Universidad del País Vasco
Euskal Herriko Unibertsitatea

ZIENTZIA ETA TEKNOLOGIA FAKULTATEA
FACULTAD DE CIENCIA Y TECNOLOGÍA

Trabajo de Fin de Grado
Grado en Ingeniería Electrónica

Estudio, desarrollo y evaluación de técnicas de aprendizaje automático en tareas de clasificación y/o predicción

Detección de exoplanetas

Autor:
Adrián Meléndez Lorenzo
Director:
Luis Javier Rodríguez Fuentes

© 2022, Adrián Meléndez Lorenzo

Leioa, 20 de junio de 2022

Índice

1. Introducción

1.1. La explosión de los datos

En las últimas décadas se ha vivido una de las mayores revoluciones tecnológicas conocidas hasta la fecha. El rápido desarrollo de los dispositivos electrónicos y las comunicaciones ha dado lugar a una producción creciente de datos de todo tipo, cuyo tratamiento y almacenamiento es fundamental para que la sociedad funcione como la conocemos hoy en día. Los registros sanitarios, las cuentas bancarias o las búsquedas de internet son algunos de los ejemplos en los que se observa la necesidad y la importancia del tratamiento de datos.

Este aumento casi exponencial en la producción de datos ha dado lugar a una nueva rama de estudio, la ciencia de datos, una parte de la cual se dedica al estudio y desarrollo de métodos automáticos para clasificar y estudiar grandes cantidades de datos de manera rápida y eficaz. Algunos de estos métodos automáticos utilizan técnicas de aprendizaje automático o "Machine Learning", abreviado muchas veces como ML.

Hoy en día el ML está presente en diversas áreas de conocimiento. Los algoritmos de aprendizaje automático son utilizados tanto en el desarrollo de la conducción autónoma de un automóvil como en la predicción de las subidas y bajadas de las acciones en bolsa. Las técnicas de ML son una de las herramientas más usadas actualmente y, si se dispone de una base de datos lo bastante grande y representativa, se pueden obtener resultados realmente sorprendentes.

1.2. Detección de exoplanetas

En este trabajo se aplicarán técnicas de aprendizaje automático para la detección de exoplanetas. Los exoplanetas son aquellos que se encuentran fuera del sistema solar, orbitando alrededor de estrellas diferentes del Sol. La búsqueda de este tipo de objetos es interesante en el campo de la astrofísica, ya que si se encuentran en la zona habitable de su estrella podrían albergar vida extraterrestre.

Para llevar a cabo este proyecto se analizarán una serie de datos recogidos por el telescopio espacial de la NASA Kepler [1]. Estos datos reúnen la intensidad de la luz medida por el telescopio procedente de diferentes estrellas en diferentes instantes de tiempo. A partir de estos datos se puede determinar si existe algún objeto de tamaño considerable orbitando alrededor de la estrella. Esto es posible porque al interponerse el planeta entre la estrella y el telescopio, la intensidad de la luz medida disminuirá de manera apreciable, tal y como muestra la Figura 1.1. Este es un método clásico de detección de exoplanetas sobre el que aplicaremos algoritmos de ML para estudiar una gran cantidad de datos de manera eficiente.

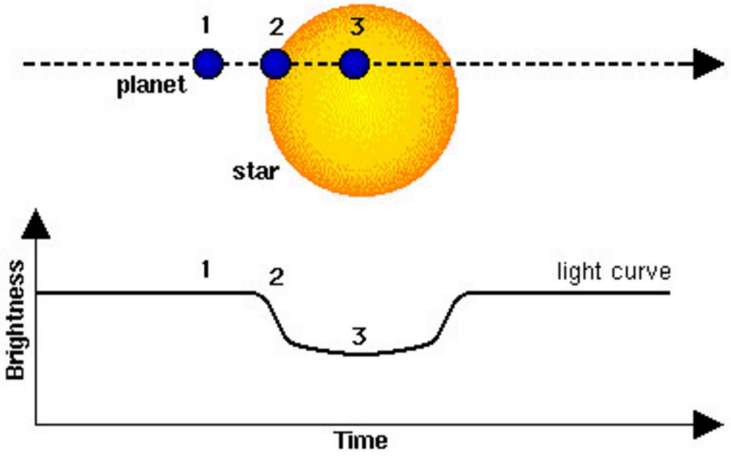


Figura 1.1: Ejemplo de la detección de exoplanetas mediante el análisis de la intensidad de la luz recibida [2].

1.3. Objetivos del trabajo

Durante el desarrollo del trabajo se explicarán diferentes métodos de aprendizaje automático así como sus principales ventajas y desventajas. Se compararán las diferentes alternativas y se comentará la razón por la que han sido elegidas.

Una vez estén claros los diferentes métodos a utilizar, se procederá con la parte práctica y se aplicarán los algoritmos necesarios a los datos del telescopio Kepler. Para ello, se utilizará el lenguaje de programación `Python` y sus módulos `scikit-learn` para los algoritmos de ML, `pandas` para el tratamiento de datos y `matplotlib` para las representaciones gráficas.

Además, se llevará a cabo un tratamiento previo de los datos con el objetivo de comparar los resultados con y sin dicho preproceso. Así, se verá la importancia de disponer de una buena base de datos antes incluso de aplicar ningún algoritmo de clasificación. El tratamiento previo implicará equilibrar, normalizar y aplicar filtros a los datos iniciales antes de aplicar sobre ellos los métodos de ML.

2. Fundamento teórico de los algoritmos a utilizar

En ML se distinguen dos tipos de tareas principales: clasificación y regresión. La primera implica clasificar los datos en diferentes categorías; en nuestro caso, la existencia o no de exoplanetas alrededor de una estrella. La segunda tarea requiere la obtención de una función que devuelva un valor numérico tomando los valores de las características como entrada; por ejemplo, calcular la renta media en una población. En este trabajo se utilizarán algoritmos de clasificación. En este capítulo se explica la base teórica de los métodos aplicados. Las tareas de clasificación pueden requerir dos tipos de aprendizaje: supervisado y no supervisado. El aprendizaje supervisado es aquel en el que el conjunto de datos de entrenamiento dispone de etiquetas que indican a qué clase pertenece cada punto. Por su parte, en el aprendizaje no supervisado no existen estas etiquetas. Además, se explicarán diversas técnicas para preparar el conjunto de datos antes del entrenamiento en caso de que dicho conjunto no esté debidamente balanceado. [3].

2.1. Algunas técnicas de aprendizaje supervisado

2.1.1. Regresión logística

La regresión logística, al igual que la regresión lineal, trata de estimar un valor numérico o una serie de valores numéricos a partir de unos valores de entrada. Sin embargo, los algoritmos de regresión pueden utilizarse también como algoritmos de clasificación. La regresión logística permite calcular la probabilidad de que una instancia pertenezca a una clase de entre dos posibles. Por tanto, si dicha probabilidad es mayor del 50 %, se podrá decir que la instancia pertenece a esa clase y en caso contrario, que pertenece a la otra. Esto lo convierte en un clasificador binario. Dichas probabilidades se calculan mediante la ecuación:

$$\hat{p} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \sigma(\mathbf{x}^\intercal \boldsymbol{\theta}) \tag{2.1}$$

En esta ecuación:

- $\boldsymbol{\theta}$ es el vector de parámetros del modelo, el que se obtiene durante el entrenamiento.
- $\mathbf{x}$ es el vector de características de la instancia, de forma que $x = [x_0, x_1, \ldots, x_n]$, donde siempre $x_0 = 1$.
- $\mathbf{x}^\intercal \boldsymbol{\theta} = \theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n$.
- $h_{\boldsymbol{\theta}}$ se denomina función hipótesis.
- La función logística $\sigma(\cdot)$ es una función sigmoide (con forma de S) definida como:

$$\sigma(t) = \frac{1}{1 + \exp(-t)} \tag{2.2}$$

A partir de aquí, un clasificador por regresión logística puede calcular la probabilidad de que una instancia pertenezca a una clase y hacer su predicción como se ha comentado anteriormente:

$$y = \begin{cases} 0 & \text{si } \hat{p} < 0.5 \\ 1 & \text{si } \hat{p} \geq 0.5 \end{cases} \tag{2.3}$$

Si observamos la forma de la sigmoide en la Figura 2.1, podemos ver que las condiciones anteriores se resumen en

$$y = \begin{cases} 0 & \text{si } \mathbf{x}^\intercal \boldsymbol{\theta} < 0 \\ 1 & \text{si } \mathbf{x}^\intercal \boldsymbol{\theta} \geq 0 \end{cases} \tag{2.4}$$

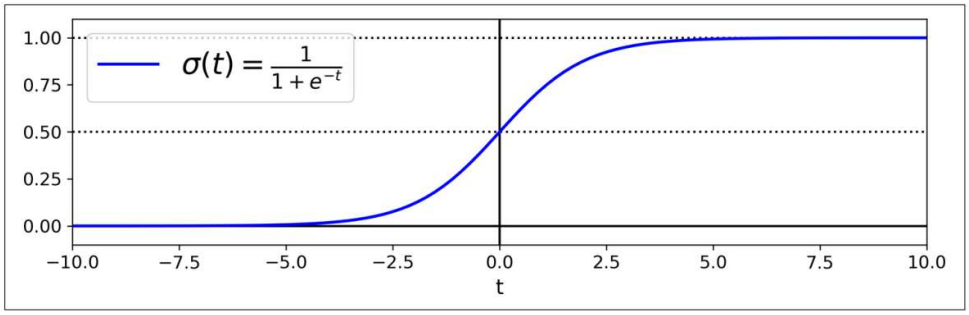


Figura 2.1: Comportamiento de la función $\sigma(t)$ [3].

Entrenar este modelo implica encontrar los parámetros del vector $\boldsymbol{\theta}$ de forma que se obtengan probabilidades altas para instancias que pertenecen a la clase $y = 1$ y probabilidades bajas para instancias que pertenecen a la clase $y = 0$. Para un solo punto de entrenamiento $\mathbf{x}$, podemos definir una función, denominada función de pérdida, de la forma:

$$c(\boldsymbol{\theta}) = \begin{cases} -\log(\hat{p}) & \text{si } y = 1 \\ -\log(1 - \hat{p}) & \text{si } y = 0 \end{cases} \tag{2.5}$$

Esta función de pérdida tiene las siguientes características:

- Para $y = 1$ se hace muy grande cuando la probabilidad tiende a 0 y se acerca a 0 cuando la probabilidad tiende a 1.
- Para $y = 0$ se hace muy grande cuando la probabilidad tiende a 1 y se acerca a 0 cuando la probabilidad tiende a 0.

Es decir, devolverá una pérdida muy grande en los casos erróneos y una muy baja cuando se cumple lo que se quiere obtener. Por tanto, para estimar el vector $\boldsymbol{\theta}$ habrá que minimizar esta función de pérdida. Se puede definir una función de pérdida para un conjunto de entrenamiento completo como la pérdida media sobre todas las instancias. Se denomina pérdida logística y viene dada por:

$$J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log\left(\hat{p}^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - \hat{p}^{(i)}\right) \right] \tag{2.6}$$

Sin embargo, no se conoce ninguna fórmula analítica cerrada que permita calcular el vector $\boldsymbol{\theta}$ que minimiza esta función de pérdida. La buena noticia es que esta función de

pérdida es convexa; es decir, se puede encontrar el mínimo global mediante la técnica de descenso por gradiente. Para ello serán necesarias las derivadas parciales de la función de pérdida, que vienen dadas por:

$$\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} \left(\sigma\left(\boldsymbol{\theta}^{\intercal} \mathbf{x}^{(i)}\right) - y^{(i)} \right) x_j^{(i)} \tag{2.7}$$

Descenso por gradiente

La idea general del descenso por gradiente es ajustar los parámetros del modelo de forma iterativa para minimizar la función de pérdida. En concreto, empieza con valores aleatorios en $\boldsymbol{\theta}$ y después los va modificando de manera gradual, paso a paso, intentando reducir la función de pérdida en cada iteración hasta que converge en un mínimo (Figura 2.2).

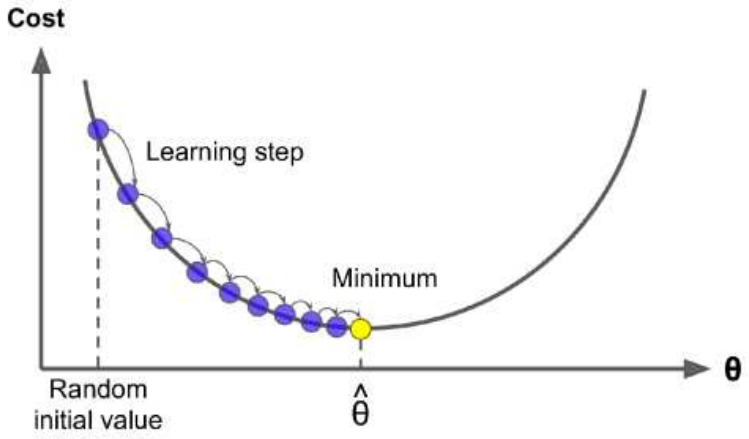
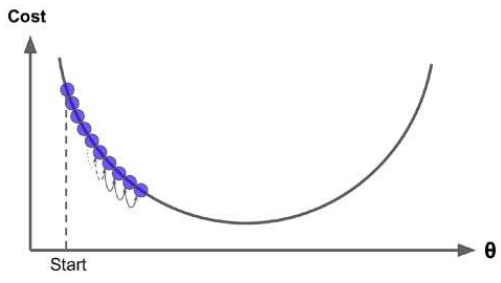
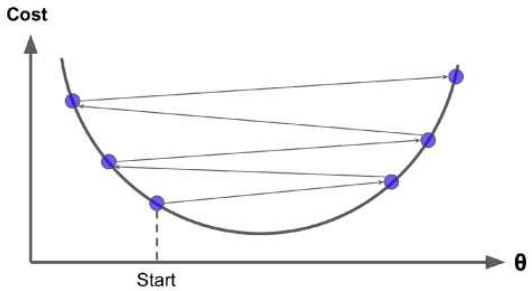


Figura 2.2: Proceso de aproximación al valor óptimo de los parámetros mediante descenso por gradiente [3].

Un parámetro importante en este proceso es el tamaño de los pasos. Se puede modificar mediante un hiperparámetro conocido como tasa de aprendizaje. Si este es demasiado pequeño, el algoritmo requerirá demasiadas iteraciones, mientras que si es demasiado grande existe la posibilidad de no llegar al mínimo. Esto se puede ver gráficamente en la Figura 2.3.



(a) Tasa de aprendizaje demasiado baja.



(b) Tasa de aprendizaje demasiado alta.

Figura 2.3: Ejemplos de descenso por gradiente [3].

No siempre tenemos una función con forma de cuenco (convexa) como en los ejemplos, por lo que podrían aparecer mínimos locales o mesetas que dificulten la búsqueda del mínimo. Por suerte, la función de pérdida logística es convexa y esto asegura que se va a encontrar el mínimo global. Bajo este método se ve claramente que cuanto más grande sea la dimensión del espacio de parámetros, es decir, cuantas mas características se intenten analizar, más difícil será encontrar el mínimo de la función de pérdida y mayor será el coste computacional. Existen varios métodos para implementar el descenso por gradiente. A continuación, se explicará el funcionamiento del descenso por gradiente por lotes. Para empezar, es necesario construir el vector gradiente de la función de pérdida, que no es más que el vector cuyas componentes son las derivadas parciales:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\boldsymbol{\theta}) \\ \\ \frac{\partial}{\partial \theta_1} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\boldsymbol{\theta}) \end{pmatrix} \tag{2.8}$$

Una vez obtenido el vector gradiente, sólo es necesario ir en la dirección opuesta a este para ir "cuesta abajo", es decir;

$$\boldsymbol{\theta}^{\text{(siguiente paso)}} = \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \tag{2.9}$$

En la ecuación anterior, η es la tasa de aprendizaje mencionada anteriormente. Por tanto, sólo es necesario iterar hasta que $\boldsymbol{\theta}$ sufra una modificación de su valor menor que la precisión deseada entre una iteración y la siguiente. Así, queda completamente especificado el funcionamiento de un clasificador por regresión logística.

2.1.2. Máquinas de vectores soporte (SVM)

Las máquinas de vectores soporte o en inglés "Support Vector Machines" (SVM) son uno de los métodos más populares de ML debido a su alta eficacia y fiabilidad con tamaños de datos pequeños y medianos. En este apartado se explican los conceptos básicos de su funcionamiento.

Clasificación SVM lineal

Este tipo de clasificación trata de encontrar el hiperplano de separación con el margen más ancho posible entre las clases. Se muestra un ejemplo con dos clases en la Figura 2.4. La línea continua representa el límite de decisión mientras que las líneas discontinuas delimitan la zona más ancha posible que separa las clases. Los puntos redondeados que marcan las líneas discontinuas se denominan "vectores soporte". Añadir instancias fuera del margen marcado no afectará de ninguna forma al límite de decisión puesto que este solo se ve afectado por los vectores soporte. Esto es lo que se conoce como "clasificación de margen amplio". Además, si se impone que todas las instancias se encuentren fuera de esta calle central se denomina "clasificación de margen duro". Sin embargo, este método

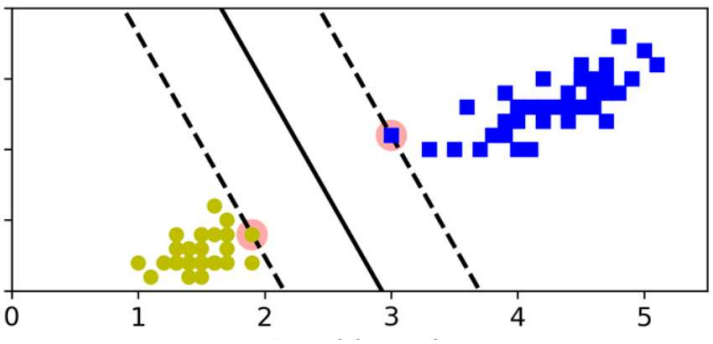


Figura 2.4: Ejemplo de clasificación SVM lineal con 2 clases [3].

sólo permite clasificar datos linealmente separables, lo cual no siempre se cumple. Por tanto, se debe encontrar un equilibrio entre mantener el margen lo más ancho posible y limitar las violaciones del margen; es decir, que no haya demasiados puntos en la zona central o incluso en el lado equivocado. Esto es lo que se llama "clasificación de margen blando". Al igual que ocurría en el caso de la regresión logística, existen una serie de hiperparámetros que permiten seleccionar cuál de estos dos aspectos se debe priorizar y en qué medida. Permitir un mayor número de violaciones del margen puede dar lugar a un clasificador erróneo, pero muchas veces es necesario aceptar estas violaciones con el objetivo de que el clasificador final generalice mejor al resto de situaciones. A diferencia de la regresión logística, las SVM no generan como salida la probabilidad de que una instancia pertenezca a una determinada clase.

Clasificación SVM no lineal

La clasificación SVM lineal es eficaz y muy útil. Sin embargo, las bases de datos pueden ni siquiera acercarse a ser linealmente separables. Esto implica la necesidad de una nueva forma de clasificación. Una idea para manejar conjuntos de datos no linealmente separables es añadir nuevas características. Esto se aprecia claramente en el ejemplo de la Figura 2.5. Al añadir una nueva característica $x_2 = (x_1)^2$ los datos pasan a ser linealmente separables.

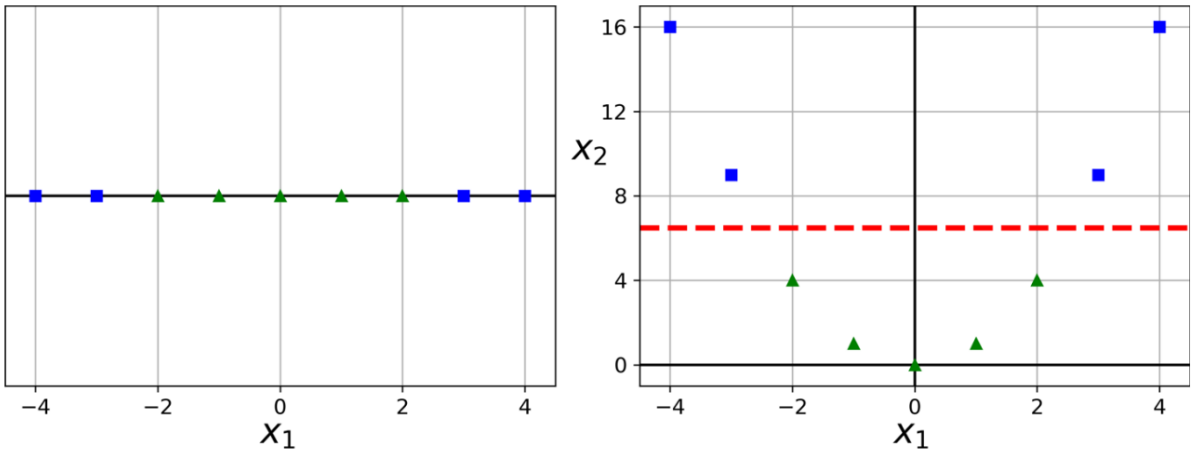


Figura 2.5: Ejemplo de adición de características polinomiales para clasificar conjuntos de datos no linealmente separables [3].

La adición de características polinomiales es sencilla de implementar y da muy buenos

resultados. Como se ha comentado, sin las nuevas características algunos conjuntos de datos no son separables, pero definir demasiadas características adicionales puede dar lugar a un algoritmo demasiado lento. Por suerte, para las SVM existe lo que se conoce como el truco del kernel (*kernel trick*), el cual permite obtener el mismo resultado que si se añadieran nuevas características polinomiales pero sin hacerlo realmente. Por tanto, no es necesario trabajar con un número muy grande de características. El kernel que utiliza características polinomiales se denomina kernel polinomial.

En lugar de añadir nuevas características polinomiales, otra forma de tratar los conjuntos de datos no separables linealmente es utilizar una función de similitud, que mide cuánto se parece cada instancia a un punto de referencia determinado. Por ejemplo, para el caso anterior definimos la función de similitud tal que:

$$\phi_\gamma(\mathbf{x}, l) = \exp(-\gamma||\mathbf{x} - l||^2) \tag{2.10}$$

El valor de la función para cada instancia depende de su distancia al punto de referencia. Para dos puntos de referencia distintos: $\mathbf{x} = -2$ y $\mathbf{x} = 1$, la función es una campana centrada en el propio punto. Es decir, por cada punto de referencia que elijamos tenemos una nueva característica para cada instancia relacionada con la similitud entre dicha instancia y dicho punto de referencia (véase la Figura 2.6).

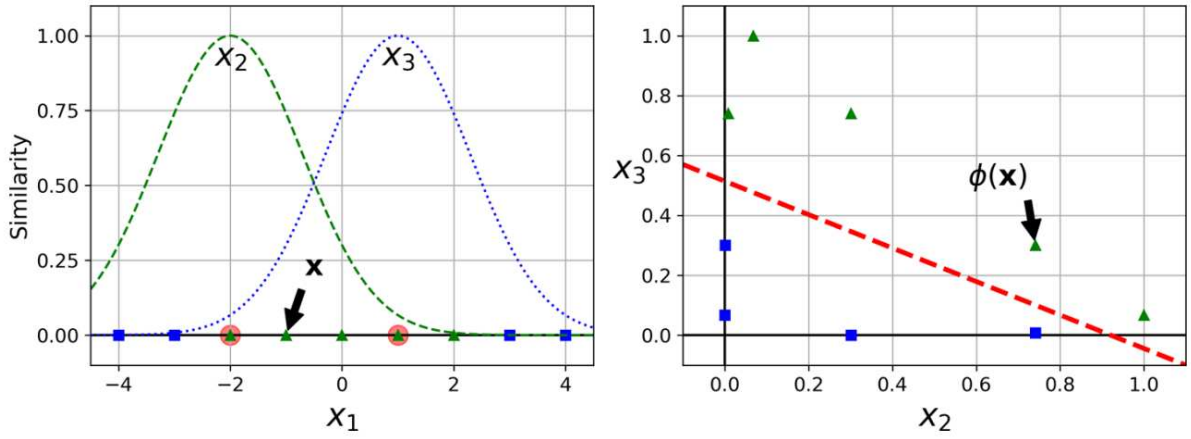


Figura 2.6: Ejemplo de adición de características de similitud para clasificar conjuntos de datos no linealmente separables [3].

Al introducir estas nuevas características de similitud, el conjunto se vuelve linealmente separable. Cuantos más puntos de referencia se tomen (que podría ser todo el conjunto de datos), más nuevas características habrá y mayor es la probabilidad de que el conjunto se vuelva separable. Si el conjunto de datos es muy grande, acabará habiendo un número igual de grande de características, lo cual puede aumentar la complejidad computacional de manera desproporcionada. Al igual que en el caso anterior, el truco del kernel aparece para evitar este problema. En este caso se denomina kernel de base radial gaussiana. También presenta hiperparámetros para ajustar el modelado, entre ellos el parámetro γ de la ecuación (2.10)

Existen otros kernels pero se utilizan con mucha menos frecuencia. Algunos de ellos están diseñados específicamente para estructuras de datos concretas.

Predicciones y funciones de decisión

En esta sección la notación será algo diferente que en la anterior. En la regresión logística el vector de parámetros $\boldsymbol{\theta}$ tenía índices de 0 a n y se fijaba $x_0 = 1$ en el vector de características. Ahora, el valor θ_0 (sesgo) será b y el resto del vector de parámetros será $\mathbf{w}$.

Comenzando por el clasificador SVM lineal, al igual que en la sección anterior, la predicción se realiza mediante la siguiente función de decisión:

$$y = \begin{cases} 0 & \text{si } \mathbf{w}^\intercal \mathbf{x} + b < 0 \\ 1 & \text{si } \mathbf{w}^\intercal \mathbf{x} + b \geq 0 \end{cases} \tag{2.11}$$

El límite de decisión representa los puntos donde $\mathbf{w}^\intercal \mathbf{x} + b = 0$, mientras que las líneas discontinuas, recordando la Figura 2.4, están formadas por los puntos en los que la función de decisión es igual a +1 y -1, respectivamente. Aquí se puede ver que la pendiente de la función de decisión es igual a la norma del vector de peso, $||\mathbf{w}||$. Disminuir esta pendiente implica alejar los puntos en los que la función es igual a ± 1; es decir, cuanto menor sea esta pendiente más ancha será la zona entre las líneas discontinuas. Por tanto, queremos minimizar $||\mathbf{w}||$. Esto es lo mismo que minimizar $\frac{1}{2}\mathbf{w}^\intercal \mathbf{w} = \frac{1}{2}||\mathbf{w}||^2$, que es más sencillo matemáticamente. Además, si queremos evitar la violación del margen es necesario imponer otra condición. La función de decisión debe ser mayor que 1 para todas las instancia de una clase y menor que -1 para las de la otra. Se define:

$$t^{(i)} = \begin{cases} -1 & \text{si } y^{(i)} = 0 \\ 1 & \text{si } y^{(i)} = 1 \end{cases} \tag{2.12}$$

Esto se puede expresar como la restricción $t^{(i)}\left(\mathbf{w}^\intercal \mathbf{x}^{(i)} + b\right) \geq 1$ para todas las instancias. El objetivo entonces es resolver un problema de optimización sujeto a la condición anterior. Si lo que se necesita es un margen blando, será necesario introducir una cierta holgura, $\zeta^{(i)}$, que indique a cada instancia cuánto se le permite violar el margen. Como se ha comentado anteriormente, es necesario un equilibrio entre la anchura máxima del margen, la minimización de $\frac{1}{2}\mathbf{w}^\intercal \mathbf{w}$, y la holgura del margen blando. Para ello, se introduce el hiperparámetro C, que permite definir la compensación entre los dos objetivos. Por tanto, el problema de optimización restringida final es:

$$\begin{aligned} &\text{minimizar:} \quad \frac{1}{2}\mathbf{w}^\intercal \mathbf{w} + C\sum_{i=1}^{m} \zeta^{(i)} \\ &\text{sujeto a:} \quad t^{(i)}\left(\mathbf{w}^\intercal \mathbf{x}^{(i)} + b\right) \geq 1 - \zeta^{(i)} \quad \text{y} \quad \zeta^{(i)} \geq 0 \text{ para } i = 1, 2, \ldots, m \end{aligned} \tag{2.13}$$

Estos problemas se denominan problemas de optimización cuadrática convexa con restricciones lineales. Existen distintas formas de solucionarlos pero no se explican en este trabajo.

Sin embargo, para utilizar el truco del kernel vamos a fijarnos en un problema de optimización restringida diferente.

Problema dual y SVM kernelizadas

Dado un problema de optimización restringido, se puede demostrar que bajo unas determinadas condiciones[1], las cuales cumplen las SVM, existe otro problema de optimización, denominado *problema dual*, que tiene la misma solución que el primero. La forma de este problema dual viene dada por:

$$\begin{aligned} &\text{minimizar:} \quad \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha^{(i)}\alpha^{(j)}t^{(i)}t^{(j)}\mathbf{x}^{(i)\intercal}\mathbf{x}^{(j)} - \sum_{i=1}^{m}\alpha^{(i)} \\ &\text{sujeto a:} \quad \alpha^{(i)} \geq 0 \text{ para } i = 1, 2, \ldots, m \end{aligned} \tag{2.14}$$

donde $\alpha^{(i)}$ se denominan multiplicadores de Karush–Kuhn–Tucker (KKT) y sólo pueden ser mayores o iguales que 0.

Una vez resuelto este problema y habiendo encontrado el $\boldsymbol{\alpha}$ que minimiza la nueva función, la relación entre el problema primario y su dual es la siguiente, donde N_S es el número de vectores soporte:

$$\begin{aligned} \mathbf{w} &= \sum_{i=1}^{m}\alpha^{(i)}t^{(i)}\mathbf{x}^{(i)} \\ b &= \frac{1}{N_S}\sum_{\substack{i=1 \\ \alpha^{(i)}>0}}^{m}\left[t^{(i)} - \mathbf{w}^{\intercal}\mathbf{x}^{(i)}\right] \end{aligned} \tag{2.15}$$

donde $\alpha^{(i)} > 0$ implica sumar sólo sobre las instancias que son vectores soporte, por propiedades de los multiplicadores KKT.

El problema dual es más rápido de resolver que el problema primario para los casos en los que el número de instancias del conjunto de datos es más pequeño que el número de características de cada instancia. Lo más importante es que la existencia del problema dual permite el uso del truco del kernel.

Supóngase que se quieren añadir características polinomiales de segundo orden al conjunto de datos. Estas nuevas características vendrán dadas, en el caso de un polinomio de segundo orden, por la siguiente función:

$$\phi(\mathbf{x}) = \phi\left(\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}\right) = \begin{pmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{pmatrix} \tag{2.16}$$

Véase que el resultado tiene 3 dimensiones; es decir, se ha aumentado en uno la dimensión del espacio de características. Si se calcula el producto escalar de 2 vectores transformados:

$$\begin{aligned} \phi(\mathbf{a})^{\intercal}\phi(\mathbf{b}) &= \begin{pmatrix} a_1^2 & \sqrt{2}a_1a_2 & a_2^2 \end{pmatrix}\begin{pmatrix} b_1^2 \\ \sqrt{2}b_1b_2 \\ b_2^2 \end{pmatrix} = a_1^2b_1^2 + 2a_1b_1a_2b_2 + a_2^2b_2^2 = \\ &= (a_1b_1 + a_2b_2)^2 = \left(\begin{pmatrix} a_1 & a_2 \end{pmatrix}\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}\right)^2 = (\mathbf{a}^{\intercal}\mathbf{b})^2 \end{aligned} \tag{2.17}$$

[1] La función objetivo debe ser convexa y las restricciones del problema deben ser continuamente diferenciables y convexas.

Esto implica que si se aplica la función ϕ sobre todas las instancias $\mathbf{x}^{(i)}$ entonces la función a minimizar en el problema dual, véase la ecuación (2.14), incluirá el producto $\phi(\mathbf{x}^{(i)})^\intercal\phi(\mathbf{x}^{(j)})$. Pero este es igual al cuadrado del producto sin aplicar la transformación, $\left(\mathbf{x}^{(i)\intercal}\mathbf{x}^{(j)}\right)^2$; es decir, para resolver el problema dual para obtener el vector de parámetros utilizando las características polinomiales no es necesario aplicar la función que da las características polinomiales a cada instancia. En esto consiste precisamente el truco del kernel. El resultado será exactamente el mismo que si se calculasen todas las nuevas características y se resolviese el correspondiente problema de optimización.

Por tanto, se define una función *kernel* como aquella que es capaz de calcular el producto escalar de dos vectores transformados a partir de los vectores iniciales. Para un polinomio de segundo orden el *kernel* es $K(\mathbf{a},\mathbf{b}) = (\mathbf{a}^\intercal\mathbf{b})^2$, pero existen otras funciones similares dependiendo de las nuevas características que se quieran implementar. Aquí se resumen algunas de las más importantes:

$$\text{Lineal:} \quad K(\mathbf{a},\mathbf{b}) = \mathbf{a}^\intercal\mathbf{b} \tag{2.18}$$

$$\text{Polinomial:} \quad K(\mathbf{a},\mathbf{b}) = (\gamma\mathbf{a}^\intercal\mathbf{b} + r)^d \tag{2.19}$$

$$\text{Base radial Gaussiana:} \quad K(\mathbf{a},\mathbf{b}) = \exp(-\gamma||\mathbf{a}-\mathbf{b}||^2) \tag{2.20}$$

Para hacer predicciones con el truco del kernel además es necesario modificar la ecuación (2.15) de manera que involucre productos escalares. Resolviendo esto se obtiene:

$$\begin{aligned} h_{\mathbf{w},b}\left(\phi\left(\mathbf{x}^{(n)}\right)\right) = \mathbf{w}^\intercal\phi\left(\mathbf{x}^{(n)}\right) + b = \left(\sum_{i=1}^{m}\alpha^{(i)}t^{(i)}\phi\left(\mathbf{x}^{(i)}\right)\right)^\intercal\phi\left(\mathbf{x}^{(n)}\right) + b = \\ = \sum_{i=1}^{m}\alpha^{(i)}t^{(i)}\left(\phi\left(\mathbf{x}^{(i)}\right)^\intercal\phi\left(\mathbf{x}^{(n)}\right)\right) + b = \\ = \sum_{\substack{i=1\\ \alpha^{(i)}>0}}^{m}\alpha^{(i)}t^{(i)}K(\mathbf{x}^{(i)},\mathbf{x}^{(n)}) + b \end{aligned} \tag{2.21}$$

donde en el último paso se ha usado el hecho de que $\alpha^{(i)} \neq 0$ sólo para vectores soporte y se puede reducir la suma sólo a estos ($\alpha^{(i)} > 0$). Llevando a cabo el mismo procedimiento para el término de sesgo:

$$\begin{aligned} b = \frac{1}{N_S}\sum_{\substack{i=1\\ \alpha^{(i)}>0}}^{m}\left(t^{(i)} - \mathbf{w}^\intercal\phi\left(\mathbf{x}^{(i)}\right)\right) = \\ = \frac{1}{N_S}\sum_{\substack{i=1\\ \alpha^{(i)}>0}}^{m}\left(t^{(i)} - \left(\sum_{j=1}^{m}\alpha^{(j)}t^{(j)}\phi\left(\mathbf{x}^{(j)}\right)\right)^\intercal\phi\left(\mathbf{x}^{(i)}\right)\right) = \\ = \frac{1}{N_S}\sum_{\substack{i=1\\ \alpha^{(i)}>0}}^{m}\left(t^{(i)} - \sum_{\substack{j=1\\ \alpha^{(j)}>0}}^{m}\alpha^{(j)}t^{(j)}K(\mathbf{x}^{(i)},\mathbf{x}^{(j)})\right) \end{aligned} \tag{2.22}$$

Las ecuaciones (2.21) y (2.22) describen ambos términos en función de las variables que el truco del kernel permite sustituir. Así, se puede obtener el resultado del aprendizaje con las características añadidas sin tener que calcularlas explícitamente.

2.1.3. Árboles de decisión

Los árboles de decisión, al igual que las SVM, son uno de los métodos de ML más versátiles y más ampliamente utilizados. Pueden aplicarse tanto a problemas de regresión como de clasificación pero este apartado se centrará en su aplicación a la clasificación binaria.

Un árbol de decisión trata de distribuir un conjunto de puntos en subconjuntos disjuntos en función de las respuestas a diferentes preguntas relacionadas con las características. Así, cada pregunta corresponde a un nodo del árbol y en función de la respuesta se construyen los diferentes nodos hijo, cada uno conteniendo un subconjunto disjunto del conjunto de datos representado en el nodo padre. En la Figura 2.7 se muestra un ejemplo para determinar si una papaya es sabrosa o no. El proceso da lugar a una estructura de árbol para tomar la decisión y de ahí el nombre de este método.

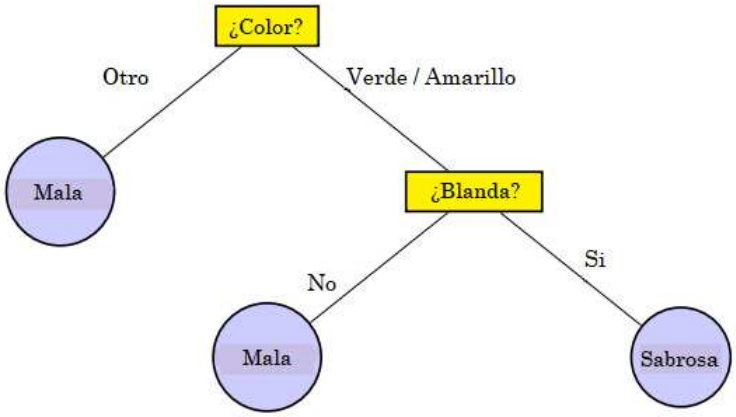


Figura 2.7: Ejemplo de árbol de decisión [4].

Por tanto, en el entrenamiento se construye una estructura de este tipo. Después, las instancias de test se hacen pasar por esta serie de preguntas hasta que se llega a un nodo final que presenta la predicción para dicha instancia. Al igual que en la regresión logística, los árboles de decisión permiten estimar la probabilidad de que una instancia pertenezca a una clase particular. Para ello, una vez se llega al nodo final que da la predicción, se calcula el ratio de las instancias de entrenamiento que acaban en dicho nodo y son de la clase dada, frente a todas las instancias que acaban en ese nodo.

El algoritmo utilizado por *Scikit-Learn* para el entrenamiento de los árboles de decisión se llama "Classification and Regression Trees" (CART). Funciona dividiendo el primer conjunto de entrenamiento en dos subconjuntos a partir de una de las características k y un umbral t_k. La función de pérdida a minimizar es la siguiente:

$$J(k, t_k) = \frac{m_{\text{izq}}}{m} G_{\text{izq}} + \frac{m_{\text{dcha}}}{m} G_{\text{dcha}} \tag{2.23}$$

Donde $G_{\text{izq/dcha}}$ miden la pureza los subconjuntos izq/dcha y $m_{\text{izq/dcha}}$ es el número de instancias de cada subconjunto. Se dice que un conjunto es "puro" si todas las instancias de dicho conjunto pertenecen a la misma clase. Existen dos formas de medir la pureza: la

impureza de Gini, ecuación (2.24) y la entropía, ecuación (2.25):

$$G_i = 1 - \sum_{k=1}^{n} p_{i,k}^2 \tag{2.24}$$

$$H_i = - \sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log_2(p_{i,k}) \tag{2.25}$$

donde $p_{i,k}$ es el ratio de instancias de clase k entre todas las del conjunto i. En general, se suele utilizar la impureza Gini porque es más rápida de calcular.

Una vez el algoritmo ha separado el conjunto original de muestras en dos subconjuntos, divide cada uno de ellos en otros dos utilizando la misma lógica, y así sucesivamente. La operación cesa cuando se alcanza el número máximo de nodos impuesto previamente o cuando la separación no reduce la impureza del conjunto inicial. Existen algunos otros parámetros que controlan también las condiciones de detención. El algoritmo CART hace una búsqueda voraz de una división óptima del nivel superior sin comprobar si dicha división conducirá o no a la menor impureza posible de los niveles inferiores. Generalmente, un algoritmo voraz de este tipo produce con frecuencia una solución razonablemente buena, pero sin garantías de que sea óptima.

Una característica de los árboles de decisión es que, aunque el entrenamiento pueda ser computacionalmente costoso, las predicciones se realizan muy rápidamente. Los árboles de decisión suelen ser aproximadamente equilibrados, por lo que atravesar uno hasta llegar a un nodo final que dé la predicción requiere recorrer del orden de $O\left(\log_2(m)\right)$ nodos.

Un problema típico de los árboles de decisión es el sobreajuste; esto es, la generación de un árbol que se ajusta muy bien al conjunto de entrenamiento pero que no es capaz de generalizar a problemas nuevos de manera eficaz. Para evitar el sobreajuste existen los llamados hiperparámetros de regularización. Estos hiperparámetros dependen del algoritmo en cuestión; en *Scikit-Learn* se han implementado los siguientes:

- `max_depth`: restringe la profundidad máxima del árbol.
- `min_samples_split`: número mínimo de muestras que debe tener un nodo antes de dividirse.
- `min_samples_leaf`: número mínimo de muestras que debe tener un nodo terminal.
- `min_weight_fraction_leaf`: igual que `min_samples_leaf` pero expresado como una fracción del número total de instancias.
- `max_leaf_nodes`: número máximo de nodos finales.
- `max_features`: número máximo de características que se evalúan para dividir en cada nodo.

En la Figura 2.8 se muestra un ejemplo de aplicación de estos hiperparámetros para evitar el sobreajuste.

Los árboles de decisión tienen mucho que ofrecer, son sencillos de interpretar y fáciles de usar. Sin embargo, tienen algunas limitaciones. Debido a que las separaciones se realizan

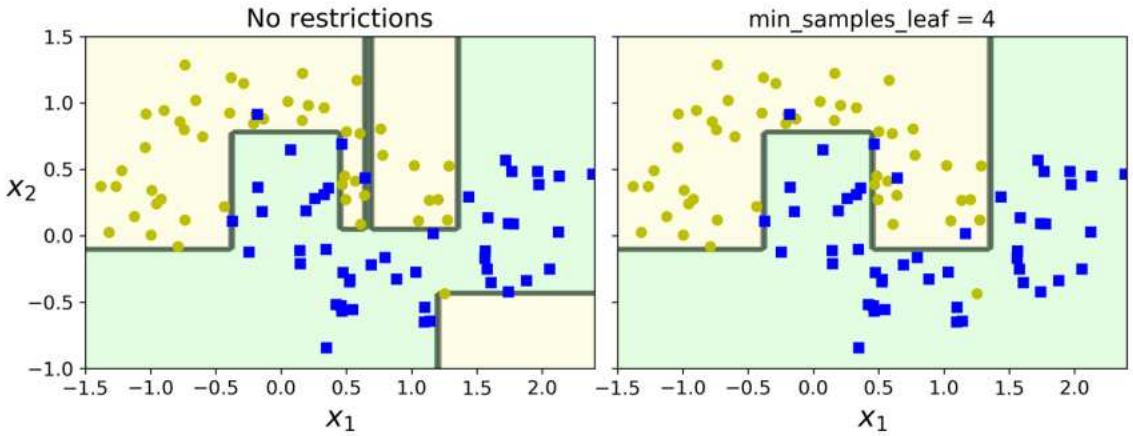


Figura 2.8: Ejemplo de regularización para evitar el sobreajuste. Es obvio que el modelo de la izquierda está sobreajustado.

mediante umbrales en las características, los límites de decisión son siempre ortogonales, por lo que son sensibles a la rotación del conjunto de entrenamiento. Pero el principal problema de los árboles de decisión es que son muy sensibles a pequeñas variaciones en los datos de entrenamiento. Como se verá en el siguiente apartado, esta inestabilidad puede ser compensada mediante lo que se conoce como *random forests*, haciendo la media de las predicciones de varios árboles.

2.1.4. Combinación de modelos y *random forests*

La idea básica de la combinación de modelos (*ensemble models*) es agregar las predicciones dadas por un grupo de predictores para obtener una predicción mejor que cada una de las predicciones individuales. Existen diferentes métodos de combinación; en concreto, si se entrena un grupo de árboles de decisión, cada uno sobre un subconjunto aleatorio diferente del conjunto de entrenamiento, el conjunto de modelos resultante se denomina *random forest*. Pese a su simplicidad, este método de combinación es uno de los más potentes en ML.

Lo propio es usar métodos de combinación al final de un proyecto, una vez se hayan creado unos buenos predictores individuales.

Clasificadores por votación

La manera más simple de obtener un predictor por combinación consiste en reunir las predicciones de una serie de predictores de distinto tipo y dar como predicción final aquella que reciba más votos. Este clasificador por votación tendrá un resultado mejor que cada uno de los clasificadores individuales. Esto se denomina "hard voting".

Si todos los clasificadores individuales son capaces de calcular la probabilidad de que una instancia pertenezca a una clase, se puede buscar la clase con la probabilidad más alta promediada entre todos los clasificadores. Esto se denomina "soft voting" y a menudo es más eficaz que el método anterior porque da más peso a los votos más seguros.

Bagging y *pasting*

En el caso anterior, se entrenaban clasificadores de distinto tipo (usando diferentes técnicas de ML) haciendo uso de todo el conjunto de entrenamiento. Otro enfoque es utilizar la misma técnica de ML para entrenar distintos clasificadores utilizando diferentes subconjuntos aleatorios del conjunto de entrenamiento. Si el muestreo se realiza con reemplazo, esto es, si se pueden repetir instancias, el método se denomina *bagging*; si el muestreo no permite el reemplazo, se denomina *pasting*. Una vez los clasificadores están entrenados, se pueden realizar predicciones de la misma forma que se ha visto anteriormente. La principal ventaja de este método es que los diferentes clasificadores individuales se pueden entrenar de forma paralela, lo que permite un buen escalado a conjuntos de entrenamiento de mayor tamaño.

Random forests

Como ya se ha mencionado, un *random forest* es un conjunto de árboles de decisión generalmente entrenados por el método *bagging*. Además, para añadir aleatoriedad en la construcción de los árboles individuales, en vez de buscar la mejor característica para dividir un nodo, se selecciona de forma aleatoria una característica de entre un subconjunto de todas ellas. Así, se obtiene una mayor diversidad de árboles que puede conseguir un mejor rendimiento al combinar las predicciones de todos los árboles.

Existe otra forma de aumentar la aleatoriedad de los árboles, que es utilizar también umbrales aleatorios para cada característica. Esto da lugar a un bosque de árboles extremadamente aleatorios que se denomina *Extra-Trees*.

Otro punto a favor de los *random forests* es que hacen sencillo medir la importancia relativa de cada una de las características. Para ello, es necesario fijarse en cuánto reducen la impureza, de media, los nodos de los árboles que utilizan una característica concreta a través de todos los árboles del bosque. Los *random forests* permiten así llevar a cabo una selección de características.

2.1.5. *k* vecinos más cercanos

Los métodos de los k vecinos más cercanos, comúnmente conocidos como k-NN (por sus siglas en inglés), son de los más simples en ML. Su funcionamiento se basa en asignar a cada nueva instancia la clase de la instancia o instancias de entrenamiento más cercanas en el espacio de características. Este método será útil si las instancias de una misma clase tienden a estar juntas. Una de sus principales ventajas es que, incluso para bases de datos inmensas, encontrar el vecino más cercano es un proceso muy rápido [4].

Para poder decir si dos instancias pertenecientes a un dominio $\mathscr{X}$ son cercanas, se introduce el concepto de métrica, ρ. Esto es, $\rho : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ es una función que devuelve la "distancia" entre dos instancias. Por ejemplo en el caso $\mathscr{X} = \mathbb{R}^3$ se tiene la métrica euclídea, $\rho(\mathbf{x}, \mathbf{x}') = ||\mathbf{x} - \mathbf{x}'|| = \sqrt{\sum_{i=1}^{3}(x_i - x_i')^2}$.

Una vez definida la métrica, se tiene un conjunto de instancias de la forma $S = \{(\mathbf{x_i}, y_i)\}$, donde $\mathbf{x_i}$ es el vector de características e y_i la etiqueta de su clase, Para cada instancia

$\mathbf{x}$ se crea una lista ordenada de los subíndices del resto de instancias $\mathbf{x_j}$ en función de su distancia a $\mathbf{x_i}$, de menor a mayor. Esta lista es $\pi_1(\mathbf{x}), \ldots, \pi_m(\mathbf{x})$, donde m es el número de instancias de S. Es decir, se cumple

$$\rho(\mathbf{x}, \mathbf{x}_{\pi_i(\mathbf{x})}) \leq \rho(\mathbf{x}, \mathbf{x}_{\pi_{i+1}(\mathbf{x})}) \tag{2.26}$$

Bajo esta notación, la regla de los k vecinos más próximos construye un mapa tal que a cada punto $\mathbf{x} \in \mathscr{X}$ se le asigna la etiqueta mayoritaria en $\{y_{\pi_i(\mathbf{x})} : i \leq k\}$; esto es, la etiqueta que más se repita entre las k instancias más cercanas al punto. Un ejemplo de este mapa para el caso 1-NN se puede ver en la Figura 2.9.

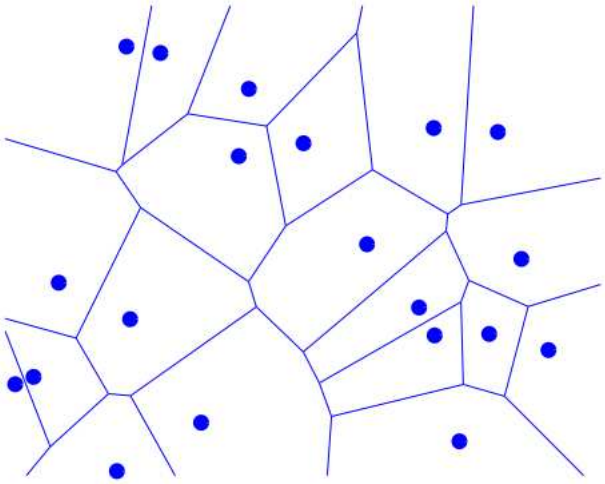


Figura 2.9: Ejemplo de mapa con los límites de decisión para la regla 1-NN [4].

2.2. Técnicas de compensación de datos no balanceados

En numerosas ocasiones, aplicar el método de clasificación no es la parte más difícil del problema, sino que son más complicados los pasos previos. Esto se debe a que el conjunto de datos puede no ser adecuado para obtener unos buenos resultados. Por ello, es necesario preparar los datos antes de proceder a la clasificación. En el problema abordado en este trabajo, por las características del conjunto de datos, lo más importante será tratar el desequilibrio entre instancias positivas y negativas.

2.2.1. Submuestreo y sobremuestreo aleatorios

Submuestreo de la clase mayoritaria

Submuestrear consiste en eliminar de forma aleatoria instancias de la clase mayoritaria hasta obtener el ratio deseado de instancias entre ambas clases. Esta es una buena opción cuando hay muchos datos pero es necesario ser consciente de que eliminar instancias implica la pérdida de información.

Sobremuestreo de la clase minoritaria

Con el mismo objetivo que en el caso anterior, este método consiste en añadir nuevas instancias de la clase minoritaria hasta obtener el ratio deseado. Es una buena alternativa cuando hay pocos datos para trabajar. Un punto en contra importante es que el sobremuestreo puede provocar un sobreajuste del clasificador, haciendo que este generalice peor al conjunto de test.

2.2.2. Sobremuestreo sintético de la minoría (SMOTE)

A diferencia de las técnicas anteriores, en lugar de añadir instancias de forma aleatoria, la técnica SMOTE crea instancias sintéticas. Las nuevas instancias se crean realizando diferentes cálculos sobre el conjunto de datos. Estos cálculos pueden elegirse de forma específica para la aplicación del conjunto de datos; sin embargo, la forma más general es trabajar sobre el espacio de características de las instancias. Esta técnica se basa en añadir instancias de la clase minoritaria en las rectas que unen cada una de las instancias de dicha clase con sus k vecinos más cercanos de esa misma clase. Dependiendo del nivel de sobremuestreo deseado, se elige un mayor o menor número de estos k vecinos, de forma aleatoria, para la generación de cada nueva instancia sintética. Por ejemplo, si el algoritmo utiliza $k = 5$ vecinos más próximos y se desea un sobresampleo del 200 %, se eligen 2 de los 5 primeros vecinos de forma aleatoria y se crea una instancia en cada una de las dos direcciones dadas por dichos vecinos [5].

Las instancias sintéticas se generan de la siguiente forma:

1) Se toma la diferencia entre una de las instancias de la clase minoritaria y su vecino más próximo.

2) Se multiplica la diferencia por un número aleatorio entre 0 y 1 y se le suma a la instancia inicial.

3) Se tiene así un punto aleatorio en la recta que une ambas instancias.

Combinación de SMOTE y submuestreo

El método más eficaz a la hora de equilibrar un conjunto de datos es una combinación entre eliminar instancias de la clase mayoritaria y añadirlas en la clase minoritaria mediante SMOTE. La aplicación de un submuestreo previo sobre la clase mayoritaria hace que el sesgo inicial del clasificador hacia la clase mayoritaria se vea revertido en favor de la clase minoritaria.

3. Evaluación del rendimiento de los modelos

Tras estudiar las técnicas que se aplicarán posteriormente al conjunto de datos, es fundamental definir diferentes medidas para caracterizar el rendimiento o eficacia de los modelos en la tarea de clasificación propuesta. En este capítulo se presentan los conceptos más importantes para determinar la eficacia de un modelo [6].

3.1. Métricas de rendimiento en tareas de clasificación

3.1.1. Matriz de confusión

La matriz de confusión separa los datos sobre los que se prueba el modelo en función de su valor predicho y su valor real. En este trabajo, se tiene una clasificación binaria; es decir, se puede predecir que una instancia pertenezca a la clase negativa o a la clase positiva, que corresponderían a los valores 0 y 1, respectivamente. Un valor 0 predice que no existe exoplaneta orbitando a la estrella y un valor 1 que sí existe exoplaneta. La matriz de confusión recoge entonces todas las opciones posibles (véase la Figura 3.1).

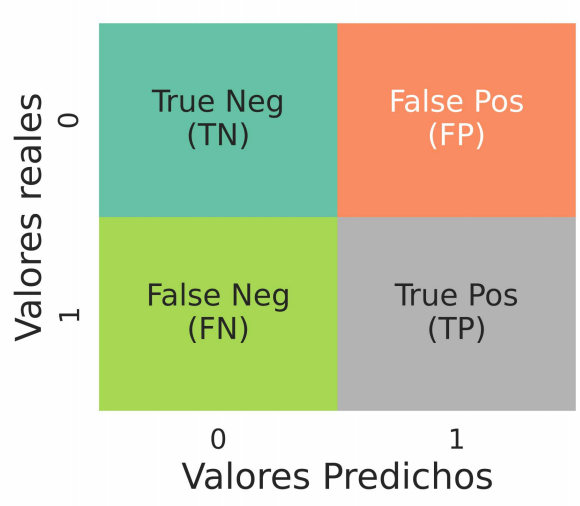


Figura 3.1: Ejemplo de matriz de confusión.

En función de los valores predichos y los reales, se tienen 4 posibilidades, cada una representada en una esquina de la matriz de confusión:

- Verdaderos Negativos (*True Negatives*) (TN): Aquellas instancias predichas como falsas (0) y que efectivamente son falsas. Representadas en la esquina superior iz-

quierda de la matriz.

- Verdaderos Positivos (*True Positives*) (TP): Aquellas instancias predichas correctamente como positivas (1). Corresponden a la esquina inferior derecha de la matriz.
- Falsos Negativos (*False Negatives*) (FN): Aquellas instancias predichas como negativas (0) pero que realmente son positivas (1). Representadas en la esquina inferior izquierda de la matriz.
- Falsos Positivos (*False Positives*) (FP): Aquellas instancias predichas como positivas (1) pero que realmente son negativas (0). Representadas en la esquina superior derecha de la matriz.

La matriz de confusión es útil en la mayoría de los casos ya que permite ver el número nominal de instancias que se predicen correctamente o no en vez de un porcentaje.

3.1.2. Ratio de Falsos Positivos — Error tipo I

El ratio de falsos positivos (*False Positive Rate*) (FPR) se puede entender como la fracción de falsas alarmas que se producen. Se define como

$$FPR = \frac{FP}{FP + TN} \tag{3.1}$$

Obviamente, si se disminuye el umbral para considerar una instancia positiva, entonces se obtendrán más instancias positivas. De esta forma, es posible asegurar que todas las instancias positivas sean detectadas, a costa de obtener un mayor número de falsas alarmas.

3.1.3. Ratio de Falsos Negativos — Error tipo II

El ratio de falsos negativos (*False Negative Rate*) (FNR) se puede entender como la fracción de casos positivos sin detectar; en este trabajo, el número de exoplanetas que no se detectan. Se define como

$$FNR = \frac{FN}{FN + TP} \tag{3.2}$$

En este caso, aumentar el umbral implica predecir un menor número de instancias positivas; es decir, sólo se marcarán como positivas las instancias que presenten características muy claras de ser positivas. Esto aumentará el número de falsos negativos porque las instancias positivas que no presenten una probabilidad demasiado alta de ser positivas en la predicción serán catalogadas como negativas.

3.1.4. Ratio de Verdaderos Negativos — Especificidad

El ratio de verdaderos negativos (*True Negative Rate*) (TNR) mide la cantidad de instancias negativas que se han predicho correctamente como negativas. Se define como

$$TNR = \frac{TN}{FP + TN} \tag{3.3}$$

Como se ha mencionado anteriormente, el conjunto de datos a estudiar es no balanceado. Esto significa que existen muchas más instancias negativas que positivas. Por tanto, clasificar una instancia negativa correctamente será fácil; es decir, se esperan especificidades altas para todos los modelos. Por esta razón, el ratio de verdaderos negativos no será útil para seleccionar un modelo por encima de otro.

3.1.5. Ratio de Verdaderos Positivos — Recall

El ratio de verdaderos positivos (*True Positive Rate*) (TPR) mide la cantidad de instancias positivas que se han predicho correctamente como positivas. Se define como

$$TPR = \frac{TP}{FN + TP} \tag{3.4}$$

Maximizar el recall es el factor más importante cuando es absolutamente necesario catalogar correctamente las instancias positivas. En este trabajo significa asegurarse de predecir correctamente los casos que sí presentan exoplanetas, a pesar de aumentar el número de falsas alarmas. Un ejemplo claro para entender cuándo esto es necesario es el de una enfermedad muy contagiosa en la que es vital no dejar pasar a ningún enfermo.

3.1.6. Valor de predicción positiva — Precisión

El valor de predicción positiva (*Positive Predictive Value*) (PPV) mide la cantidad de instancias positivas que se han predicho correctamente entre todas las instancias que se han catalogado como positivas. Se define como

$$PPV = \frac{TP}{FP + TP} \tag{3.5}$$

Maximizar la precisión significa que cada vez que se predice una instancia como positiva, esta sea realmente positiva.

3.1.7. Exactitud — *Accuracy*

La exactitud (ACC) mide la cantidad de instancias, tanto positivas como negativas, que se han predicho correctamente. Se define como

$$ACC = \frac{TP + TN}{FP + TP + FN + TN} \tag{3.6}$$

No es apropiado utilizar la exactitud en problemas no balanceados, ya que la clase mayoritaria será fácil de predecir, obteniendo así un resultado de $ACC > 0.9$ para cualquier modelo simple, no necesariamente bueno. Esto es precisamente lo que ocurre en el caso del conjunto de datos de los exoplanetas, en el que hay muchas más instancias negativas. En esta situación, incluso un modelo trivial que prediga todas las instancias como negativas será capaz de obtener una exactitud alta.

3.1.8. Puntuación f_1

Permite tener en cuenta la precisión y el recall al mismo tiempo. Es la media armónica de ambos:

$$f_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3.7}$$

Es uno de los valores más utilizados porque permite buscar un equilibrio entre los positivos detectados y la precisión a la hora de dar una predicción positiva.

3.1.9. Puntuación f_β

Se trata de una generalización del f_1 de forma que se puede dar más importancia a la precisión o al recall en función del valor de β. Se define como

$$f_1 = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall} \tag{3.8}$$

Valores de $0 < \beta < 1$ dan más importancia a la precisión mientras que valores de $\beta > 1$ dan más importancia al recall. Por ejemplo, es comúnmente utilizado el valor f_2 en el que se da al recall el doble de importancia que a la precisión.

3.1.10. Curva ROC

La curva ROC (*Receiver Operating Characteristic*) representa el TPR en función del FPR. Cada punto de la curva representa un umbral distinto al tomar la decisión de clasificación por un cierto sistema. La mejor opción es un valor alto de TPR con un valor bajo de FPR; es decir, una curva ROC será mejor cuanto más se acerque a la esquina superior izquierda (véase la Figura 3.2).

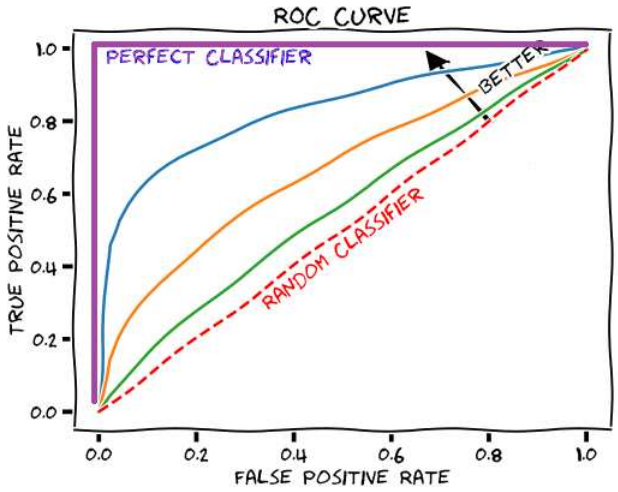


Figura 3.2: Ejemplo de curvas ROC para distintos sistemas clasificadores. La línea diagonal representa un clasificador aleatorio. La línea morada representa un clasificador perfecto [7].

Con objetivo de tener un único número que represente la calidad de esta curva se puede calcular el área bajo la misma. Cuanto más cerca esté la curva de la esquina superior izquierda, mayor será el área. Se denomina puntuación ROC AUC (*Area Under Curve*). Esta puntuación no es demasiado útil en problemas no balanceados [8].

3.1.11. Curva precisión-recall (PR)

Es la curva que representa la precisión (PPV) en función del recall (TPR). Es obvio que a mayor recall peor será la precisión del clasificador. Visualizar esta curva permite ver a partir de qué recall la precisión empieza a caer drásticamente, permitiendo así obtener el modelo más equilibrado o más ajustado a las necesidades de cada tarea. Al igual que en el caso anterior, se puede calcular el área bajo la curva para obtener un número que refleja la calidad del clasificador. Ya que el valor PR AUC se centra más en la clase positiva, es más útil que la ROC AUC en problemas no balanceados.

3.2. Validación cruzada

La validación cruzada o *cross validation* es una técnica que permite poner a prueba un modelo sobre distintas particiones de los datos. Es necesario recordar que del conjunto total de datos recogidos se realiza una primera separación en un conjunto de entrenamiento y un conjunto de test. El conjunto de test se debe dejar intacto hasta que los modelos estén preparados para probarse ante muestras nunca vistas por los mismos. Sin embargo, para diseñar un buen modelo hay que probarlo también sobre datos no utilizados en su entrenamiento. El método de validación cruzada consiste en separar el conjunto de entrenamiento en k subconjuntos, de forma que en cada una de las iteraciones se entrena el modelo con todos los subconjuntos menos uno, conocido como conjunto de validación, que es con el que se prueba el modelo (véase la Figura 3.3).

En cada iteración, se obtiene un valor de las métricas elegidas para la caracterización del rendimiento del modelo, pudiendo así calcular la media sobre todas las particiones de la validación cruzada. Si un modelo es capaz de obtener buenos resultados en todas las particiones, sus predicciones serán fiables cuando se aplique a datos nuevos.

Figura 3.3: Método de validación cruzada en k particiones.

4. Clasificación de los datos reales

En este capítulo se aplican los métodos explicados en capítulos anteriores al conjunto de datos reales medidos por el telescopio. En primera instancia, será necesario realizar una preparación previa de los datos. A continuación, se procederá a obtener los modelos de clasificación para su posterior análisis y evaluación. El código utilizado se puede consultar en el siguiente repositorio de *GitHub*: https://github.com/AdrianMelendez/ML_Exoplanets.git

4.1. Preparación de los datos

Los datos se han obtenido de la plataforma *Kaggle* [1]. Este conjunto de datos reúne series temporales del flujo de luz recibido desde diferente estrellas observadas. En concreto, se reúnen los datos recibidos desde 5087 estrellas para cada una de las cuales se han tomado 3198 medidas del flujo de luz recibido. De las 5087 estrellas observadas, sólo 37 están confirmadas con presencia de exoplanetas. Esto implica un gran desequilibrio entre la clase positiva y la clase negativa, desequilibrio que será necesario tener en cuenta en los experimentos de clasificación.

A partir de estas series temporales es necesario obtener las características que se utilizarán en el entrenamiento de los modelos. Para ello, se hará uso del módulo *tsfresh*[2] de *Python*. Este módulo permite calcular un gran número de características de una serie temporal; por ejemplo, la media o el máximo, además de muchas otras de mayor complejidad.

Aplicando *tsfresh* se han extraído 96 parámetros de cada una de las series temporales. Serán estos parámetros los que se utilizarán como datos de entrada en la tarea de clasificación.

4.2. Resultados del entrenamiento

4.2.1. Regresión logística

Para el caso de la regresión logística será necesario realizar un escalado (normalización) de los datos, de tal forma que la media de los valores sea 0 y su desviación estándar sea 1; es decir, se resta el valor medio y se divide por la desviación típica.

A continuación, usando el método de validación cruzada, se calculan la precisión, el

[1] https://www.kaggle.com/datasets/keplersmachines/kepler-labelled-time-series-data
[2] https://tsfresh.readthedocs.io/en/latest/index.html

recall y la puntuación f_4. En este trabajo se ha elegido esta puntuación f_4 como la métrica más importante a tener en cuenta. Esto se debe a que esta puntuación da 4 veces más importancia al recall que a la precisión. De esta forma, el modelo será capaz de predecir correctamente la mayoría de instancias positivas a pesar de aumentar el número de falsas alarmas. Es importante destacar que, debido al desequilibrio de los datos (sólo el 0.73 % de los mismos es positivo), será necesario realizar una validación cruzada que mantenga este porcentaje entre clases.

También se compararán los resultados obtenidos para el conjunto de datos desequilibrado con los obtenidos aplicando el método SMOTE o el submuestreo + SMOTE (véase la Tabla 4.1). El sobremuestreo SMOTE se ha realizado hasta obtener un ratio entre muestras positivas y negativas igual a 1. Por otro lado, el submuestreo aleatorio se ha realizado hasta conseguir un ratio de 0.01. Se puede observar que la aplicación del método SMOTE permite aumentar el recall y la puntuación f_4; sin embargo, el submuestreo no ofrece mejores resultados.

Tabla 4.1: Resultados de la validación cruzada con $k = 5$ usando el modelo de regresión logística.

	Desequilibrado	SMOTE	Submuestreo + SMOTE
Recall	0.46 ± 0.20	0.73 ± 0.12	0.73 ± 0.12
Precisión	0.95 ± 0.10	0.40 ± 0.07	0.37 ± 0.06
f_4	0.47 ± 0.20	0.70 ± 0.11	0.69 ± 0.11

Los resultados anteriores se han conseguido sin alterar el hiperparámetro C de la regresión logística, que toma el valor $C = 1$ por defecto. Por ello, se ha buscado el valor de este hiperparámetro que mejor resultado ofrezca en la validación cruzada para el caso de la aplicación de SMOTE sobre el conjunto de datos. Se obtienen los resultados de la Tabla 4.2.

Tabla 4.2: Resultado de la validación cruzada usando el modelo de regresión logística y aplicando el método SMOTE para distintos valores del hiperparámetro C.

C	Recall	Precisión	f_4
1	0.73 ± 0.12	0.40 ± 0.07	0.70 ± 0.11
2	0.73 ± 0.12	0.48 ± 0.11	0.71 ± 0.11
3	0.73 ± 0.12	0.55 ± 0.14	0.72 ± 0.11
4	0.76 ± 0.16	0.57 ± 0.13	0.74 ± 0.15
5	0.74 ± 0.18	0.56 ± 0.13	0.72 ± 0.17

De los resultados se deduce que el valor que mejores resultados ofrece es $C = 4$. En la Figura 4.1 se muestra la matriz de confusión en una de las iteraciones de la validación cruzada. Para obtener esta matriz, se ha entrenado y probado el modelo con $C = 4$, usando los conjuntos de entrenamiento y validación correspondientes en una de las iteraciones de la validación cruzada.

Se observa que en la partición hay 8 casos positivos, de los cuales 5 se detectan correctamente y 3 no. Aunque se podría modificar el diseño del modelo para obtener un

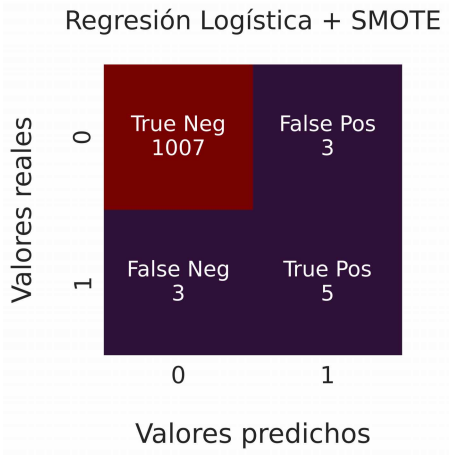


Figura 4.1: Matriz de confusión del modelo de regresión logística aplicando el sobremuestreo SMOTE con un valor del hiperparámetro $C = 4$ en una de las iteraciones de la validación cruzada.

100 % de recall y detectar así los 8 casos positivos, esto provocaría una mayor cantidad de falsas alarmas y disminuiría la precisión. Esta configuración ofrece la mayor puntuación f_4, dando así una respuesta a este compromiso entre recall y precisión sin desequilibrar demasiado la balanza hacia uno de los lados.

Finalmente, el modelo de regresión logística con $C = 4$ y aplicando el método SMOTE previamente para compensar el desequilibrio obtiene los siguientes resultados para la validación cruzada con $k = 5$:

$$\text{Recall} = 0.76 \pm 0.16 \quad | \quad \text{Precisión} = 0.57 \pm 0.13 \quad | \quad f_4 = 0.74 \pm 0.15 \tag{4.1}$$

4.2.2. Máquinas de vectores soporte (SVM)

En el caso de las SVM es necesario mantener el escalado previo de los datos, como se explica en el fundamento teórico explicado anteriormente. Se han realizado las pruebas para 3 tipos de kernel: lineal, rbf y polinomial.

Kernel lineal

Los resultados en función del tratamiento del desequilibrio se muestran en la Tabla 4.3.

Los ratios entre instancias positivas y negativas conseguidos mediante el sobremuestreo SMOTE y el submuestreo aleatorio vuelven a ser 1 y 0.01, respectivamente. Además, $C = 1$ por defecto.

En esta ocasión, el mejor resultado viene dado por el modelo sin tratamiento previo para el desequilibrio. De nuevo, se puede buscar el valor del hiperparámetro de regularización C que ofrece una mayor puntuación f_4. Los resultados de esta búsqueda se recogen en la Tabla 4.4.

Tabla 4.3: Resultados de la validación cruzada con $k = 5$ usando el modelo SVM con kernel lineal.

	Desequilibrado	SMOTE	Submuestreo + SMOTE
Recall	0.81 ± 0.10	0.74 ± 0.21	0.74 ± 0.21
Precisión	0.97 ± 0.05	0.61 ± 0.04	0.70 ± 0.17
f_4	0.82 ± 0.10	0.73 ± 0.20	0.73 ± 0.20

Tabla 4.4: Resultado de la validación cruzada usando el SVM con kernel lineal sin tratamiento previo para el desequilibrio para distintos valores del hiperparámetro C.

C	Recall	Precisión	f_4
0.2	0.84 ± 0.10	0.97 ± 0.05	0.85 ± 0.09
0.4	0.84 ± 0.10	0.97 ± 0.05	0.85 ± 0.09
0.6	0.81 ± 0.10	0.96 ± 0.09	0.82 ± 0.09
0.8	0.81 ± 0.10	0.97 ± 0.05	0.82 ± 0.10
1.0	0.81 ± 0.10	0.97 ± 0.05	0.82 ± 0.10

De aquí se deduce que el mejor valor para el hiperparámetro en el modelo SVM con kernel lineal es $C = 0.2$. En la Figura 4.2 se muestra la matriz de confusión resultante sobre el conjunto de validación en una de las iteraciones.

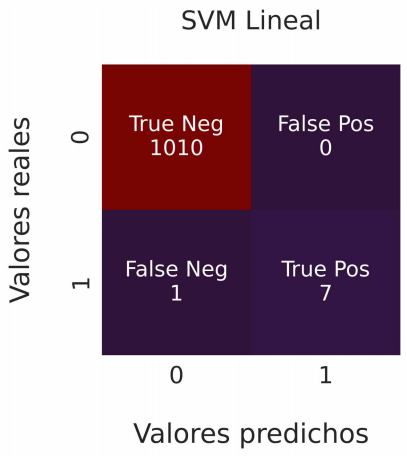


Figura 4.2: Matriz de confusión del modelo SVM con kernel lineal sin tratamiento del desequilibrio con un valor del hiperparámetro $C = 0.2$ en una de las iteraciones de la validación cruzada.

Los resultados del modelo SVM con kernel lineal y $C = 0.2$ en la validación cruzada con $k = 5$ son:

$$\text{Recall} = 0.84 \pm 0.10 \quad | \quad \text{Precisión} = 0.97 \pm 0.05 \quad | \quad f_4 = 0.85 \pm 0.09 \tag{4.2}$$

Kernel rbf

Los resultados de clasificación usando la SVM con kernel rbf y los valores por defecto de los hiperparámetros se muestran en la Tabla 4.5.

Tabla 4.5: Resultados de la validación cruzada con $k = 5$ usando SVM con kernel rbf.

	Desequilibrado[3]	SMOTE	Submuestreo + SMOTE
Recall	0.65 ± 0.12	0.70 ± 0.10	0.67 ± 0.07
Precisión	0.23 ± 0.06	0.45 ± 0.12	0.37 ± 0.09
f_4	0.58 ± 0.10	0.68 ± 0.09	0.64 ± 0.06

El mejor resultado viene dado por el modelo que incluye el método SMOTE. Por tanto, se busca el valor óptimo del hiperparámetro C para dicho modelo (véase la Tabla 4.6).

Tabla 4.6: Resultado de la validación cruzada usando el SVM con kernel rbf y SMOTE para distintos valores del hiperparámetro C.

C	Recall	Precisión	f_4
0.4	0.67 ± 0.07	0.26 ± 0.04	0.61 ± 0.05
0.6	0.70 ± 0.10	0.34 ± 0.07	0.66 ± 0.09
0.8	0.70 ± 0.10	0.39 ± 0.09	0.67 ± 0.09
1.0	0.70 ± 0.10	0.45 ± 0.12	0.68 ± 0.09
1.2	0.65 ± 0.16	0.49 ± 0.13	0.64 ± 0.16

El modelo con mejor puntuación f_4 es, de hecho, el que utiliza $C = 1$. La matriz de confusión para una de las iteraciones de la validación cruzada se muestra en la Figura 4.3.

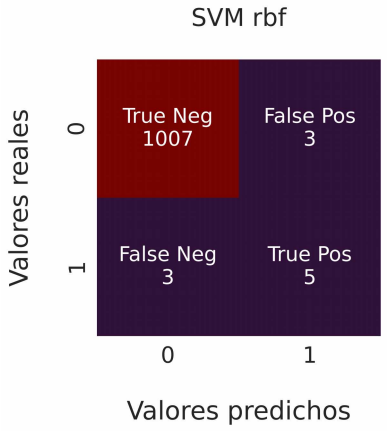


Figura 4.3: Matriz de confusión del modelo SVM con kernel rbf aplicando SMOTE y con un valor del hiperparámetro $C = 1$ en una de las iteraciones de la validación cruzada.

[3]Es necesario fijar el parámetro `class_weight`="balanced" de la función `SVC` de *Sklearn* tanto para el kernel rbf como para el polinomial.

Los resultados del modelo SVM con kernel rbf y $C = 1$ en la validación cruzada con $k = 5$:

$$\text{Recall} = 0.70 \pm 0.10 \quad | \quad \text{Precisión} = 0.45 \pm 0.12 \quad | \quad f_4 = 0.67 \pm 0.09 \tag{4.3}$$

Kernel polinomial

Los resultados de clasificación usando la SVM con kernel polinomial y valores por defecto de los hiperparámetros se muestran en la Tabla 4.7.

Tabla 4.7: Resultados de la validación cruzada con $k = 5$ usando el modelo SVM con kernel polinomial.

	Desequilibrado[3]	SMOTE	Submuestreo + SMOTE
Recall	0.56 ± 0.17	0.78 ± 0.12	0.78 ± 0.12
Precisión	0.32 ± 0.11	0.03 ± 0.01	0.02 ± 0.01
f_4	0.54 ± 0.16	0.28 ± 0.08	0.25 ± 0.05

En este caso el mejor valor de la puntuación f_4 viene dado por el modelo sin tratamiento del desequilibrio. En la Tabla 4.8 se muestran los resultados obtenidos usando distintos valores del hiperparámetro C.

Tabla 4.8: Resultado de la validación cruzada usando el SVM con kernel polinomial sin tratamiento previo para el desequilibrio para distintos valores del hiperparámetro C.

C	Recall	Precisión	f_4
4	0.62 ± 0.11	0.49 ± 0.10	0.61 ± 0.11
5	0.67 ± 0.15	0.53 ± 0.14	0.66 ± 0.15
6	0.67 ± 0.15	0.53 ± 0.18	0.65 ± 0.15
7	0.67 ± 0.15	0.52 ± 0.19	0.65 ± 0.15
8	0.67 ± 0.15	0.54 ± 0.25	0.65 ± 0.15

En esta ocasión, la mejor puntuación f_4 se obtiene con $C = 5$. La matriz de confusión para una de las iteraciones de la validación cruzada se muestra en la Figura 4.4.

Los resultados del modelo SVM con kernel polinomial y $C = 5$ en la validación cruzada con $k = 5$ son:

$$\text{Recall} = 0.67 \pm 0.15 \quad | \quad \text{Precisión} = 0.53 \pm 0.14 \quad | \quad f_4 = 0.66 \pm 0.15 \tag{4.4}$$

En esta sección se ha podido comprobar que los mejores resultados se obtienen para el caso del kernel lineal. Además, para dicho modelo no era útil realizar un tratamiento previo del desequilibrio. Esto indica que los datos de entrenamiento son separables linealmente en un inicio y añadir nuevas instancias rompe dicha propiedad, dando lugar a resultados peores.

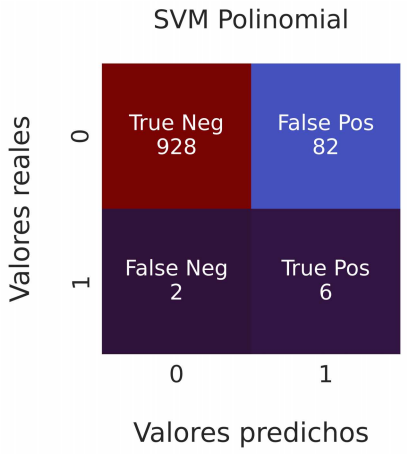


Figura 4.4: Matriz de confusión del modelo SVM con kernel polinomial sin tratamiento del desequilibrio y $C = 5$ en una de las iteraciones de la validación cruzada.

4.2.3. Árboles de decisión

Para el caso de los árboles de decisión los parámetros por defecto a utilizar vuelven a ser 1 y 0.01 para el sobresampleo y subsampleo, respectivamente. Además, se usará la entropía como criterio para medir la pureza. Los resultados bajo esta configuración se muestran en la Tabla 4.9.

Tabla 4.9: Resultados de la validación cruzada con $k = 5$ usando un árbol de decisión.

	Desequilibrado	SMOTE	Submuestreo + SMOTE
Recall	0.84 ± 0.10	0.84 ± 0.10	0.84 ± 0.10
Precisión	0.85 ± 0.12	0.79 ± 0.15	0.70 ± 0.12
f_4	0.84 ± 0.09	0.83 ± 0.09	0.83 ± 0.09

El mejor resultado se obtiene para el modelo sin tratamiento del desequilibrio. Por tanto, se buscan los valores óptimos de los hiperparámetros para este modelo. En esta ocasión, se pueden variar, por ejemplo, `max_leaf_nodes` o `max_features`. Los resultados para cada caso se recogen en las Tablas 4.10 y 4.11, respectivamente.

Tabla 4.10: Resultado de la validación cruzada usando un árbol de decisión sin tratamiento previo del desequilibrio para distintos valores del hiperparámetro `max_leaf_nodes`.

`max_leaf_nodes`	Recall	Precisión	f_4
2	0.84 ± 0.10	1.00 ± 0.00	0.85 ± 0.10
3	0.84 ± 0.10	1.00 ± 0.00	0.85 ± 0.10
4	0.84 ± 0.10	1.00 ± 0.00	0.85 ± 0.10
5	0.84 ± 0.10	0.97 ± 0.06	0.85 ± 0.10

En esta ocasión, fijar únicamente el valor de `max_features` da lugar a mejores resultados que fijando `max_leaf_nodes`. Por tanto, se utiliza sólo el valor de `max_features` $= 25$.

Tabla 4.11: Resultado de la validación cruzada usando un árbol de decisión sin tratamiento previo del desequilibrio para distintos valores del hiperparámetro `max_features`.

`max_features`	Recall	Precisión	f_4
20	0.84 ± 0.10	0.77 ± 0.15	0.83 ± 0.09
25	0.90 ± 0.09	0.97 ± 0.05	0.90 ± 0.09
30	0.84 ± 0.10	0.76 ± 0.14	0.83 ± 0.09
35	0.84 ± 0.10	0.92 ± 0.07	0.84 ± 0.09

La matriz de confusión en una de las iteraciones de la validación cruzada se muestra en la Figura 4.5.

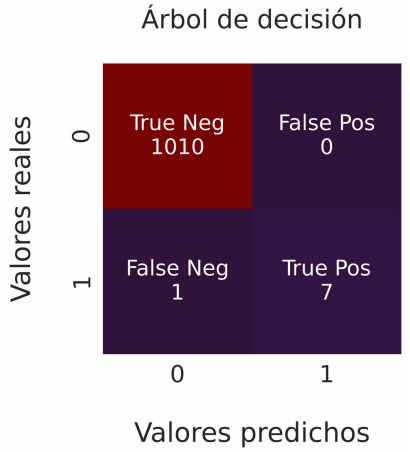


Figura 4.5: Matriz de confusión de un árbol de decisión sin tratamiento del desequilibrio y `max_features` = 25 en una de las iteraciones de la validación cruzada.

Se observa que el árbol de decisión ofrece unos resultados lo suficientemente buenos con valores bajos del parámetro `max_leaf_nodes`. Esto indica que la tarea de clasificación es en su mayor parte sencilla y requiere la evaluación de un bajo número de características para obtener una eficacia razonable. Sin embargo, se ha encontrado también que para el valor de `max_features` = 25 las tres métricas alcanzan su máximo valor, obteniendo así un diseño muy eficaz del árbol de decisión en el entrenamiento. Los resultados en la validación cruzada con $k = 5$ del árbol de decisión con `max_features` = 25 son:

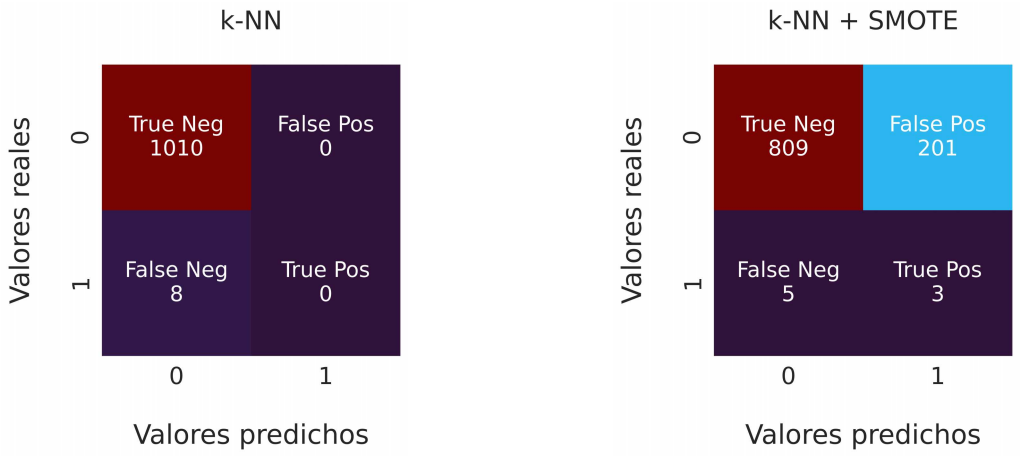
$$\text{Recall} = 0.90 \pm 0.09 \quad | \quad \text{Precisión} = 0.97 \pm 0.05 \quad | \quad f_4 = 0.90 \pm 0.09 \tag{4.5}$$

4.2.4. *k* vecinos más cercanos

Los resultados para el modelo k-NN con los parámetros por defecto se muestran en la Tabla 4.12. Dichos resultados son bastante malos. De hecho, en el caso sin tratamiento del desequilibrio, el modelo de k vecinos más cercanos no es capaz de detectar ni una instancia positiva, dando lugar a puntuaciones nulas. Se puede ver esto en su matriz de confusión (véase la Figura 4.6). Por tanto, ante el resultado de este modelo en su entrenamiento, se descartará el uso de modelos k-NN en la búsqueda del mejor sistema para esta tarea de clasificación.

Tabla 4.12: Resultados de la validación cruzada con $k = 5$ usando el método de los k vecinos más cercanos.

	Desequilibrado	SMOTE	Submuestreo + SMOTE
Recall	0.00 ± 0.00	0.39 ± 0.19	0.36 ± 0.20
Precisión	0.00 ± 0.00	0.01 ± 0.01	0.01 ± 0.01
f_4	0.00 ± 0.00	0.14 ± 0.07	0.13 ± 0.07



(a) Sin tratamiento previo del desequilibrio. **(b)** Aplicando SMOTE.

Figura 4.6: Matrices de confusion del método k-NN en una iteración de la validación cruzada.

4.2.5. *Random forests*

En la Tabla 4.13 se muestran los resultados de clasificación obtenidos para un *random forest* con el criterio de pureza por defecto, "gini", para los distintos tratamientos del desequilibrio. Se han utilizado 100 árboles de decisión.

Tabla 4.13: Resultados de la validación cruzada con $k = 5$ usando un *random forest*.

	Desequilibrado	SMOTE	Submuestreo + SMOTE
Recall	0.81 ± 0.11	0.72 ± 0.21	0.75 ± 0.19
Precisión	1.00 ± 0.00	1.00 ± 0.00	1.00 ± 0.00
f_4	0.82 ± 0.11	0.73 ± 0.20	0.76 ± 0.19

Para valores mayores que 5 de los hiperparámetros `max_leaf_nodes` y `max_features` no existe ningún cambio apreciable en las puntuaciones de las métricas. Por tanto, se considerará únicamente `max_features`= 25, igual que en el árbol de decisión simple. La matriz de confusión para una de las iteraciones se muestra en la Figura 4.7. Los resultados en la validación cruzada con $k = 5$ del *random forest* con `max_features` = 25 son:

$$\text{Recall} = 0.84 \pm 0.10 \quad | \quad \text{Precisión} = 1.0 \pm 0.0 \quad | \quad f_4 = 0.85 \pm 0.10 \tag{4.6}$$

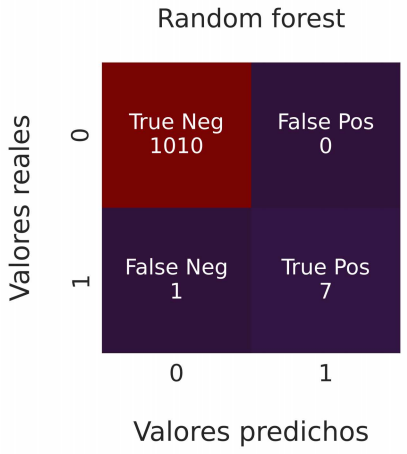


Figura 4.7: Matriz de confusión del modelo *random forest* con `max_features` = 25 sin tratamiento del desequilibrio en una de las iteraciones de la validación cruzada.

4.2.6. Combinación de modelos por votación

Tras ver los resultados de las técnicas de clasificación anteriores, se llevará a cabo una combinación de los mejores modelos. Los modelos elegidos para dicha combinación son:

- Regresión logística con escalado de los datos, tratamiento SMOTE para el desequilibrio y $C = 4$.
- SVM con kernel lineal, escalado de los datos, sin tratamiento del desequilibrio y $C = 0.2$.
- Árbol de decisión sin tratamiento del desequilibrio , entropía como criterio de impureza y `max_features` = 25.
- *Random forest* sin tratamiento del desequilibrio, criterio de impureza "gini", 100 árboles de decisión y `max_features` = 25.

Hard voting

En la Figura 4.8a se muestra la matriz de confusión en una de las iteraciones de la validación cruzada al aplicar la combinación "hard voting".

Los resultados en la validación cruzada con $k = 5$ de dicha combinación son:

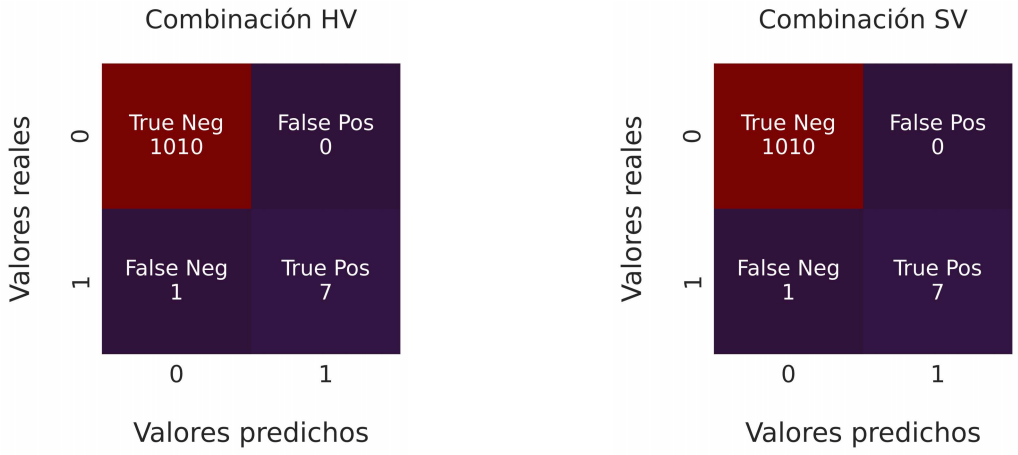
$$\text{Recall} = 0.84 \pm 0.10 \quad | \quad \text{Precisión} = 1.0 \pm 0.0 \quad | \quad f_4 = 0.85 \pm 0.10 \tag{4.7}$$

Soft voting

En la Figura 4.8b se muestra la matriz de confusión obtenida mediante "soft voting" en una de las iteraciones de la validación cruzada con unos pesos para las votaciones $[1, 8, 5, 5]$ (obtenidos tras una serie de pruebas preliminares).

Los resultados en la validación cruzada con $k = 5$ de dicha combinación son:

$$\text{Recall} = 0.84 \pm 0.10 \quad | \quad \text{Precisión} = 1.0 \pm 0.0 \quad | \quad f_4 = 0.85 \pm 0.10 \tag{4.8}$$



(a) "Hard voting". **(b)** "Soft voting".

Figura 4.8: Matrices de confusión de dos modelos de combinación en una de las iteraciones de la validación cruzada.

En general, todas las puntuaciones presentan errores del orden de $0.1 \sim 0.2$. Dicha magnitud de los errores se debe a que en cada partición de la validación cruzada hay pocos casos positivos (al igual que en el conjunto total) y la correcta predicción de una instancia más o una instancia menos altera significativamente el resultado de las puntuaciones en las diferentes iteraciones.

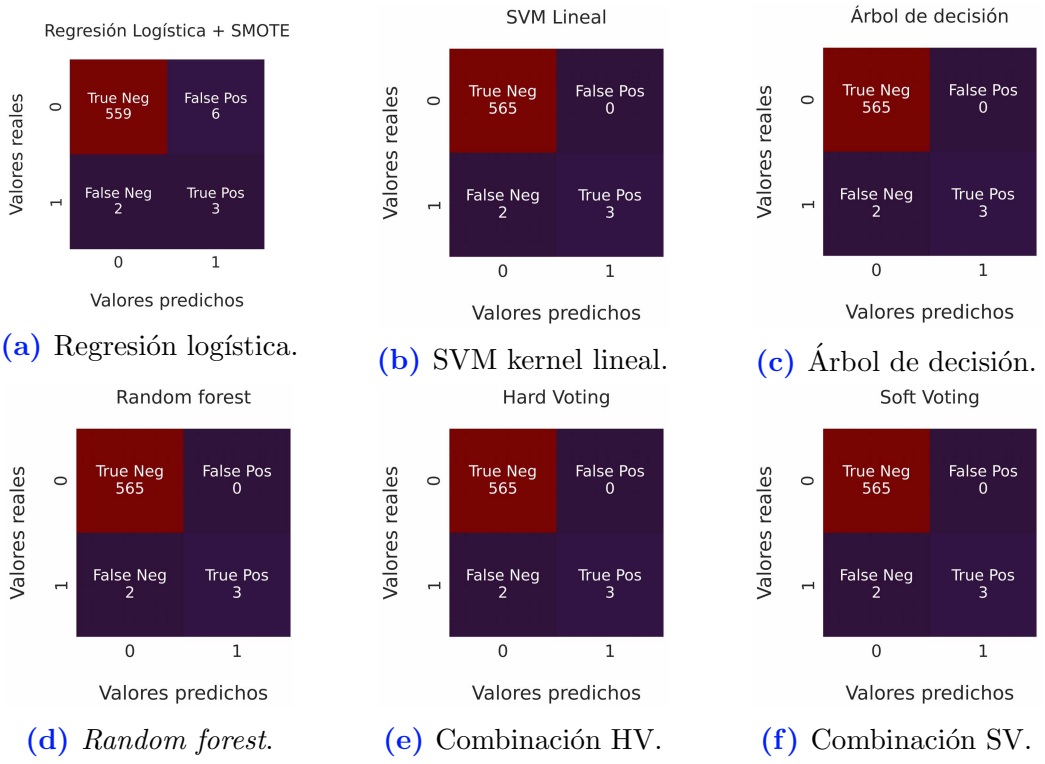
4.3. Resultados sobre el conjunto de test

Una vez probados y diseñados los modelos sobre el conjunto de entrenamiento, se procede a evaluar dichos modelos sobre el conjunto de test. El conjunto de test está formado por 570 nuevas instancias, de las cuales 5 son positivas. Los resultados obtenidos se muestran en la Tabla 4.14. En dicha tabla se recogen de izquierda a derecha los resultados para la regresión logística, la SVM lineal, el árbol de decisión, el *random forest* y los modelo de combinación con "hard voting" y "soft voting".

Tabla 4.14: Resultados de los modelos sobre el conjunto de test.

	RL	SVM lineal	AD	RF	HV	SV
Precisión	0.333	1.000	1.000	1.000	1.000	1.000
Recall	0.600	0.600	0.600	0.600	0.600	0.600
f_4	0.573	0.614	0.614	0.614	0.614	0.614
PR AUC	0.604	0.604	0.802	0.906	0.819	0.837

Las matrices de confusión correspondientes a cada uno de los modelos se muestran en la Figura 4.9.



(a) Regresión logística. **(b)** SVM kernel lineal. **(c)** Árbol de decisión.

(d) *Random forest*. **(e)** Combinación HV. **(f)** Combinación SV.

Figura 4.9: Matrices de confusión de los diferentes modelos sobre el conjunto de test.

Estos resultados demuestran que el número de instancias positivas es demasiado pequeño como para establecer cuál de los modelos está funcionando mejor. Por otra parte, se puede observar que existen dos instancias positivas más difíciles de detectar que el resto. Esto da lugar a los mismos resultados en el recall ya que todos los modelos predicen correctamente tres de las cinco instancias positivas.

A pesar de ello, si hubiese que elegir el mejor clasificador, el modelo *random forest* es el que presenta una mayor puntuación de PR AUC.

Pero, como ya se ha comentado, todos los modelos obtienen predicciones similares ya que hay muy pocas instancias positivas.

Una forma alternativa de medir la bondad de los modelos consistiría en usar, no decisiones de clasificación, sino probabilidades (o más en general, puntuaciones) de pertenencia a la clase positiva: serían mejores aquellos modelos que dieran puntuaciones más altas a las instancias positivas y más bajas a las negativas. La métrica de rendimiento se calcularía directamente sobre las puntuaciones, no sobre las decisiones.

5. Conclusiones y líneas futuras

En este trabajo se ha llevado a cabo un análisis del fundamento teórico de las principales técnicas de aprendizaje supervisado en tareas de clasificación para luego poner dichos métodos en práctica ante un caso específico. La tarea elegida ha sido la detección de exoplanetas. Dicha tarea no es sencilla y el tamaño del conjunto de datos disponible no era demasiado grande, teniendo que dejar de lado métodos como las redes neuronales, que requieren una cantidad de datos mucho mayor. Sin embargo, se han puesto a prueba los métodos de regresión logística, máquinas de vectores soporte, árboles de decisión, k vecinos más cercanos y varios métodos de combinación como son los *random forest* y los clasificadores de votación.

Además, dicha tarea de clasificación presentaba un gran desequilibrio de los datos, ya que de todos los casos analizados muy pocos llegan a ser efectivamente positivos. Por tanto, se han utilizado diversos tratamientos previos de los datos para combatir esta situación. En primer lugar, se ha utilizado el método SMOTE para generar nuevas instancias positivas, pero se han obtenido resultados muy similares al caso en el que no se trata el desequilibrio. Además, se ha probado con una combinación de submuestreo aleatorio y SMOTE sin demasiado éxito. Esto indica que, a pesar del desequilibrio, cuantos más datos se utilicen en el entrenamiento mejor serán los resultados, de forma que no es necesario eliminar el exceso de muestras negativas.

Se ha analizado también la eficacia de cada uno de los modelos diseñados mediante algunas de las métricas más importantes: la precisión, el recall o la puntuación f_4. En este caso, se ha dado mayor importancia a la puntuación f_4 porque en esta el recall tiene cuatro veces más importancia que la precisión. De esta forma, se aseguraba un alto recall, para evitar falsos negativos, sin olvidar completamente la precisión, que influye en la cantidad de falsos positivos. Gracias a estas puntuaciones, se ha demostrado que, a pesar de su simplicidad, los árboles de decisión son muy eficaces y pueden obtener buenas predicciones. Así, también se han obtenido buenos resultados con el *random forest* y los modelos de combinación.

De cara al futuro, lo más importante, es la obtención de nuevos datos para formar un conjunto mayor. De esta forma, si se llegan a alcanzar las decenas de miles de instancias, será posible utilizar redes neuronales u otros modelos más complejos que los utilizados en este trabajo. Además, con un conjunto más grande de datos se podrían diseñar modelos de combinación entrenando cada uno de los clasificadores individuales secuencialmente (*boosting*). Otra línea de trabajo futuro importante es el uso de métricas basadas en puntuaciones en vez de en decisiones, por ejemplo el *Likelihood Ratio*.

En conclusión, las tareas de clasificación dependen en su mayor parte del conjunto de datos. No se puede obtener un sistema clasificador eficaz sin una base de datos representativa, sea cual sea la complejidad de su diseño. Es posible siempre diseñar un modelo más complejo, pero el objetivo era conseguir un clasificador capaz de reducir el número de posibles casos positivos de exoplanetas sin una excesiva cantidad de falsas alarmas. Los modelos han sido diseñados con dicha intención a pesar de que, debido a las limitaciones del conjunto de test, no se haya comprobado completamente su eficacia.

Bibliografía

[1] "Exoplanet hunting in deep space: Kepler labelled time series data," Último acceso: 03/06/22. [Online]. Available: https://www.kaggle.com/keplersmachines/kepler-labelled-time-series-data

[2] "How to find an extrasolar planet," Último acceso: 03/06/22. [Online]. Available: https://www.esa.int/Science_Exploration/Space_Science/How_to_find_an_extrasolar_planet

[3] A. Géron, *Hands-on machine learning with Scikit-Learn, Keras and TensorFlow : Concepts, tools, and techniques to build intelligent systems.* O'Reilly Media, 2019.

[4] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press, 2014.

[5] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, p. 321–357, Jun 2002. [Online]. Available: http://dx.doi.org/10.1613/jair.953

[6] J. Czakon, "24 evaluation metrics for binary classification (and when to use them)," Último acceso: 03/06/22. [Online]. Available: https://neptune.ai/blog/evaluation-metrics-binary-classification

[7] R. Draelos, "Measuring performance," 2 2020, Último acceso: 03/06/22. [Online]. Available: https://glassboxmedicine.com/2019/02/23/measuring-performance-auc-auroc/

[8] T. Saito and M. Rehmsmeier, "The precision-recall plot is more informative than the ROC plot when evaluating binary classifiers on imbalanced datasets," *PloS One*, vol. 10, no. 3, Mar. 2015.