow \mathbb{R}$ y $K_2 : X \times X \longrightarrow \mathbb{R}$ dos funciones kernel, con $X \subseteq \mathbb{R}^n$, $a \in \mathbb{R}$ $a \geq 0$, $f(\cdot)$ una función real en X, $\phi : X \longrightarrow \mathbb{R}^n$, un tercer kernel $K_3 : \mathbb{R}^n \times \mathbb{R}^n \longrightarrow \mathbb{R}$ y $B \in \mathcal{M}^{n \times n}$ simétrica y semidefinida positiva.*

Entonces las siguientes operaciones definen nuevos kernels:

i) $K(x, z) = K_1(x, z) + K_2(x, z)$

ii) $K(x, z) = a\, K_1(x, z)$

iii) $K(x, z) = K_1(x, z)\, K_2(x, z)$

iv) $K(x, z) = f(x)\, f(z)$

v) $K(x,z) = K_3(\phi(x), \phi(z))$

vi) $K(x,z) = x'\, B\, z$

Prueba:

Dado un conjunto finito de puntos $C = \{x_1, ..., x_l\}$, siendo K_1 y K_2 las correspondientes matrices kernel obtenidas de este conjunto de datos con las funciones $K_1(\cdot,\cdot)$ y $K_2(\cdot,\cdot)$ respectivamente. Consideramos un vector real cualquiera $\alpha \in \mathbb{R}$. Recordemos que una matriz es semidefinida positiva $\Longleftrightarrow \alpha' K \alpha \geq 0$.

i) Tenemos que $\alpha'(K_1 + K_2)\alpha = \alpha' K_1 \alpha + \alpha' K_2 \alpha \geq 0 \Longrightarrow K_1 + K_2$ es una matriz semidefinida positiva y por tanto $(K_1 + K_2)(\cdot,\cdot)$ es una función kernel.

ii) Análogamente al caso anterior tenemos: $\alpha' a K_1 \alpha = a \alpha' K_1 \alpha \geq 0 \Longrightarrow a K_1(\cdot,\cdot)$ es una función kernel.

iii) Sea $K = K_1 \otimes K_2$ el producto de Schur de las matrices K_1 y K_2, es decir, el elemoento K_{ij} de la matriz K es de la forma $K_{1ij}K_{2ij}$. El producto de Schur entre dos matrices semidefinidas positivas es en sí mismo semidefinido positivo. Luego $\alpha' K \alpha \geq 0 \Longrightarrow K(\cdot,\cdot)$ es una función kernel como queríamos.

iv) Considerando la función unidimensional como función asociada

$\phi : x \longrightarrow f(x) \in \mathbb{R} \Longrightarrow K(\cdot,\cdot)$ es el correspondiente kernel.

v) Como K_3 es un kernel, la matriz obtenida de restringir K_3 a los puntos $\{\phi(x_1), ..., \phi(x_l)\}$ es semidefinida como necesitamos $\Longrightarrow K(\cdot,\cdot)$ es una función kernel como queríamos.

vi) Consideramos la diagonalización de $B = V'\, \Lambda\, V$ con V ortogonal y Λ diagonal con los autovalores que no serán negativos por ser B semidefinida positiva.

Tomamos la matriz A de forma que $A = \sqrt{\Lambda}\, V$. Entonces tenemos:

$K(x,z) = x'\, B\, z = x'\, V'\, \Lambda\, V\, z = x'\, A'\, A\, z = \langle A\, x, A\, z \rangle$

que sería el producto interno.

A raíz de esta última propiedad tenemos que, con el simple hecho de tener una matriz semidefinida positiva de dimensiones adecuadas, estamos implícitamente ante un posible núcleo.

Los resultados anteriores nos muestran cómo podemos crear núcleos a partir de núcleos ya existentes mediante operaciones simples. Nuestro enfoque ha sido demostrar que las nuevas funciones son nuevos kernels demostrando que su matriz asociada cumplía la propiedad (FPSF). Esto es suficiente para verificar que la función es un núcleo y por tanto demuestra que existe una inmersión que nos lleva a un espacio de características donde la función calcula el producto interno correspondiente. A menudo esta información es suficiente para construir un núcleo adecuado para una aplicación particular. A veces, sin embargo, es útil entender el efecto de la combinación del núcleo en la estructura de la función del espacio correspondiente.

Observemos el cambio que sufre el espacio de características cuando aplicamos las operaciones mencionadas anteriormente.

En la prueba del apartado iv) se usa una construcción del espacio característico. En el apartado ii) es un simple reescalado por el vector $\sqrt{a}$. En la suma de dos kernels del apartado i) el vector de características es la concatenación de los correspondientes vectores, es decir:

$$\phi(x) = [\phi_1(x), \phi_2(x)]$$

Por tanto la formación del núcleo quedaría:

$$\begin{aligned} K(x,z) &= \langle \phi(x), \phi(z) \rangle = \langle [\phi_1(x), \phi_2(x)], [\phi_1(z), \phi_2(z)] \rangle \\ &= \langle \phi_1(x), \phi_1(z) \rangle + \langle \phi_2(x), \phi_2(z) \rangle \\ &= K_1(x,z) + K_2(x,z) \end{aligned}$$

En el apartado iii) las funciones de inmersión son el producto de las funciones de inmersión de uno y otro kernel. Los ϕ_{ij} que nos quedarían serían:

$$\phi(x)_{ij} = \phi_1(x)_i \phi_2(x)_j \text{ con } i = 1, ..., N_1, j = 1, .., N_2,$$

donde N_i es la dimensión del espacio de características correspondiente a cada núcleo.

La representación del producto interno sería:

$$\begin{aligned} K(x,z) = \langle \phi(x), \phi(z) \rangle &= \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \phi(x)_{ij} \phi(z)_{ij} \\ &= \sum_{i=1}^{N_1} \phi_1(x)_i \phi_1(z)_i \sum_{j=1}^{N_2} \phi_2(x)_j \phi_2(z)_j \\ &= K_1(x,z) K_2(x,z) \end{aligned}$$

La definición del espacio construido en estos casos puede parecer depender de la elección del sistema de coordenadas, ya que hace uso de las funciones especificas usadas. El hecho de que el núcleo pueda expresarse simplemente en término de las bases de los kernels, muestra que de hecho es invariante a esta elección.

Para el caso que tengamos la potencia de un solo núcleo: $K(x,z) = K_1(x,z)^s$, nosotros obtenemos inductivamente que el correspondiente espacio de características sería:

$$\phi_i(x) = \phi_1(x)_1^{i_1} \phi_1(x)_2^{i_2} \cdots \phi_1(x)_N^{i_N}$$

donde $\mathbf{i} = (i_1, ..., i_N) \in \mathbb{N}^N$ satisfaciendo $\sum_{j=1}^{N} i_j = s$

Operaciones con Matrices

En la sección anterior hemos tratado con funciones kernel, pero no en todos los casos disponemos de las funciones en sí mismas: Hay casos en los que solo disponemos de sus respectivas matrices kernel.

También se puede transformar el espacio de características mediante la realización de operaciones sobre la matriz kernel, siempre que mantengamos la matriz simétrica y semidefinida positiva. Este tipo de transformaciones plantea la cuestión de cómo evaluar el núcleo en puntos diferentes a los x_i.

En algunos casos es posible que hayamos construido la matriz kernel tanto en los puntos de entrenamiento, como en los de test, de manera que la matriz transformada contiene toda la información que vayamos a exigir. En otros caso, la transformación de la matriz kernel corresponde a una transformación computable en el espacio de características, por lo tanto, permite el cálculo del kernel en los puntos de test. Además de estos problemas computacionales, también existe el peligro de que adaptando el núcleo basándonos en su matriz kernel, nuestro ajuste sea demasiado dependiente del conjunto de datos de entrenamiento y no se adapte bien al conjunto de datos test. Este problema también es conocido como problema de sobreajuste.

De momento ignoraremos estos problemas y mencionaremos algunas transformaciones que pueden ser útiles en diferentes contextos donde es posible explicar el correspondiente efecto en el espacio de características.

- Transformaciones simples: Hay algunas transformaciones muy simples que tienen un significado muy práctico. Por ejemplo sumar una constante a todas las entradas de la matriz, corresponde a la adición de una constante de inmersion. Otra operación muy simple es sumar una constante a la diagonal de la matriz de Gram. Esto corresponde a añadir una nueva constante característica para cada entrada, por tanto aumenta la independencia de todas las entradas.

- Centrado de datos: El centrado de datos en el espacio de características es una transformación más compleja, pero que de nuevo puede ser realizado por las operaciones de la matriz kernel. El objetivo es mover el origen del espacio de características al centro de masas de los ejemplos de entrenamiento. Además la elección del centro de masas puede ser caracterizado como el origen tal que la suma de las normas de los puntos sea mínima. Como la suma de las normas es la traza de nuestra matriz kernel, que equivale a la suma de sus valores propios. Un mayor detalle de este cambio en la matriz kernel podemos hallarlo en [14].

- Blanqueamiento: Si una aproximación en una dimensión baja no logra captar los datos con suficiente precisión, nosotros podemos encontrar la descomposición en valores propios, útil con el fin de modificar la escala del espacio de características mediante a un ajuste del tamaño de los autovalores. Esta técnica es conocida como blanqueamiento.

El ejemplo final de ajustar los autovalores para crear un kernel que se ajuste mejor a los datos usa usualmente correspondientes etiquetas o salidas. Nosotros podemos ver estas operaciones en la primera fase del aprendizaje en la que se selecciona el espacio de características más adecuado para los datos. Como muchos algoritmos tradicionales de aprendizaje, los métodos kernel mejoran su rendimiento cuando los datos son preprocesados y se seleccionan las características adecuadas.

La sección del kernel puede verse como la selección de la topología correcta para el espacio de entradas, es decir, una topología que, o bien codifica correctamente nuestro conocimiento previo relativo a la similitud entre los puntos o bien aprende la topología más adecuada de la formación del conjunto. La visualización de los kernels como la definición de una topología sugiere que deberíamos hacer uso de los conocimientos previos sobre invariantes en el espacio de entradas. Por ejemplo, la traslación y rotación de caracteres en el espacio de entradas dejan sus etiquetas sin sufrir cambios, lo que significa que estas imágenes transformadas, aunque distantes en la métrica original van a volver a acercarse en la topología definida por el núcleo. En la parte III del libro [14] se puede encontrar una serie de métodos para la creación de núcleos para los diferentes tipos de datos.

3.4.7. Tipos de Kernel

Hasta ahora hemos visto cómo comprobar que una función cumple los requisitos necesarios para ser una función kernel, y varias formas de expresar los kernels y como trabajar con ellos cuando partimos de kernels previos. En la práctica se suele utilizar una serie de funciones kernels a las que se les ha estudiado sus propiedades y se ha comprobado que hay un gran número de casos en los cuales dichas funciones son aplicables.

A continuación se muestra una lista con kernels muy utilizados en la práctica:

Polinomial

Se define el kernel polinomial derivado de K_1 como:

$$K(x, z) = p(K_1(x, z))$$

donde $p(\cdot)$ es un polinomio con coeficientes positivos cualquiera. Frecuentemente nos referimos al caso especial:

$$K_d(x, x') = (\langle x, x' \rangle + R)^d$$

Definido sobre un espacio vectorial X de dimensión n, donde R y d son parámetros. El desarrollo del kernel polinomial K_d usando el teorema binomial nos da:

$$K_d(x, x') = \sum_{s=0}^{d} \binom{d}{s} R^{d-s} \langle x, z \rangle^s$$

Nota: La dimensión del espacio de características para el kernel polinomial $K_d(x, x') = (\langle x, x' \rangle + R)^d$ es

$$\binom{n+d}{d}$$

Función de Base Radial Gaussiana

Las funciones con base radial reciben gran atención, más concretamente la forma gaussiana. Los kernels gaussianos son los kernels más ampliamente utilizados y se han estudiado con mucho detalle en campos relacionados.

$$K(x, x') = \exp\left(-\gamma \|x - x'\|^2\right)$$

A veces no nos restringimos a calcular la distancia euclídea en el espacio de entradas. Si por ejemplo $K_1(x, z)$ es un kernel con correspondiente función relacionada ϕ_1 con espacio de características $\mathcal{F}_1$ nosotros podemos crear el kernel gaussiano en $\mathcal{F}_1$:

$$\|\phi_1(x) - \phi_1(z)\|^2 = K_1(x, x) - 2K_1(x, z) + K_1(z, z),$$

dando el derivado kernel gaussiano como:

$$K(x, z) = \exp\left(-\gamma(K_1(x, x) - 2K_1(x, z) + K_1(z, z))\right)$$

El parámetro γ controla la flexibilidad del kernel de forma similar que el grado d en los kernels polinomiales.

Función de Base Radial Exponencial

Las funciones de base radial exponencial son de la forma:

$$K(x, x') = exp\left(- \frac{\|x - x'\|}{2\sigma} \right)$$

All-subsets Kernel

Como un ejemplo de una combinación diferente de características consideramos un espacio con mapa característica ϕ_A para cada subconjunto $A \subseteq \{1, 2, ..., b\}$ de entradas, incluido el conjunto vacío. Usamos la funcion:

$$\phi_i(x) = x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$$

Con la restricción $\vec{i} = (i_1, ..., i_n) \in \{0, 1\}^n$. La función asociada ϕ_A esta dada por:

$$\phi_A(x) = \prod_{i \in A} x_i$$

Esto genera todos los monomios característicos para todas las combinaciones de diferentes indices hasta n, pero aquí, al contrario que en el caso polinomial, cada factor en el monomio tiene grado 1.

En definitiva el $all - subsets$ kernel es definido por la función ϕ:

$$\phi : x \longmapsto (\phi_A(x))_{A \subseteq \{1,\dots,n\}}$$

con el correspondiente kernel $K_{\subseteq}(x, z)$ dado por:

$$\begin{aligned} K_{\subseteq}(x,z) =& \langle \phi(x), \phi(z) \rangle = \\ &= \sum_{A \subseteq \{1..n\}} \prod_{i \in A} x_i z_i \end{aligned}$$

ANOVA Kernels

Tanto el kernel polinomial, como all-subsets kernel, tienen limitado el control de qué características usar y la forma de ponderar sus parámetros. Nosotros ahora presentamos un método que permite una mayor libertad en la especificación del conjunto de monomios.

El ANOVA kernel K_d es como el all-subsets, excepto que se restringe a subconjuntos cardinalidad d. Podemos usar la notación $\mathbf{x^i}$ para la expresión $x_1^{i_1} x_2^{i_2} \cdots x_n^{i_n}$, donde $\vec{i} = (i_1, ..., i_n) \in \{0, 1\}^n$ y además la restricción:

$$\sum_{j=1}^{n} i_j = d$$

para el caso $d = 0$ hay una característica con valor constante 1 correspondiente al conjunto vacío. La diferencia entre el ANOVA y el polinomial es la exclusión de las coordenadas repetidas. Para más información acerca de la historia de este método ver la sección 9.10 de [14].

La función característica del núcleo ANOVA de grado d viene dado por:

$$\phi_d : x \longmapsto (\phi_A(x))_{|A|=d}$$

donde para cada subconjunto A es definido por:

$$\phi_A(x) = \prod_{i \in A} x_i = \mathbf{x}^{\mathbf{i}_A}$$

donde $\mathbf{i}_A$ es la función indicador del conjunto A.

La dimensión del resultante ϕ_A es claramente $\binom{n}{d}$, ya que es el número de tales subconjuntos, mientras que el producto interno resultante viene dado por:

$$\begin{aligned} K_d(x,z) = \langle \phi_d(x), \phi_d(z) \rangle &= \sum_{|A|=d} \phi_A(x)\phi_A(z) \\ &= \sum_{1 \le i_1 < i_2 < \cdots < i_d \le n} (x_{i_1} z_{i_1})(x_{i_2} z_{i_2}) \cdots (x_{i_d} z_{i_d}) \\ &= \sum_{1 \le i_1 < i_2 < \cdots < i_d \le n} \prod_{j=1}^{d} x_{i_j} z_{i_j} \end{aligned}$$

Perceptrón Multicapa ó Sigmoidal

El perceptrón más establecido, con una capa oculta, también tiene una representación válida en los kernels,

$$K(x, x') = \tanh(\gamma\langle x, x'\rangle + R)$$

para ciertos valores escalares γ y R que aquí serían los parámetros de nuestro kernel.

Splines

Los Splines son una opción muy usada en el modelado debido a su flexibilidad. Un Spline finito, de orden κ con N nodos situados en τ_s es de la forma:

$$K(x, x') = \sum_{r=0}^{\kappa} x^r x'^r + \sum_{s=1}^{N} (x - \tau_s)_+^{\kappa} (x' - \tau_s)_+^{\kappa} d\tau$$

Se define un Spline infinito en el intervalo [0,1) como:

$$K(x, x') = \sum_{r=0}^{\kappa} x^r x'^r + \int_0^1 (x - \tau_s)_+^{\kappa} (x' - \tau_s)_+^{\kappa} d\tau$$

En el caso particular de que $\kappa = 1$, (S_1^{∞}) el kernel está dado por:

$$K(x, x') = 1 + \langle x, x'\rangle + \frac{1}{2}\langle x, x'\rangle \min(x, x') - \frac{1}{6}\min(x, x')^3,$$

donde la solucion es cúbica a trozos. En principio los kernels splines son kernels univariantes, pero gracias a la propiedad de que a partir del producto tensorial de kernels se obtiene un nuevo kernel, podemos construir kernels splines multidimensionales.

Series de Fourier

Las series de Fourier consideran una expansión en un espacio de caracteristicas $2N + 1$ dimensional. Este kernel esta definido en el intervalo [-$\frac{\pi}{2}$, $\frac{\pi}{2}$].

$$K(x, x') = \frac{\sin(N + \frac{1}{2})(x - x')}{\sin(\frac{1}{2}(x - x'))}$$

Al igual que los kernels splines, pueden usarse de base para construir kernels multidimensionales.

3.4.8. Kernels Asimétricos

Hay veces que, dado el conjunto $\mathcal{C} = \{x_1, ..., x_n\}$ que representan l objetos en $\mathbb{R}^d$, solo disponemos de una medida de similitud entre los objetos no simétrica, que podemos acumular en una matriz $S \in \mathcal{M}^{n \times n}$. Como hemos dicho que la medida de similitud no es simétrica, $s_{ij} \neq s_{ji}$ para algún par i,j.

Algunos ejemplos reales en los que se obtiene una medida de similitud no simétrica pueden ser:

- Página web que enlaza a otra; una puede enlazar a la otra sin que esta enlace sobre la misma.
- Niños en la clase que quieran estar sentados juntos; dos niños pueden no tener la misma preferencia en estar sentados juntos.
- Aparición en diferentes textos, la palabra i puede aparecer en textos donde aparece la palabra j, pero no al contrario.

Tanto para la clasificación como para la regresión las funciones de decisión son respectivamente de la forma:

$$f_1(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b \qquad f_2(x) = \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) K(x_i, x) + b$$

En caso de que la matriz K propuesta a ser matriz kernel sea asimétrica procederemos mediante una serie de técnicas para la simetrización de la correspondiente matriz.

Interpretación de Asimetría

Hay una particular forma de elegir s_{ij}, que denotaremos con $\wedge$, y definimos como:

$$s_{ij} = \frac{|x_i \wedge x_j|}{|x_i|} = \frac{\sum_{k=1}^{n} |\min(x_{ik}, x_{jk})|}{\sum_{k=1}^{n} |x_{ik}|}$$

En particular $s_{ij} \geq 0$ y $s_{ij} \leq 1$

Asumimos que acumulamos nuestro conjunto de datos C en una matriz de datos X.

Esta medida es útil para diversos tipos de ejemplos. Para el caso de que la matriz X sea una matriz de términos por documentos, la medida $|x_i|$ mediría el número de documentos indexados por el término i, y $|x_i \wedge x_j|$ el número de documentos indexados por ambos. Por lo tanto s_{ij} puede ser interpretado como el grado en el que el tema representado por el término i es un subconjunto del tema representado por el término j. Esta medida numérica de subsethood es debida a Kosko [7].

En el caso de ser una matriz de cocitación $|x_i|$ es el número de citas recibidas por el autor o página web i u $|x_i \wedge x_j|$ son los autores o webs que se citan mutuamente. Este tipo de problemas tienen en común que las normas de los individuos calculadas por $|x_i|$ siguen la ley de Zipf [12]: hay algunos individuos con unas normas muy grandes (muy citados) y hay una gran cantidad de individuos con normas muy pequeñas.

Esta asimetría puede interpretarse como una especie de jerarquía de los individuos que se organizan en una especie de árbol. En lo alto se encuentran los individuos con grandes normas que corresponden a individuos muy famosos. En la base se encuentran individuos con norma pequeña que son individuos raros.

Con este tipo de norma las matrices de similitud que se generan son asimétricas y frente a este tipo de matrices se usan unos métodos para poder obtener unas

matrices kernels con todas las propiedades necesarias. Describimos a continuación algunos métodos conocidos:

1) Una forma sencilla para lograr la simetrización de nuestra matriz es considerar la matriz de valores medios. De la descomposición:

$$S = \frac{1}{2}(S + S^T) + \frac{1}{2}(S - S^T)$$

nos quedamos con la parte simétrica, quedandonos con la matriz

$$K = \frac{1}{2}(S + S^T).$$

Sin embargo de esta forma despreciamos información importante, además nosotros ignoramos la parte asimétrica.

2) El método sugerido por Schölkopf [13] consiste en tomar como matriz kernel $K = S^T S$. Este método gana sentido cuando tenemos matrices de cocitación porque $K_{ij} = 1$ cuando hay un único k tal que $S_{ki} = S_{kj} = 1$, es decir, existe un autor que cita simultaneamente a i y a j. Sin embargo vamos a perder un caso de similitud que se produce cuando dos autores citan a su vez a un tercero. Esta información la recoge SS^T.

3) Para el caso de la clasificación podemos utilizar la etiqueta del conjunto al que corresponden para la construcción de nuestra matriz kernel. La construcción de la matriz K la hace a partir de la matriz de similitud dada de la forma:

$$K = \{k_{ij}\}_{i,j=1}^{n}, \quad k_{ij} = \begin{cases} \text{máx}(s_{ij}, s_{ji}) & \text{si } x_i, x_j \in C_1 \text{ ó } x_i, x_j \in C_2 \\ \text{mín}(s_{ij}, s_{ji}) & c.c \end{cases}$$

siendo C_1 y C_2 las dos clases diferenciadas a clasificar. Este método es denominado pick-out. De esta forma conseguimos que la matriz K sea simétrica. Para una mayor información de porque consideramos la matriz kernel de esta forma podemos ayudarnos de [11].

Sin embargo con estas transformaciones nos aseguraríamos la simetría de la matriz, pero no necesariamente cumple la propiedad semidefinida positiva. En este caso sustituiríamos la matriz K por $K + \lambda I$, con $\lambda > 0$ suficientemente grande, para que los autovalores se hagan no negativos.

3.5. Estimación de Parámetros del SVR

Los problemas primal y dual descritos anteriormente, dependen de una serie de parámetros que pueden hacer variar la solución. Además, cuando introducimos el kernel, éste depende a su vez de unos parámetros de los cuales depende nuestra solución. Es necesario dar un procedimiento de selección de dichos parámetros. El procedimiento más simple y utilizado consiste en definir una malla sobre el espacio de los parámetros y seleccionar aquellos valores de los parámetros que proporcionan mejores estimadores del error esperado de predicción. Para ello hace falta tener un procedimiento de estimación de dicho error esperado, como se propone a continuación.

3.5.1. Validación Cruzada

Probablemente el método más simple y más ampliamente utilizado para la estimación del error de predicción, sea la validación cruzada. Este método estima directamente la generalización del error cuando se aplica el método, de función de predicción $\hat{f}(x)$, a una muestra de test. Se denota por $Err = E[L(y, \hat{f}(x))]$, siendo $L(y, \hat{f}(x))$ una función de perdida adecuada.

Idealmente, si tuviésemos un conjunto suficientemente grande, podríamos seleccionar un conjunto de validación para evaluar nuestro modelo de predicción. Puesto que a menudo el conjunto no es lo suficientemente grande como nosotros requeriríamos esto no suele ser posible. En casos en los que no dispongamos un conjunto amplio, nosotros podemos usar la validación cruzada para evaluar nuestro predictor. Cuando aplicamos la validación cruzada con k-pliegues, subdividimos nuestro conjunto en k partes de aproximadamente el mismo tamaño. Utilizamos $k-1$ partes para ajustar el modelo y la restante para validarlo. Realizamos este mismo procedimiento k veces, seleccionando cada vez un conjunto de validación distinto. En cada caso nosotros calculamos el error de predicción del modelo y por último promediamos estos errores.

En otras palabras, sea $\kappa : \{1, ..., n\} \longmapsto \{1, ..., k\}$ una función que indica la partición a la que se le asigna i-ésima observación por la aleatorización. Para $j \in \{1, ..., n\}$ denotamos por $\hat{f}^{-\kappa(j)}(x)$ la función ajustada con el conjunto de datos a excepción del subconjunto $\kappa(j)$-ésimo. La correspondiente estimación de error mediante la validación cruzada con k-pliegues es:

$$CV(\hat{f}) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{-\kappa(i)}(x_i)).$$

Las elecciones más comunes son $k = 5$ y $k = 10$. El caso $k = n$ es conocido como *leave-one-out*. En este caso $\kappa(i) = i$, y la función de predicción para el caso i-ésimo se realiza con la totalidad del conjunto excepto el i-ésimo elemento. Esta validación cruzada es muy costosa, y por eso se reserva para los casos donde el conjunto de datos es muy pequeño. Hay algunos casos especiales en los que es beneficioso utilizar otros k-pliegues. Un estudio sobre la elección de estos k lo podemos encontrar en [4].

En nuestro caso, como utilizamos la validación cruzada para la estimación de los parámetros de nuestro predictor, la situación que nos queda es la siguiente. Sea θ nuestro conjunto de parámetros a estimar. La estimación de error sería:

$$CV(\hat{f}, \theta) = \frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{f}^{-\kappa(i)}(x_i, \theta)).$$

La función $CV(\theta)$ nos proporciona una estimación de la curva de error test, y buscamos el parámetro $\hat{\theta}$ que minimice dicha función. Nosotros elegiríamos como función predictora aquella formada con estos parámetros, es decir, $f(x, \hat{\theta})$ ajustada con todo el conjunto de datos.

Capítulo 4

Support Vector Regression en R

En la actualidad existe una gran diversidad de repositorios web y de paquetes software de libre distribución dedicados a la implementación de SVM, SVR y muchas de sus variantes.

Para apreciar el funcionamiento práctico de estos algoritmos, practicaremos la regresión mediante SVR a ciertos datos muestrales obtenidos del repositorio UCI Machine learning repository [8].

Para abordar este tipo de regresión con este conjunto de datos, utilizaremos el programa *R*, en el cual podremos realizar tanto la regresión, como un estudio descriptivo. Usaremos la librería *e*1071 [10], paquete software pensado para resolver problemas de clasificación y regresión mediante máquinas de vectores soporte, la cual podemos instalar fácilmente en *R*.

R implementa diversas formulaciones SVR con la posibilidad de usar tres importantes tipos de kernels y la posibilidad de usar tecnicas como la validación cruzada descrita en la sección 3.5.1, para la elección de los parámetros del kernel.

4.1. Funciones Específicas SVR

Comenzaremos describiendo las funciones que utilizaremos de la librería *e*1071 dedicadas a la resolución de problemas de regresión mediante máquinas de vectores soporte.

- El comando llamado `svm` es el utilizado por esta librería a la hora de obtener tanto la clasificación como la regresión mediante máquinas de vectores soporte.

```
## S3 method for class 'formula'
svm(formula, data = NULL, ..., subset, na.action =
na.omit, scale = TRUE)
## Default S3 method:
svm(x, y = NULL, scale = TRUE, type = NULL, kernel =
"radial", degree = 3, gamma = if (is.vector(x)) 1 else
1 / ncol(x), coef0 = 0, cost = 1, nu = 0.5,
```

```
class.weights = NULL, cachesize = 40, tolerance = 0.001,
epsilon = 0.1, shrinking = TRUE, cross = 0, probability = FALSE,
 fitted = TRUE, ..., subset, na.action = na.omit)
```

Se proporciona una interfaz de fórmula. Si la variable de predictor "y" incluye factores, la interfaz de fórmula debe ser utilizada para obtener una matriz de modelo correcto. El modelo de probabilidad para la clasificación se ajusta a una distribución logística utilizando máxima verosimilitud para los valores de decisión de todos los clasificadores binarios, y calcula las probabilidades de clase a posteriori para el problema de optimización cuadrática utilizando multiclase. El modelo de regresión probabilística asume errores de Laplace para los predictores, y estima el parámetro de escala utilizando máxima verosimilitud.

Un objeto de la clase `svm` contiene el modelo ajustado, incluyendo:

- ▷ `SV` → los vectores de soporte resultantes.
- ▷ `index` → índice de los vectores de soporte resultantes en la matriz de datos. Este índice se refiere a los datos preprocesados.
- ▷ `coefs` → los correspondientes coeficientes veces las etiquetas de formación.
- ▷ `rho` → el termino independiente.
- ▷ `sigma` → en caso de un modelo de regresión probabilístico, el parámetro de escala de la distribución de Laplace mediante estimación de máxima verosimilitud.
- ▷ `probA, probB` → vectores numéricos de longitud $\frac{k(k-1)}{2}$, siendo k, el número de las clases que contienen los parámetros de la distribución logística.

- El comando llamado `tune.svm` se utiliza para obtener una estimación del error cometido por los modelos según los parámetros específicos.

```
tune.svm (method, x, y = NULL, data = list(), validation.x =
     NULL, validation.y = NULL, ranges = NULL,
     predict.func = predict, tunecontrol = tune.control(), ...)
best.tune(...)
```

Se le proporciona el conjunto de datos junto al kernel con los posibles valores de los coeficientes específicos del modelo. Esta función calcula el error como medida de rendimiento, el error de clasificación se utiliza para la clasificación, y el error cuadrático medio para la regresión. Este error es calculado mediante validación cruzada.

La validación cruzada se realiza aleatoriamente a partir del conjunto de datos, una vez construidas estos pliegues, se mantienen constantes durante el proceso de formación.

Un objeto de la clase `tune.svm` contiene:

- ▷ `best.parameters` → tabla de datos $1 \times k$, k número de parámetros.
- ▷ `best.performance` → mejor rendimiento del modelo.
- ▷ `performances` → si se solicita, un conjunto de datos con todas las combinaciones de parámetros, junto con los resultados de rendimiento correspondientes.
- ▷ `train.ind` → lista de índices de vectores utilizados para la división en conjuntos de entrenamiento y validación.
- ▷ `best.model` → modelo completo asociado al mejor conjunto de parámetros.

Además de éstas, usaremos otras funciones que no son específicas de máquinas de vectores soporte, como `predict`, utilizada por R para predecir la variable respuesta a partir de un modelo. Esta función se utiliza tanto para la clasificación como para la regresión en nuestro caso. Para la representación de las gráficas de este trabajo utilizaremos un mapa de calor obtenido con la función `filled.contour`.

4.2. Ejemplo Práctico

Trabajaremos con el fichero de título *Boston Housing Data* obtenido de [8]. Estos estudios tienen su origen en la biblioteca StatLib mantenido en la Universidad Carnegie Mellon. Su fecha de creación data del 7 de julio de 1993 y los creadores son David Harrison y Daniel L Rubinfeld [3]. Son unos estudios realizados en Boston que recoge una serie de características de las zonas de Boston y el valor medio de la vivienda en cada zona. Para ello dividieron el territorio de Boston en 506 zonas.

Se recogían un total de catorce variables de cada zona:

1. CRIM Tasa de criminalidad.
2. ZN Proporción de zona residencial por cada 25000 pies cuadrados.
3. INDUS Proporción de hectáreas de comercio al pormenor.
4. CHAS (1) Si delimita con el río Charles, (0) si no.
5. NOX Concentración de óxido nitroso.
6. RM Promedio de habitaciones por vivienda.
7. AGE Proporción de viviendas ocupadas por sus propietarios construidas antes de 1940.
8. DIS Distancia ponderadas a 5 centros de empleo en Boston.
9. RAD Índice de accesibilidad a carreteras centrales.
10. TAX Valor total de la tasa de impuestos a la propiedad por 10000 $.

11. PTRATIO Relación alumno-profesor en la zona.
12. B $1000(Bk - 0{,}63)^2$ donde Bk es la proporción de personas de color.
13. LSTAT Estado inferior de la población.
14. MEDV Valor medio en miles de dolares, de las viviendas ocupadas.

Del conjunto de variables que disponemos trece son variables continuas entre ellas la variable a la que le queremos aplicar la regresión, la variable número 14, MEDV. Y una variable binaria que toma los valores 0 o 1. En este conjunto de datos no disponemos de valores perdidos.

4.2.1. Análisis Descriptivo

Comenzaremos con un análisis descriptivo de las variables para tener una idea de como son las variables con la cual vamos a trabajar.

```
datosHousing<- read.table(file="housing.data.txt",header=FALSE)
colnames(datosHousing)<- c("CRIM","ZN","IND","CHAS","NOX","RM",
                           "AGE","DIS","RAD","TAX","PTRA","B",
                           "LST", "MEDV")
```

```
summary(datosHousing)
```

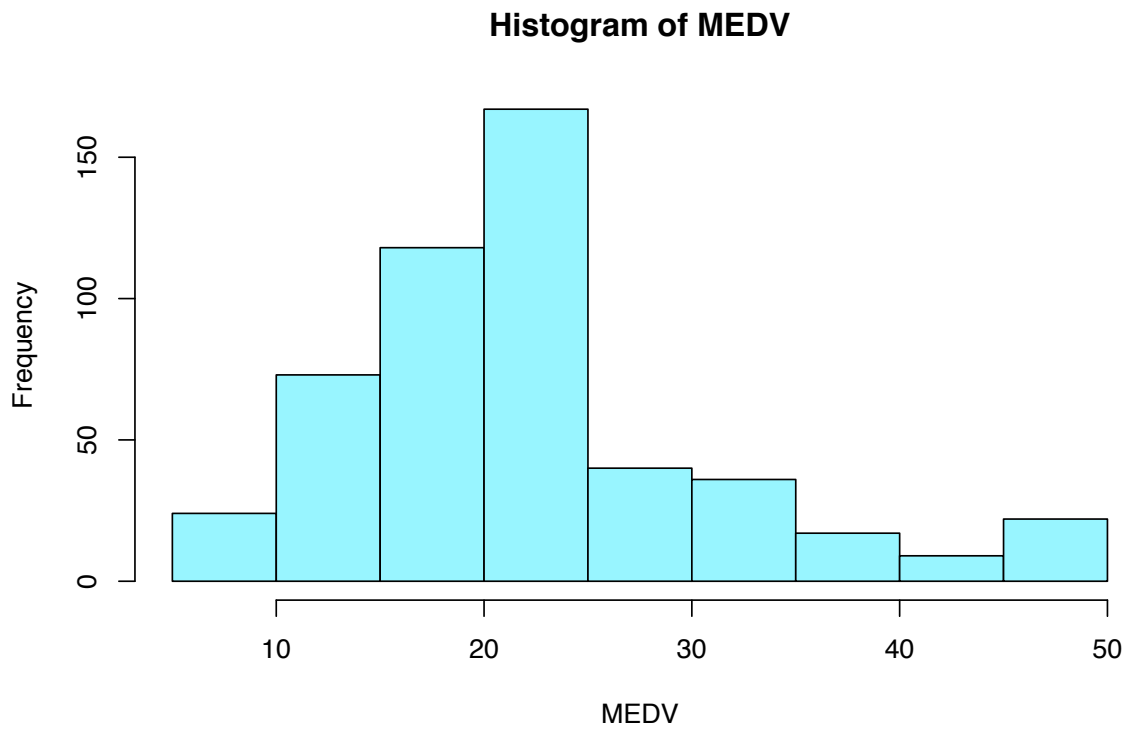
```
##       CRIM                ZN               IND              CHAS
##  Min.   : 0.00632   Min.   :  0.00   Min.   : 0.46   Min.   :0.000
##  1st Qu.: 0.08204   1st Qu.:  0.00   1st Qu.: 5.19   1st Qu.:0.000
##  Median : 0.25651   Median :  0.00   Median : 9.69   Median :0.000
##  Mean   : 3.61352   Mean   : 11.36   Mean   :11.14   Mean   :0.069
##  3rd Qu.: 3.67708   3rd Qu.: 12.50   3rd Qu.:18.10   3rd Qu.:0.000
##  Max.   :88.97620   Max.   :100.00   Max.   :27.74   Max.   :1.000
##       NOX               RM              AGE              DIS
##  Min.   :0.3850   Min.   :3.561   Min.   :  2.90   Min.   : 1.130
##  1st Qu.:0.4490   1st Qu.:5.886   1st Qu.: 45.02   1st Qu.: 2.100
##  Median :0.5380   Median :6.208   Median : 77.50   Median : 3.207
##  Mean   :0.5547   Mean   :6.285   Mean   : 68.57   Mean   : 3.795
##  3rd Qu.:0.6240   3rd Qu.:6.623   3rd Qu.: 94.08   3rd Qu.: 5.188
##  Max.   :0.8710   Max.   :8.780   Max.   :100.00   Max.   :12.127
##       RAD              TAX             PTRA             B
##  Min.   : 1.000   Min.   :187.0   Min.   :12.60   Min.   :  0.32
##  1st Qu.: 4.000   1st Qu.:279.0   1st Qu.:17.40   1st Qu.:375.38
##  Median : 5.000   Median :330.0   Median :19.05   Median :391.44
##  Mean   : 9.549   Mean   :408.2   Mean   :18.46   Mean   :356.67
##  3rd Qu.:24.000   3rd Qu.:666.0   3rd Qu.:20.20   3rd Qu.:396.23
##  Max.   :24.000   Max.   :711.0   Max.   :22.00   Max.   :396.90
```

```
##       LST              MEDV
##  Min.   : 1.73   Min.   : 5.00
##  1st Qu.: 6.95   1st Qu.:17.02
##  Median :11.36   Median :21.20
##  Mean   :12.65   Mean   :22.53
##  3rd Qu.:16.95   3rd Qu.:25.00
##  Max.   :37.97   Max.   :50.00
```

La matriz de correlaciones:

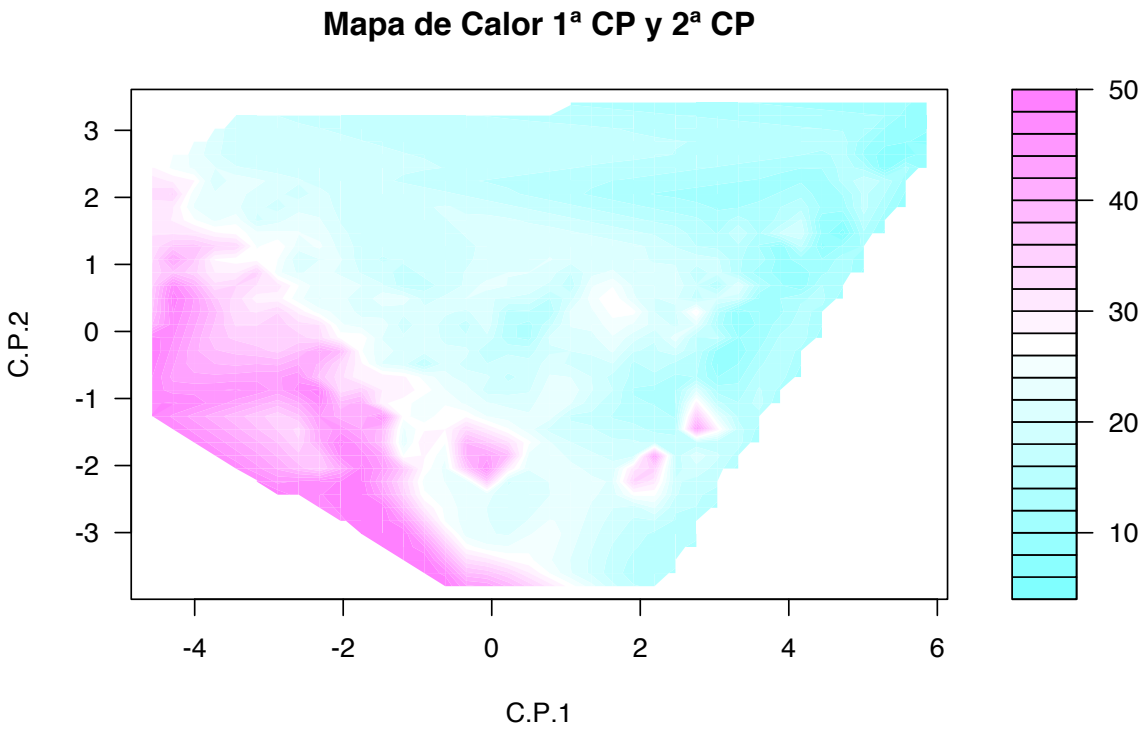
```
##       CRIM   ZN  IND CHAS  NOX   RM  AGE  DIS  RAD  TAX PTRA    B  LST MEDV
## CRIM   1.0 -0.2  0.4 -0.1  0.4 -0.2  0.4 -0.4  0.6  0.6  0.3 -0.4  0.5 -0.4
## ZN    -0.2  1.0 -0.5  0.0 -0.5  0.3 -0.6  0.7 -0.3 -0.3 -0.4  0.2 -0.4  0.4
## IND    0.4 -0.5  1.0  0.1  0.8 -0.4  0.6 -0.7  0.6  0.7  0.4 -0.4  0.6 -0.5
## CHAS  -0.1  0.0  0.1  1.0  0.1  0.1  0.1 -0.1  0.0  0.0 -0.1  0.0 -0.1  0.2
## NOX    0.4 -0.5  0.8  0.1  1.0 -0.3  0.7 -0.8  0.6  0.7  0.2 -0.4  0.6 -0.4
## RM    -0.2  0.3 -0.4  0.1 -0.3  1.0 -0.2  0.2 -0.2 -0.3 -0.4  0.1 -0.6  0.7
## AGE    0.4 -0.6  0.6  0.1  0.7 -0.2  1.0 -0.7  0.5  0.5  0.3 -0.3  0.6 -0.4
## DIS   -0.4  0.7 -0.7 -0.1 -0.8  0.2 -0.7  1.0 -0.5 -0.5 -0.2  0.3 -0.5  0.2
## RAD    0.6 -0.3  0.6  0.0  0.6 -0.2  0.5 -0.5  1.0  0.9  0.5 -0.4  0.5 -0.4
## TAX    0.6 -0.3  0.7  0.0  0.7 -0.3  0.5 -0.5  0.9  1.0  0.5 -0.4  0.5 -0.5
## PTRA   0.3 -0.4  0.4 -0.1  0.2 -0.4  0.3 -0.2  0.5  0.5  1.0 -0.2  0.4 -0.5
## B     -0.4  0.2 -0.4  0.0 -0.4  0.1 -0.3  0.3 -0.4 -0.4 -0.2  1.0 -0.4  0.3
## LST    0.5 -0.4  0.6 -0.1  0.6 -0.6  0.6 -0.5  0.5  0.5  0.4 -0.4  1.0 -0.7
## MEDV  -0.4  0.4 -0.5  0.2 -0.4  0.7 -0.4  0.2 -0.4 -0.5 -0.5  0.3 -0.7  1.0
```

Podemos representar un histograma de la variable MEDV:



Al disponer de trece variables independientes es difícil representar en una gráfica una descripción de nuestro conjunto. Para ello hacemos uso del análisis en componentes principales. Con este análisis podremos mostrar una representación de nuestra variable respuesta en función de las dos primeras componentes principales mediante un mapa de calor.

```
d <- interp(puntCP1,puntCP2,MEDV, duplicate="mean")
filled.contour(d, main = "Mapa de Calor 1ª CP y 2ª CP",xlab="C.P.1",
               ylab="C.P.2")
```



Datos Normalizados

Podemos apreciar en el resumen la diferencia de magnitudes de las variables. teniendo en cuenta las correlaciones de la variable respuesta con el resto, la diferencia de magnitud de unas variables a otras podría influenciar negativamente a nuestra regresión. Por eso aceptamos normalizar nuestras variables.

```
normalizar = function(datos){
  for (i in 1:ncol(datos)){
    datos[,i] <- (datos[,i] - mean(datos[,i]))/(sd(datos[,i]))
    i=i+1
  }
```

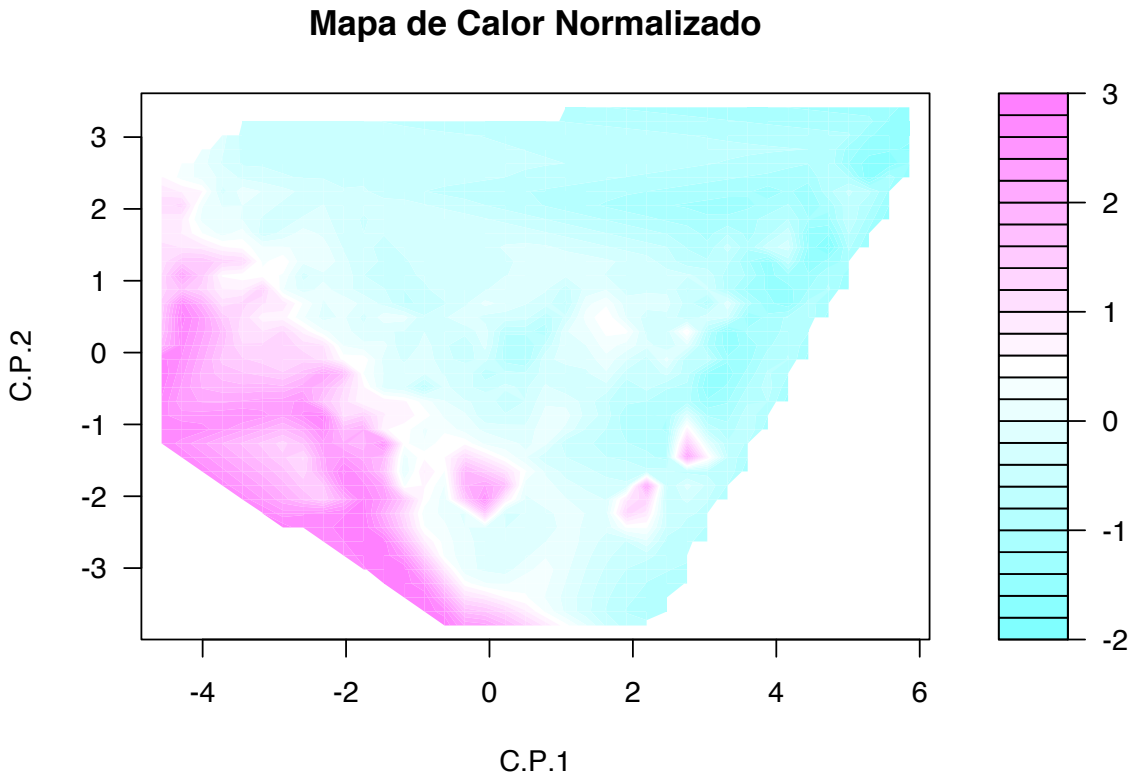
```
  return(datos)
}
```

Al aplicar esta función a nuestro conjunto de datos obtendríamos nuestros datos normalizados.

```
datosHousingNorm <- normalizar(datosHousing)
colnames(datosHousingNorm)<- c("CRIM_N","ZN_N","INDUS_N","CHAS_N",
                               "NOX_N","RM_N","AGE_N", "DIS_N",
                               "RAD_N","TAX_N", "PTRATIO_N","B_N",
                               "LSTAT_N", "MEDV_N")
```

Podemos representar el mapa de calor ahora de las variables normalizadas:

```
d <- interp(puntCP1Norm,puntCP2Norm,MEDV_N, duplicate="mean")
filled.contour(d, main = "Mapa de Calor Normalizado",
                    xlab="C.P.1", ylab="C.P.2")
```



Dado que los valores de las puntuaciones de las componentes principales son idénticos al caso anterior, el mapa de calor es el mismo salvo la diferencia que en este caso la variable respuesta esta normalizada.

4.3. Regresión mediante SVR

La librería utilizada en R $e1071$ nos pone a disposición cuatro diferentes kernels para abordar la regresión, que son:

- Linear : Kernel utilizado cuando pretendemos aproximar nuestra función con una función lineal.
$$K(x, x') = \langle x, x' \rangle$$
- Polynomial : Kernel que nos da la posibilidad de resolver nuestro problema con un kernel polinomial con diferentes grados d y coeficientes R.
$$K_d(x, x') = (\langle x, x' \rangle + R)^d$$
- Radial : Kernel que nos da la posibilidad de resolver nuestro problema con un kernel gaussiano con diferentes valores de γ.
$$K(x, x') = exp(-\gamma \|x - x'\|^2)$$
- Sigmoid : Kernel que nos da la posibilidad de resolver nuestro problema con un kernel sigmoidal con diferentes valores de γ y R .
$$K(x, x') = \tanh(\gamma \langle x, x' \rangle + R)$$

Realizaremos un estudio con cada tipo distinto de kernel y compararemos los resultados obtenidos., comparando la tasa de error cometida por cada uno de los kernels.

El programa R nos facilita la opción de estimar el error cometido por nuestro modelo, fijado la función kernel que decidamos usar y una rejilla con los posibles valores de los parámetros que puedan variar. Para ello usamos la función descrita anteriormente, *tune.svm* que mediante validación cruzada nos estima el error.

Para medir el error podemos utilizar la función de pérdida descrita en la parte teórica, podemos obtener el error medio cometido por este modelo con estos datos con esta función de pérdida. Esta función de pérdida la podemos describir en R como:

```
## Función de pérdida 1

L1 = function(ypred,y,epsilon){
       Err=0
       for (i in 1:(length(y))){
         if (abs(ypred[i]-y[i]) > epsilon){
           Err = Err + abs(ypred[i]-y[i])
         }
         i=i+1
       }
      return(Err)
    }
```

```
## Cálculo Error de un Modelo

## ErrorTotal

errorModeloL1 = function(modelo, conjtest, varResp){
  ypred = predict(modelo,conjtest)
  Err = L1(ypred,varResp,modelo$epsilon)
  return(Err)
}
```

```
## Error Medio

errorModeloL1Medio = function(modelo, conjtest, varResp){
  ypred = predict(modelo,conjtest)
  Err = L1(ypred,varResp,modelo$epsilon)/nrow(conjtest)
  return(Err)
}
```

4.3.1. Kernel Lineal

Comenzaremos realizando el estudio con el uso del kernel lineal:

```
tuneado0_Norm <- tune.svm(MEDV_N~.,data=datosHousingNorm ,
                  kernel="linear",cost = 10^(-1:2),
                  epsilon = 10^(-2:0),scale=TRUE)
summary(tuneado0_Norm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  cost epsilon
##    10     0.1
##
## - best performance: 0.3048005
##
## - Detailed performance results:
##     cost epsilon     error dispersion
## 1    0.1    0.01 0.3087752 0.10002141
## 2    1.0    0.01 0.3068002 0.10063714
## 3   10.0    0.01 0.3063899 0.10007151
```

```
## 4  100.0    0.01 0.3063138 0.09993953
## 5    0.1    0.10 0.3064131 0.10226658
## 6    1.0    0.10 0.3048102 0.10144150
## 7   10.0    0.10 0.3048005 0.10118475
## 8  100.0    0.10 0.3048677 0.10118566
## 9    0.1    1.00 0.3846444 0.08868792
## 10   1.0    1.00 0.3743516 0.08735723
## 11  10.0    1.00 0.3773173 0.08648641
## 12 100.0    1.00 0.3768366 0.08549246
```

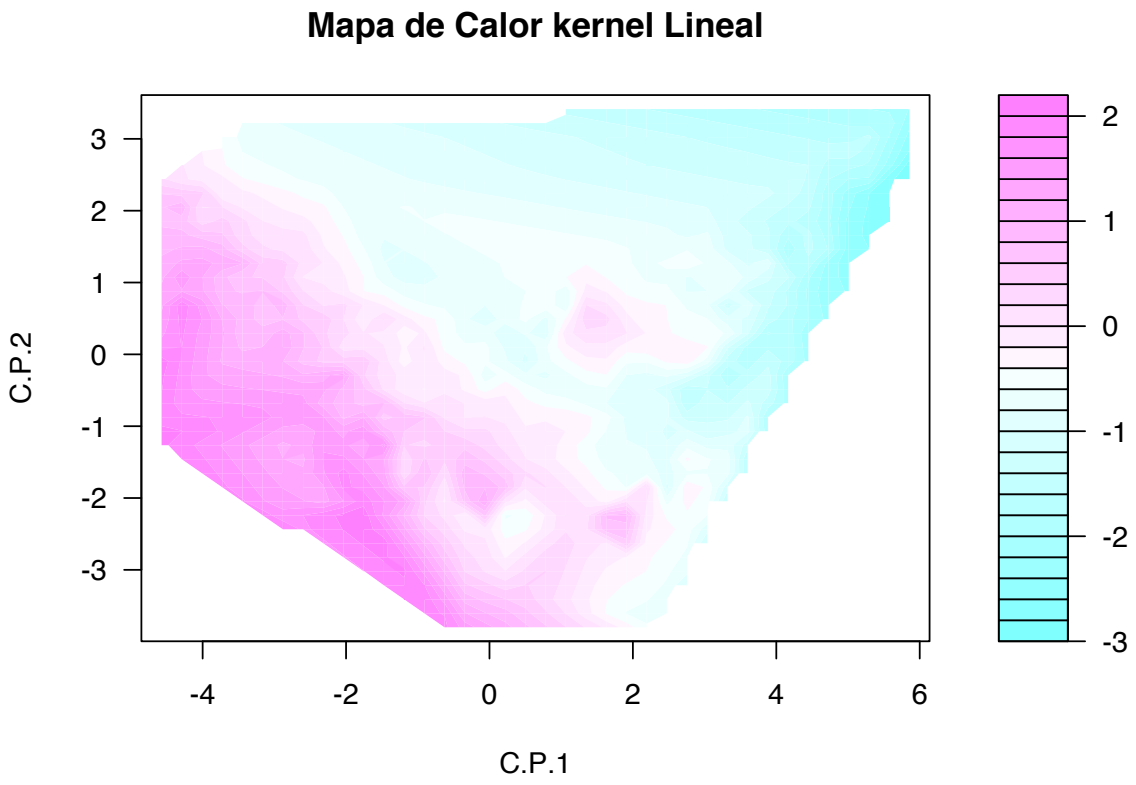
A partir de esta estimación podemos obtener el modelo que minimiza el error. En nuestro caso este modelo es:

```
mejorajuste0_Norm=tuneado0_Norm$best.model
print(mejorajuste0_Norm)
```

```
##
## Call:
## best.svm(x = MEDV_N ~ ., data = datosHousingNorm, cost = 10^(-1:2),
##     epsilon = 10^(-2:0), kernel = "linear", scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  linear
##        cost:  10
##       gamma:  0.07692308
##     epsilon:  0.1
##
##
## Number of Support Vectors:  380
```

A partir de esta función de regresión producida por este modelo nosotros podemos obtener un mapa de calor con los resultados de nuestra función aproximada para una representación visual de nuestra regresión.

```
d0 <- interp(puntCP1Norm,puntCP2Norm,ypredNorm0,duplicate="mean")
filled.contour(d0, main = "Mapa de Calor Kernel Lineal",
                    xlab="C.P.1",  ylab="C.P.2")
```




```
errorModeloL1Medio(mejorajuste0_Norm,datosHousingNorm,MEDV_N)
```

```
## 0.3235691
```

Otra medida habitual de error es el cálculo del error cuadrático medio. En este caso:

```
## 0.2869733
```

4.3.2. Kernel Polinomial

Nos planteamos un estudio similar al realizado con el kernel lineal, cambiando el kernel. En este caso el kernel polinomial dispone de dos parámetros propios junto al parámetro coste y epsilon.

```
tuneado1_Norm <- tune.svm(MEDV_N~.,data=datosHousingNorm ,
                    kernel="polynomial", cost = 10^(-1:2),
                    epsilon = 10^(-1:0), degree=(3:5),scale=TRUE)
summary(tuneado1_Norm)
```

```
##
## Parameter tuning of 'svm':
```

```
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  degree cost epsilon
##       3    1     0.1
##
## - best performance: 0.2627534
##
## - Detailed performance results:
##    degree  cost epsilon     error dispersion
## 1       3   0.1     0.1 0.3953308  0.1817405
## 2       4   0.1     0.1 0.4495784  0.2071792
## 3       5   0.1     0.1 0.6334360  0.6379059
## 4       3   1.0     0.1 0.2627534  0.1754837
## 5       4   1.0     0.1 0.4032249  0.4316404
## 6       5   1.0     0.1 2.3424848  6.4478723
## 7       3  10.0     0.1 0.5558749  1.0074161
## 8       4  10.0     0.1 2.3552565  6.3333634
## 9       5  10.0     0.1 3.9880200 10.2653011
## 10      3 100.0     0.1 1.7685265  3.9239705
## 11      4 100.0     0.1 6.1934556 16.3795169
## 12      5 100.0     0.1 6.2235547 12.2730292
## 13      3   0.1     1.0 0.5642490  0.1301769
## 14      4   0.1     1.0 0.6259962  0.1446387
## 15      5   0.1     1.0 0.7316682  0.4238583
## 16      3   1.0     1.0 0.4450963  0.1004615
## 17      4   1.0     1.0 1.3929350  2.7675810
## 18      5   1.0     1.0 1.4406510  2.9467190
## 19      3  10.0     1.0 0.3915978  0.1273010
## 20      4  10.0     1.0 3.1440790  8.5417739
## 21      5  10.0     1.0 1.9549975  4.4892543
## 22      3 100.0     1.0 0.3744535  0.1233382
## 23      4 100.0     1.0 2.7446738  7.2553780
## 24      5 100.0     1.0 1.9533168  4.6003029
```

En este caso el mejor modelo seria:

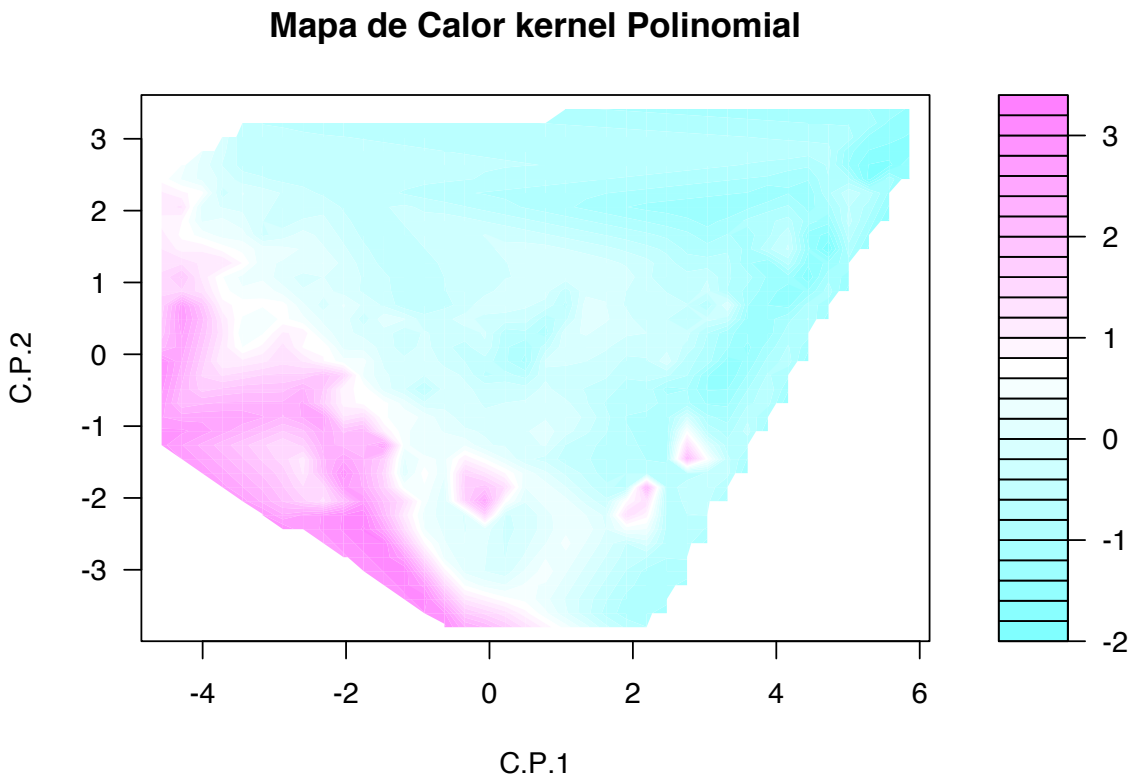
```
mejorajuste1_Norm=tuneado1_Norm$best.model
print(mejorajuste1_Norm)
```

```
##
## Call:
## best.svm(x = MEDV_N ~ ., data = datosHousingNorm, degree = (3:5),
##     cost = 10^(-1:2), epsilon = 10^(-1:0), kernel = "polynomial",
##     scale = TRUE)
```

```
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##       gamma:  0.07692308
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  372
```

Generando así un mapa de calor y un error medio:

```
d1 <- interp(puntCP1Norm,puntCP2Norm,ypredNorm1,duplicate="mean")
filled.contour(d1, main = "Mapa de Calor Kernel Polinomial",
                     xlab="C.P.1", ylab="C.P.2")
```



```
errorModeloL1Medio(mejorajuste1_Norm,datosHousingNorm,MEDV_N)
```

```
## 0.1988483
```

Se tiene también que el error cuadrático medio cometido con esta función de predicción es:

```
## 0.1098226
```

4.3.3. Kernel Radial

Ahora realizamos el estudio con el kernel radial. Tenemos en este caso un nuevo parámetro γ junto al coste y epsilon.

```
tuneado2_Norm <- tune.svm(MEDV_N~.,data=datosHousingNorm,
                  kernel= "radial", gamma = 10^(-4:-1),
                  cost = 10^(-1:1), epsilon = 10^(-1:0), scale=TRUE)
summary(tuneado2_Norm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma cost epsilon
##    0.1   10     0.1
##
## - best performance: 0.1259211
##
## - Detailed performance results:
##    gamma cost epsilon     error dispersion
## 1  1e-04  0.1     0.1 0.9798190 0.31440489
## 2  1e-03  0.1     0.1 0.7591207 0.30945188
## 3  1e-02  0.1     0.1 0.4154568 0.23749867
## 4  1e-01  0.1     0.1 0.3686023 0.24588329
## 5  1e-04  1.0     0.1 0.7537387 0.30924793
## 6  1e-03  1.0     0.1 0.3964701 0.22075054
## 7  1e-02  1.0     0.1 0.2219599 0.15499423
## 8  1e-01  1.0     0.1 0.1768023 0.13373057
## 9  1e-04 10.0     0.1 0.3967259 0.22037508
## 10 1e-03 10.0     0.1 0.2799216 0.17562604
## 11 1e-02 10.0     0.1 0.1692663 0.10806448
## 12 1e-01 10.0     0.1 0.1259211 0.07722572
## 13 1e-04  0.1     1.0 0.9798601 0.26939380
```

```
## 14 1e-03  0.1     1.0 0.8495537 0.25519538
## 15 1e-02  0.1     1.0 0.6377322 0.20342812
## 16 1e-01  0.1     1.0 0.6232611 0.22389760
## 17 1e-04  1.0     1.0 0.8451833 0.25397370
## 18 1e-03  1.0     1.0 0.6000316 0.17935163
## 19 1e-02  1.0     1.0 0.4555450 0.12728025
## 20 1e-01  1.0     1.0 0.4036266 0.11584329
## 21 1e-04 10.0     1.0 0.5969894 0.17833236
## 22 1e-03 10.0     1.0 0.4684074 0.12180408
## 23 1e-02 10.0     1.0 0.3233276 0.06902243
## 24 1e-01 10.0     1.0 0.3581760 0.07587053
```

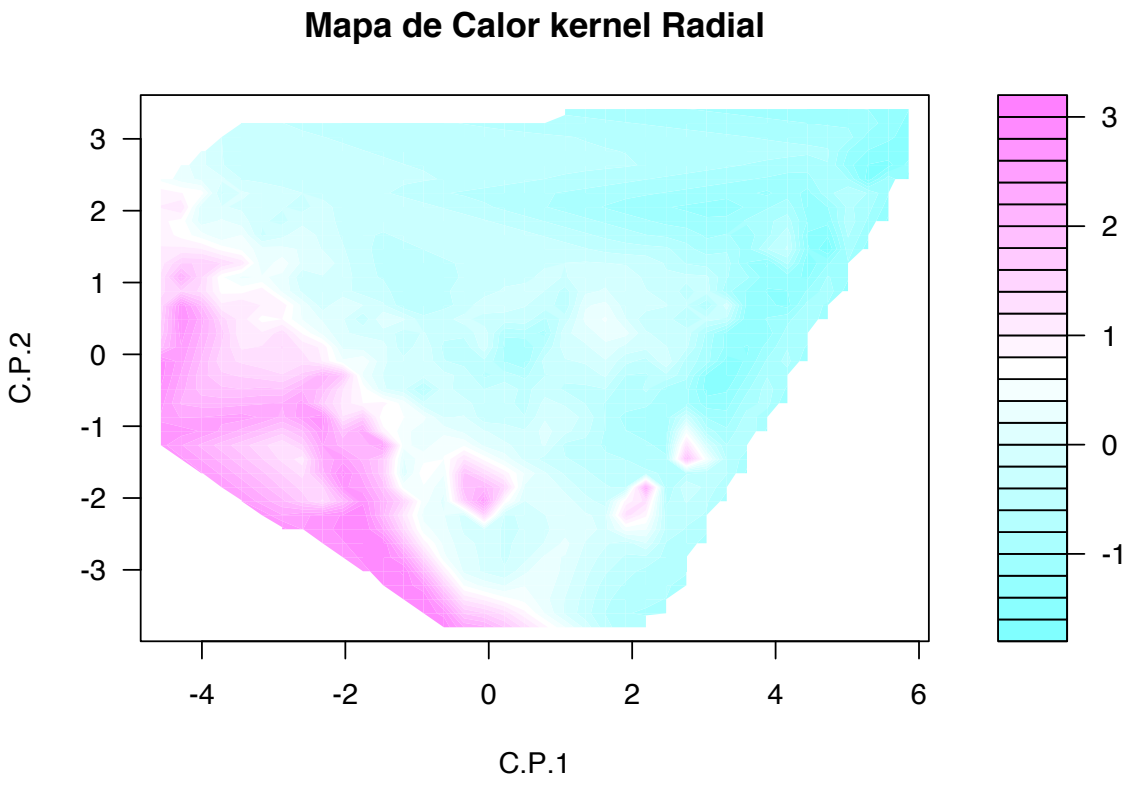
En este caso el mejor modelo seria:

```
mejorajuste2_Norm=tuneado2_Norm$best.model
print(mejorajuste2_Norm)
```

```
##
## Call:
## best.svm(x = MEDV_N ~ ., data = datosHousingNorm, gamma = 10^(-4:-1),
##     cost = 10^(-1:1), epsilon = 10^(-1:0), kernel = "radial",
##     scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  10
##       gamma:  0.1
##     epsilon:  0.1
##
##
## Number of Support Vectors:  340
```

Generando asi un mapa de calor y un error medio:

```
d2 <- interp(puntCP1Norm,puntCP2Norm,ypredNorm2,duplicate="mean")
filled.contour(d2, main = "Mapa de Calor Kernel Radial",
                    xlab="C.P.1", ylab="C.P.2")
```




```
errorModeloL1Medio(mejorajuste2_Norm,datosHousingNorm,MEDV_N)
```

```
## 0.08626746
```

Con error cuadrático medio:

```
## 0.03465188
```

Apreciamos que ambos errores de aproximación son cercanos a 0. También se observa que el mapa de calor aproximado es similar al de los datos de entrada.

4.3.4. Kernel Sigmoidal

Por último realizamos el estudio con el kernel Sigmoidal. Tenemos el parámetro γ y R junto al coste y epsilon.

```
tuneado3_Norm <- tune.svm(MEDV_N~.,data=datosHousingNorm,
                      kernel= "sigmoid", coef0 = c(0,5),
                      gamma = 10^(-3:0), cost = 10^(-1:1),scale=TRUE)
summary(tuneado3_Norm)
```

```
##
## Parameter tuning of 'svm':
##
## - sampling method: 10-fold cross validation
##
## - best parameters:
##  gamma coef0 cost
##  0.001     0   10
##
## - best performance: 0.3112602
##
## - Detailed performance results:
##    gamma coef0 cost        error   dispersion
## 1  0.001     0  0.1 8.521782e-01 2.324651e-01
## 2  0.010     0  0.1 4.936465e-01 1.539728e-01
## 3  0.100     0  0.1 7.704087e-01 2.336415e-01
## 4  1.000     0  0.1 1.923297e+01 2.951157e+00
## 5  0.001     5  0.1 1.021740e+00 2.555519e-01
## 6  0.010     5  0.1 1.021382e+00 2.554847e-01
## 7  0.100     5  0.1 1.013524e+00 2.538353e-01
## 8  1.000     5  0.1 3.557984e+01 2.988301e+00
## 9  0.001     0  1.0 4.931048e-01 1.539584e-01
## 10 0.010     0  1.0 3.126837e-01 1.106426e-01
## 11 0.100     0  1.0 5.270987e+01 1.417341e+01
## 12 1.000     0  1.0 1.882462e+03 3.321538e+02
## 13 0.001     5  1.0 1.021378e+00 2.554857e-01
## 14 0.010     5  1.0 1.017759e+00 2.547409e-01
## 15 0.100     5  1.0 9.471725e-01 2.421115e-01
## 16 1.000     5  1.0 3.583226e+03 2.627001e+02
## 17 0.001     0 10.0 3.112602e-01 1.096166e-01
## 18 0.010     0 10.0 3.270285e-01 1.165002e-01
## 19 0.100     0 10.0 5.234178e+03 1.425616e+03
## 20 1.000     0 10.0 1.884112e+05 3.431191e+04
## 21 0.001     5 10.0 1.017718e+00 2.547481e-01
## 22 0.010     5 10.0 9.841693e-01 2.501437e-01
## 23 0.100     5 10.0 7.419826e-01 2.053973e-01
## 24 1.000     5 10.0 3.566550e+05 2.853332e+04
```

En este caso el mejor modelo seria:

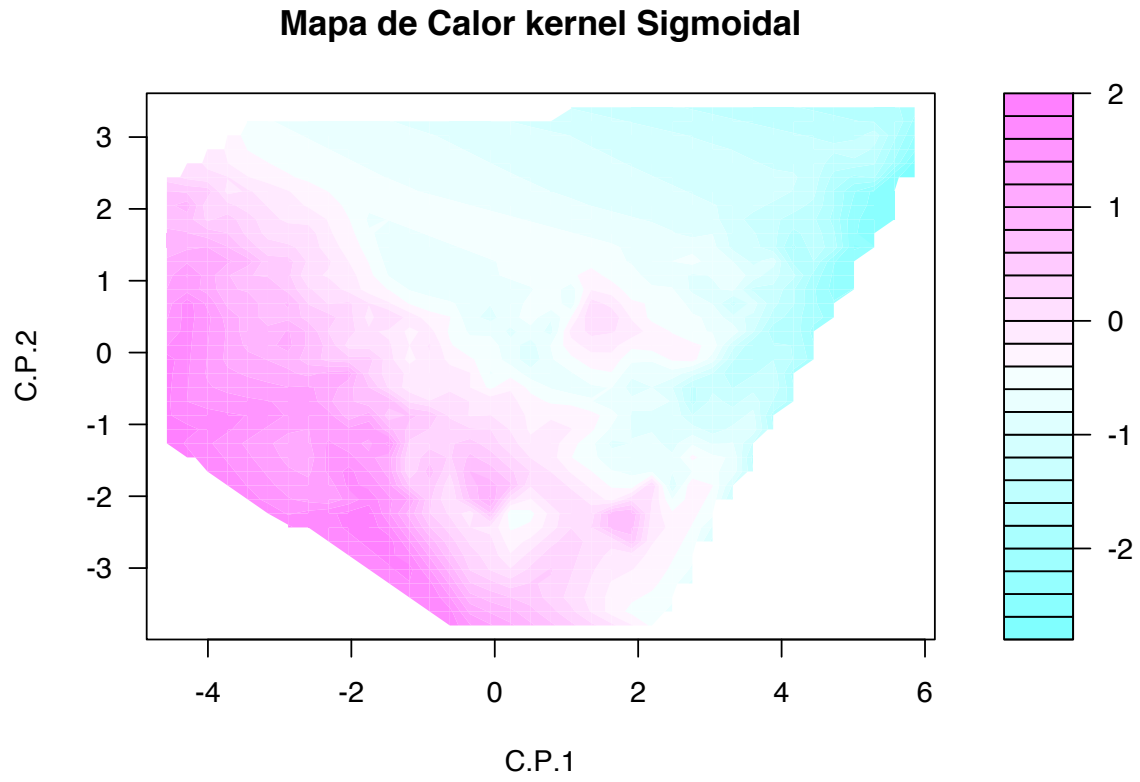
```
mejorajuste3_Norm=tuneado3_Norm$best.model
print(mejorajuste3_Norm)
```

```
##
## Call:
## best.svm(x = MEDV_N ~ ., data = datosHousingNorm, gamma = 10^(-3:0),
```

```
##     coef0 = c(0, 5), cost = 10^(-1:1), kernel = "sigmoid",
##             scale = TRUE)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  sigmoid
##        cost:  10
##       gamma:  0.001
##      coef.0:  0
##     epsilon:  0.1
##
##
## Number of Support Vectors:  390
```

Generando asi un mapa de calor y un error medio:

```
d3 <- interp(puntCP1Norm,puntCP2Norm,ypredNorm3,duplicate="mean")
filled.contour(d3, main = "Mapa de Calor Kernel Sigmoidal",
                    xlab="C.P.1", ylab="C.P.2")
```



```
errorModeloL1Medio(mejorajuste3_Norm,datosHousingNorm,MEDV_N)
```

```
## 0.3298638
```

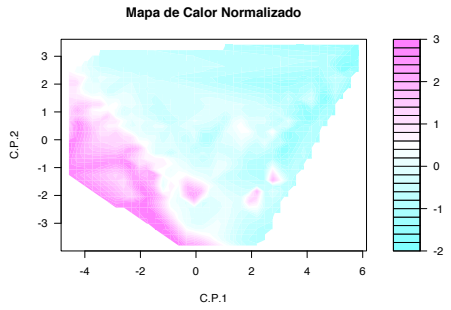
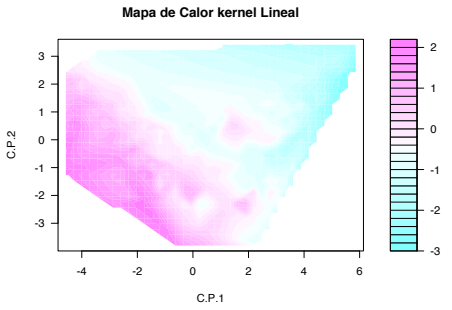
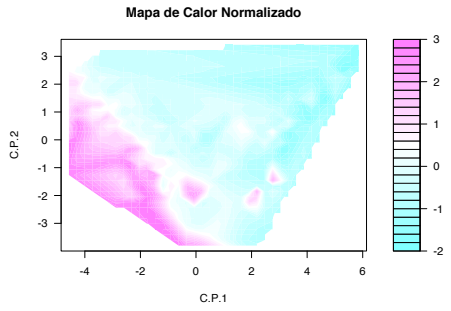
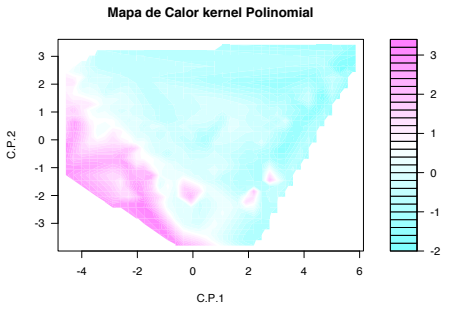
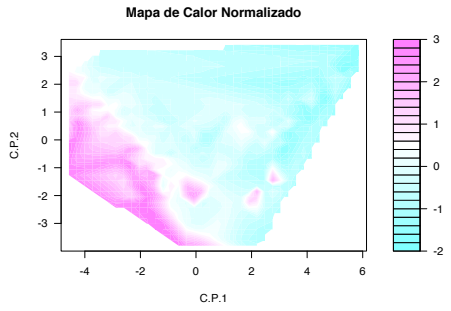
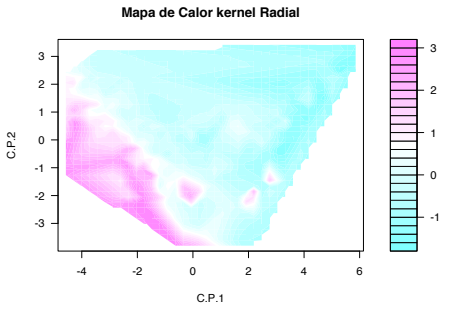
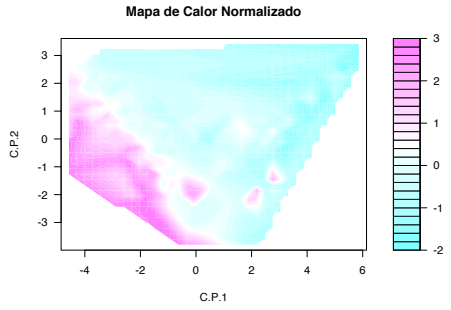
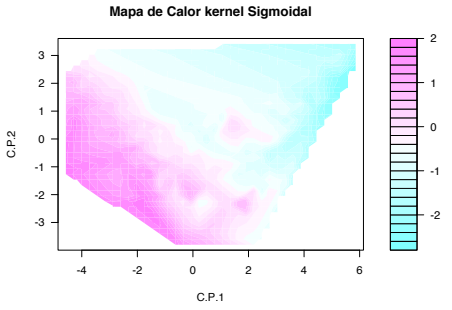
Y error cuadrático medio asociado:

```
## 0.3033291
```

Finalizaremos el estudio con un resumen con la comparación de los modelos utilizados.

4.4. Comparación de Modelos

Tras realizar el mismo estudio con cada uno de los kernels predefinidos en la librería *e*1071 podemos comparar su eficacia frente a nuestro conjunto de datos.

Original	Aproximación
	 E.L1 = 0.3235691, ECM=0.2869733
	 E.L1 = 0.1988483, ECM=0.1098226
	 E.L1 = 0.0862674, ECM=0.0346518
	 E.L1 = 0.3298638, ECM=0.3033291

Apreciamos que ambos errores alcanzan su mínimo para el caso del kernel Radial. Por lo que cabe esperar que nuestra función de aproximación sera mejor si utilizamos el kernel Radial con los mejores parámetros estimados mediante validación cruzada.

Bibliografía

[1] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3), 1995.

[2] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[3] David Harrison and Daniel L Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of environmental economics and management*, 5(1):81–102, 1978.

[4] Trevor J. Hastie, Robert John Tibshirani, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer, 2013.

[5] Peter J Huber. Robust statistics. 1981.

[6] Vsevolod I. Ivanov. Duality in nonlinear programming. *Optimization Letters*, 7(8):1643–1658, 2013.

[7] Bart Kosko. Neural networks and fuzzy systems: a dynamical approach to machine intelligence. *Englewood Cliffs. NJ: Prentice—Hall*, 1(99):1, 1992.

[8] M. Lichman. UCI machine learning repository, 2013.

[9] David G Luenberger and Manuel López Mateos. *Programación lineal y no lineal*. Number 90C05 LUEp. Addison-Wesley Iberoamericana, 1989.

[10] David Meyer, Evgenia Dimitriadou, Kurt Hornik, Andreas Weingessel, and Friedrich Leisch. *e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071), TU Wien*, 2015. R package version 1.6-7.

[11] Alberto Muñoz, Isaac Martín de Diego, and Javier M Moguerza. Support vector machine classifiers for asymmetric proximities. In *Artificial Neural Networks and Neural Information Processing—ICANN/ICONIP 2003*, pages 217–224. Springer, 2003.

[12] Alberto Muñoz. Compound key word generation from document databases using a hierarchical clustering art model. *Intelligent Data Analysis*, 1(1):25–48, 1997.

[13] Bernhard Schölkopf, Sebastian Mika, Chris JC Burges, Philipp Knirsch, Klaus-Robert Müller, Gunnar Rätsch, and Alexander J Smola. Input space versus feature space in kernel-based methods. *Neural Networks, IEEE Transactions on*, 10(5):1000–1017, 1999.

[14] John Shawe-Taylor and Nello Cristianini. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

[15] Vladimir Vapnik, Steven E Golowich, and Alex Smola. Support vector method for function approximation, regression estimation, and signal processing. In *Advances in neural information processing systems 9*. Citeseer, 1996.