Algoritmos de Aprendizaje Automtico y sus aplicaciones

Trabajo de Fin de Grado

Javier Díaz Bustamante Ussia

2 de septiembre de 2015

Índice

Resumen

En este trabajo vamos a estudiar diferentes algoritmos de Aprendizaje Automático (*Machine Learning*) para la clasificación de sucesos. Para comprobar su funcionamiento, los utilizaremos sobre distintas bases de datos, una de supervivientes del Titanic y otra de búsqueda del bosón de Higgs.

1. Introducción

En un mundo cada vez más tecnológico, las bases de datos crecen cada día. Cuando alguien entra en una página web, realiza una compra en un comercio, una empresa realiza un estudio de mercado, se celebran unas elecciones, se están almacenando datos. Con este ingente flujo de datos, la física no se podía quedar atrás, hoy en día los grandes aceleradores de partículas manejan al día millones de sucesos que hay que catalogar, clasificar y estudiar, pero la enorme cantidad de estos datos hace imposible su tratamiento *manual*.

Para solucionar el problema del tratamiento de datos surge el *Machine Learning* (de ahora en adelante *ML*), con algoritmos cada vez más potentes capaces de sacar el máximo partido a estos datos. En este trabajo estudiaremos en concreto cinco de ellos, todos de clasificación. Estos cinco son *Logistic Regresion* (LR), *Naïve Bayes* (NB), *Support Vector Machine* (SVM), *Random Forest* (RF) y *K-Nearest Neighbors* (KNN), explicados en detalle en la seccion 3.

Existen dos tipos de problemas en *ML*, los de clasificación y los de regresión. Todos ellos tratan de predecir el valor de una variable a partir de un conjunto de datos, la diferencia es que en los de regresión la variable predicha toma valores continuos, mientras que en la clasificación toma valores discretos, pudiendo ser una clasificación binaria (verdadero o falso) o una clasificación multiclase, como por ejemplo un algoritmo de reconocimiento de dígitos.

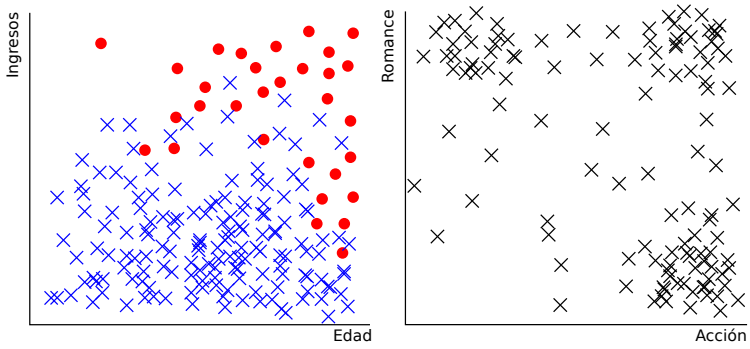
2. Consideraciones previas

2.1. Aprendizaje supervisado y no supervisado

Antes de explicar cada uno de los algoritmos, vamos a ver algunas características comunes de *ML*. Para empezar, veamos la diferencia entre *aprendizaje supervisado* y *aprendizaje no supervisado*. En *ML*, el término aprendizaje (o entrenamiento) se refiere al análisis de los datos por parte del algoritmo. Éste recibe un conjunto de datos de aprendizaje (el *training set*) y los analiza para tomar una decisión. La diferencia entre aprendizaje supervisado y aprendizaje no supervisado es que en el primero el conjunto de datos de aprendizaje se encuentra perfectamente etiquetado, mientras que en el no supervisado no lo está.

Por ejemplo, si quisiéramos entrenar un algoritmo para saber a qué tipo de personas les gusta la música clásica, podríamos crear un conjunto de aprendizaje con entradas como: "A Juan Pérez, de 64 años, de clase alta, casado y con carrera universitaria, residente en Madrid, le gusta la música clásica", y otras entradas del estilo de "A Pedro Gutiérrez, de 24 años, clase media, soltero, sin carrera universitaria, residente en Villalpando, no le gusta". Éste sería un problema de clasificación supervisada, y podemos ver un ejemplo en la gráfica 1a.

Por otra parte, podemos tener datos como por ejemplo el gusto por diferentes tipos de películas de los clientes de un videoclub. A una persona pueden gustarle un 80 % de las películas románticas que ha visto, mientras que sólo un 57 % de las películas de acción. A otra, sin embargo, le gustan un 43 % de las primeras y un 96 % de las segundas. Un ejemplo de training set podría ser el de la gráfica 1b. En ella, aunque no tengamos clasificadas a las personas, se puede comprobar de forma más o menos nítida que el videoclub tiene tres tipos de clientes distintos.

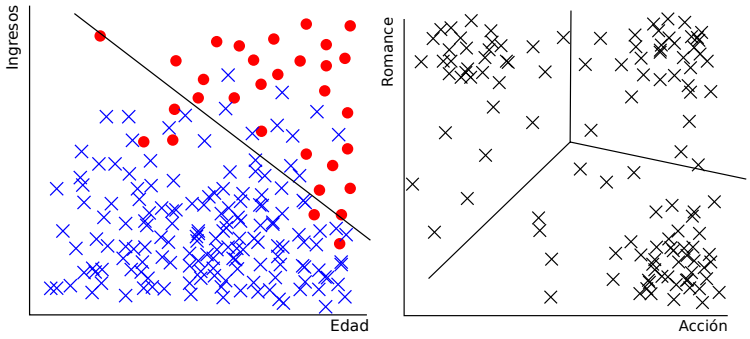


(a) Supervisado. En rojo, los amantes de la música clásica.

(b) No supervisado.

Gráfica 1: Aprendizaje supervisado (a) y no supervisado (b).

Tanto en el aprendizaje supervisado como en el no supervisado, el objetivo de un algoritmo de clasificación es el de elegir una superficie de decisión que separe distintas regiones a las que se asignará cada una de las clases. Estas superficies pueden ser más o menos complejas, dependiendo del algoritmo que se utilice. Un ejemplo de curvas de decisión se recoge en la gráfica 2.



Gráfica 2: Curvas de decisión lineales.

2.2. Métricas

Una vez el algoritmo ha sido entrenado sobre un conjunto de aprendizaje, es hora de probarlo. Para ello se emplea otro conjunto de datos, llamado conjunto de test (*test set*)[1], que se clasifica según el algoritmo disponga. En el caso de clasificación binaria, el algoritmo asignará a cada entrada del *test set* una etiqueta de clase (verdadero o falso, 1 ó 0). Nos interesa saber cómo de preciso ha sido el algoritmo, para lo que disponemos de diversas métricas. Según la etiqueta original de cada entrada y según la etiqueta predicha por el algoritmo, se pueden dar los casos de la tabla 1.

[1]Muchas veces se suele tener sólo un conjunto de datos, que se divide en distintas partes, una el *training set*, otra el *test set*, y otra el *cross-validation set*, que veremos más adelante.

	0 real	**1 real**
0 predicho	Verdadero negativo (TN)	Falso negativo (FN)
1 predicho	Falso positivo (FP)	Verdadero positivo (TP)

Tabla 1: Posibles casos.

Un buen algoritmo será aquel que prediga 0 para todos los negativos y 1 para los positivos, pero la realidad es que esto casi nunca ocurre. Necesitamos, por lo tanto, un mecanismo para evaluar la bondad de un algoritmo, y eso se consigue con las métricas definidas a continuación (ver [1]):

$$\begin{aligned} \text{Accuracy} &= \frac{TP+TN}{TP+FP+TN+FN} & \text{Recall} &= \frac{TP}{TP+FN} \\ \text{Specificity} &= \frac{TN}{FP+TN} & \text{Precision} &= \frac{TP}{TP+FP} \\ \text{F1-score} &= \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \end{aligned} \tag{1}$$

Recall (sensibilidad) nos da el cociente entre los casos positivos bien clasificados y todos los casos positivos, mientras que *Specificity* (especificidad) nos da la de los casos negativos. *Precision* (precisión) nos da el cociente de los casos positivos bien clasificados entre todos los casos clasificados como positivos.

Aunque pudiera parecer que *Accuracy* (exactitud) es una buena medida de la bondad de la clasificación, pues nos da la proporción de los casos bien clasificados frente al resto de casos, no siempre es así. En los casos de clases sesgadas (*skewed classes*), en las que la proporción de una de las dos clases es mucho mayor que la de la otra, un algoritmo que clasifique cualquier entrada de datos como la clase mayoritaria tendría una gran exactitud, aunque no cumpliría con el objetivo de clasificar correctamente los datos. Por ejemplo, supongamos un problema de detección de fraude en transacciones bancarias en el que nuestro algoritmo deba clasificar como 1 las operaciones fraudulentas. En la realidad hay muchas más transacciones legítimas que fraudulentas, en nuestro ejemplo consideraremos que son el 99 % de las transacciones. Si tenemos un algoritmo que prediga siempre que una transacción es legítima, la exactitud será del 99 %, pero no por ello será un buen algoritmo.

El valor F1 soluciona este problema. Se trata de una media armónica de la precisión y la sensibilidad. A lo largo de este trabajo consideraremos ésta como la métrica para decidir si un algoritmo es mejor que otro.

Para estudiar un algoritmo seguimos unos pasos definidos. Primero, entrenaremos el algoritmo con el *training set*, tras lo que el algoritmo define una *decisión* que utilizará para clasificar nuevos datos. A continuación, con el algoritmo ya entrenado, procedemos a predecir las clases de cada entrada de datos del *test set*, con lo que podremos calcular nuestras métricas como vimos en las ecuaciones (1). Al final, tenemos un algoritmo entrenado que generaliza su decisión mejor o peor en función de los valores de las métricas.

El concepto de generalizar viene de que normalmente tenemos un conjunto de datos ya clasificados, que usaremos para entrenar el algoritmo, pero este algoritmo lo necesitamos no para clasificar ese conjunto, sino para clasificar nuevos conjuntos de datos que estén aún sin clasificar. Por esto no medimos las métricas sobre el mismo conjunto con el que entrenamos, sino que lo hacemos sobre un

conjunto distinto, para ver si el algoritmo no sólo clasifica bien el conjunto de aprendizaje, sino cualquier nuevo conjunto de datos del que dispongamos.

2.3. Validación Cruzada

La mayoría de los algoritmos de *ML* dependen de parámetros que hay que fijar, optimizando las métricas conseguidas por el algoritmo.Hasta ahora el procedimiento a seguir era entrenar el algoritmo con el conjunto de aprendizaje, y ver la bondad del mismo con el conjunto de test. Ahora, para fijar correctamente estos parámetros, usaremos un tercer conjunto, el conjunto de validación cruzada (*Cross Validation set*. Los pasos a seguir ahora son los siguientes:

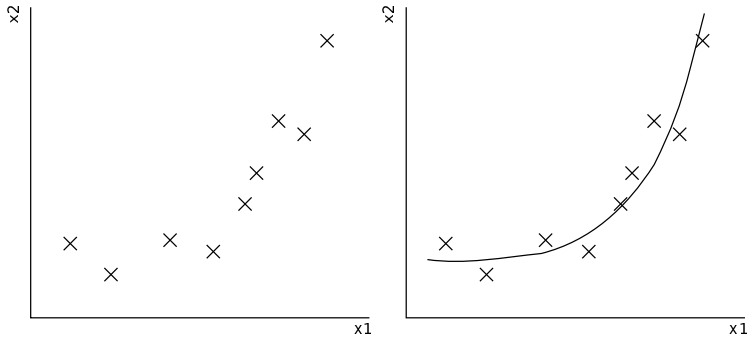
Primero, elegimos valores para nuestros parámetros y entrenamos el algoritmo. Evaluamos las métricas sobre el conjunto de validación cruzada (de ahora en adelante CVC), y volvemos a realizar estos pasos con valores distintos de los parámetros, eligiendo el conjunto de parámetros que mejor prediga las clases del CVC. Por último, con el algoritmo entrenado con los mejores parámetros, evaluamos las métricas sobre el conjunto de test. Estas métricas son las que nos dirán cómo de acertado será nuestro algoritmo al clasificar datos nuevos.

Al igual que antes, el proceso de evaluar métricas se realiza en conjuntos de datos que no hayamos utilizado para entrenar el algoritmo, aunque esta vez tampoco usamos el conjunto que nos ha servido para ajustar los parámetros. Ésto se debe a que, al usar el CVC para ajustar los parámetros, éste ya no nos dice cómo generaliza nuestro algoritmo a datos nuevos, pues el algoritmo final depende de la información que aporta el CVC. Para ver cómo generaliza, tenemos que alimentar nuestro algoritmo con datos que todavía no se hayan usado.

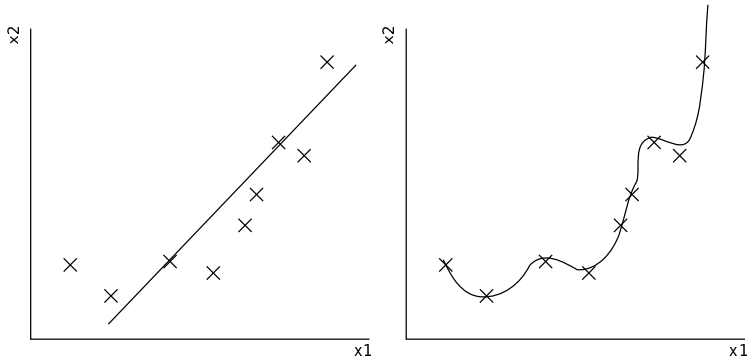
2.4. Varianza, Sesgo y Regularización

Al entrenar un algoritmo nos enfrentamos a dos errores diametralmente opuestos, el *sesgo* y la *varianza*. Tendremos alto sesgo (*high bias*) cuando nuestra hipótesis es demasiado sencilla o tengamos pocas variables de entrada. La consecuencia es que nuestro algoritmo no generaliza bien los datos nuevos porque tampoco ajusta bien los datos del training set, no somos capaces de extraer información relevante de los datos que tenemos, estamos cometiendo un *underfitting* (se podría traducir como soajuste). Para solucionarlo, lo que podemos hacer es tomar datos nuevos con más variables, crear variables derivadas de las que ya tengamos, hacer una hipótesis más compleja o modificar el parámetro de *smoothing*, que veremos en seguida.

Por el contrario, tendremos alta varianza (*high variance*) en nuestro algoritmo cuando nuestra hipótesis es demasiado compleja y no tenemos suficientes datos. Nuestro algoritmo no generalizará bien para datos nuevos porque se ciñe demasiado a los datos del training set, perdiendo la tendencia general de estos datos al darle más importancia a las variaciones específicas de este conjunto. Cometeremos un muy bajo error en los datos del training set, pagando un alto error en el test set, estamos produciendo un sobreajuste (*overfitting*). Para solucionarlo podremos tomar más datos, simplificar la hipótesis o modificar el parámetro de *smoothing*. Podemos ver un claro ejemplo de ambos problemas en la gráfica 3.



(a) Datos (b) Bien ajustado



(c) Alto sesgo (underfitting) (d) Alta varianza (overfitting)

Gráfica 3: Datos para la regresión (a), regresión correcta (b), regresión con alto sesgo (c) y regresión con alta varianza (d).

A continuación explicamos en qué consiste el parámetro de *smoothing*. Gran parte de los algoritmos se basan en una hipótesis en la que se asigna unos pesos a cada variable (*feature weights*). El algoritmo optimizará estos pesos para que la predicción realizada por la hipótesis coincida lo máximo posible con las etiquetas de cada entrada de datos. Estos pesos, por lo tanto, serán mayores cuanto más influya la variable en la clasificación. El parámetro de *smoothing* se encarga de controlar la importancia que se da a las variables, con el objetivo de no depender demasiado de ellas, evitando caer en problemas de alta varianza. Una formulación más precisa sería la siguiente:

Sea un algoritmo de clasificación en el que a cada variable se le asocia un peso θ_j, y en el que cada entrada de datos viene dada por el vector $\boldsymbol{x}^{(i)}$, donde $x_j^{(i)}$ sería el valor de la variable j de la entrada i. Cada entrada está clasificada con una etiqueta $y^{(i)}$. El algoritmo tratará de seleccionar el vector de pesos $\boldsymbol{\theta}$ que optimice un error cometido $J(\boldsymbol{\theta})$ (la fórmula concreta de este error dependerá del algoritmo en cuestión). El parámetro de *smoothing*[2] C se introduce como un término adicional al error que se minimiza. Este término es proporcional al inverso de C y a la suma cuadrática de los θ_j:

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{1}{C}\boldsymbol{\theta} \cdot \boldsymbol{\theta} \tag{2}$$

Reduciendo C se consigue minimizar el valor de los pesos θ_j, suavizando la hipótesis del algoritmo, reduciendo así la varianza del algoritmo. Por otro lado, si aumentamos C permitimos al algoritmo ajustar los pesos más de acuerdo con su hipótesis, permitiendo que ésta sea más compleja y reduciendo el sesgo del mismo. Por lo tanto, el parámetro de *smoothing* nos ayuda a mejorar nuestro algoritmo, y lo podremos fijar con el CVC, como vimos en el apartado 2.3.

3. Algoritmos

En esta sección estudiaremos diversos algoritmos de clasificación. La mayor parte de los algoritmos siempre producen una hipótesis y una función de error a minimizar. La hipótesis es una aplicación que va del espacio de las variables (*features*) al de las etiquetas (*labels* o *targets*). La forma concreta de la hipótesis depende del algoritmo. La función de error, o función objetivo, mide el error cometido por la hipótesis, y será una función que vaya del espacio de los vectores de pesos al de números reales positivos, incluyendo el 0. La forma concreta de la función objetivo también depende del algoritmo, así que vamos a estudiar los que utilizaremos.

3.1. Logistic Regression

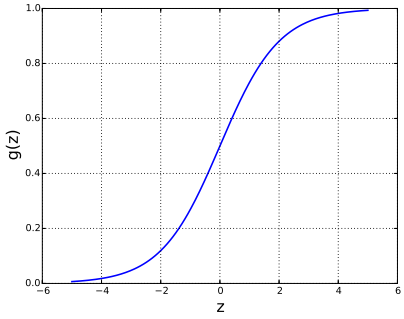
El algoritmo de Regresión Logística[3] es un algoritmo de modelo lineal (ver [2], capítulo 1). Siempre producirá una superficie de decisión que dependa linealmente de las variables de entrada, aunque si queremos una superficie más compleja sólo tendremos que crear nuevas variables no lineales con las de entrada.

[2] A veces se utiliza el parámetro de regularización $\lambda \propto C^{-1}$.

[3] El término "regresión" se debe a razones históricas, el algoritmo es de clasificación.

Se llama de Regresión Logística porque precisamente para la hipótesis hace uso de una función logística, (3). Ésta es una función sigmoidea, muy comunes en estadística por tener la típica forma de densidad de probabilidad acumulada, como podemos ver en la gráfica 4.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{3}$$



Gráfica 4: Función logística. La forma de "S" (sigmoidea) es típica de las densidades de probabilidad acumulada en estadística.

Sea $\boldsymbol{x}$ el vector de variables de entrada, y $\boldsymbol{\theta}$ el vector de pesos, ambos de la misma dimensión. Por convenio, al vector $\boldsymbol{x}$ le añadimos una primera componente, $x_0 = 1$, que tiene su peso correspondiente, θ_0. Esta componente añade un término constante a la combinación lineal de las variables de entrada[4]. La hipótesis del algoritmo es (4), donde $g(z)$ es la función logística de (3). La hipótesis se interpreta como la probabilidad de que, dados el vector $\boldsymbol{x}$ y los pesos $\boldsymbol{\theta}$, la etiqueta correspondiente a esos datos sea "1".

$$\boxed{h_{\boldsymbol{\theta}}(\boldsymbol{x}) = p(y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = g(\boldsymbol{\theta} \cdot \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta} \cdot \boldsymbol{x}}}} \tag{4}$$

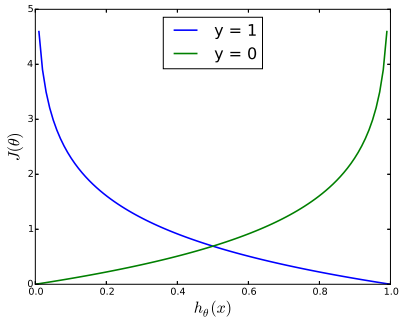
Por norma general, predeciremos "$y = 1$" si $h_{\boldsymbol{\theta}}(\boldsymbol{x}) > 0.5$, es decir (ver gráfica 4), si $\boldsymbol{\theta} \cdot \boldsymbol{x} > 0$, por lo que nuestra superficie de decisión, lineal en las variables $\boldsymbol{x}$, será $\boldsymbol{\theta} \cdot \boldsymbol{x} = 0$.

La función de coste nos da el error cometido por la hipótesis, y en el caso de la Regresión Logística tiene la siguiente forma:

$$\boxed{J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) + \left(1 - y^{(i)} \right) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) \right]} \tag{5}$$

donde m es el número de entradas de datos. Nótese que el primer término del sumatorio corresponde a los datos con $y = 1$, mientras que el segundo es el de los datos con $y = 0$. La función de coste se representa para los casos $y = 1$ y $y = 0$ en la gráfica 5.

[4]La componente θ_0 no se suele penalizar al hacer la regularización (2), cambiando $\boldsymbol{\theta} \cdot \boldsymbol{\theta}$ por $\sum_{j=1}^{n} \theta_j^2$.



Gráfica 5: Función de error frente a $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ para los casos $y = 1$ y $y = 0$.

3.2. Naïve Bayes

El clasificador Bayes Ingenuo es probablemente uno de los más simples que existen(ver [3] capítulo 20 y [4]). Se basa en asumir ingenuamente que las variables de las entradas de datos de una clase determinada son independientes. Esta asunción se expresa con la siguiente expresión

$$p(\boldsymbol{x}|C) = \prod_{j=1}^{n} p(x_j|C) \tag{6}$$

donde $\boldsymbol{x}$ es el vector de variables, cuyas componentes son x_j, y C es una determinada clase (en nuestros problemas de clasificación binaria, C será 0 ó 1).

Gracias al teorema de Bayes, sabemos que la probabilidad de que dado el vector de variables $\boldsymbol{x}$, éste sea de la clase C_l es:

$$\boxed{p(C_l|\boldsymbol{x}) = \frac{p(C_l)p(\boldsymbol{x}|C_l)}{p(\boldsymbol{x})}} \tag{7}$$

En general, para un problema de clasificación multiclase, siendo C_l cada una de las clases, el algoritmo asignará a cada entrada nueva de datos la clase C_k tal que

$$k = \max_l p(C_l|\boldsymbol{x})$$

Ésta es la hipótesis del algoritmo. Nótese que, dado un vector $\boldsymbol{x}$, en (7), $p(\boldsymbol{x})$ es una constante, por lo que sólo tenemos que preocuparnos por maximizar el numerador, teniendo en cuenta (6).

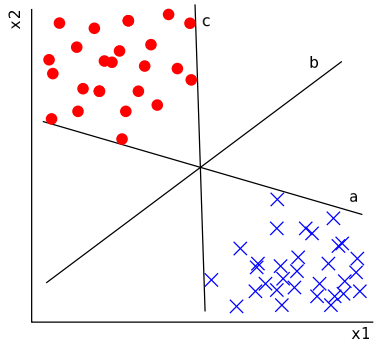
Para calcular las probabilidades $p(x_j|C)$, hay que tener en cuenta el tipo de distribución que sigue la variable j, dando lugar a diferentes algoritmos, como por ejemplo, el "Naïve Bayes Gaussiano", el "Naïve Bayes Multinomial" o el "Naïve Bayes Binomial", todos ellos nombrados según la distribución de probabilidad de las variables. También se pueden hacer algoritmos de Naïve Bayes mixtos, donde a distintas variables les correspondan distintas distribuciones de probabilidad.

Las superficies de decisión entre dos clases C_i y C_j serán las que cumplan (8).

$$\{\boldsymbol{x} : p(C_i|\boldsymbol{x}) = p(C_j|\boldsymbol{x}) > p(C_k|\boldsymbol{x}) \quad \forall k \neq i, j\} \tag{8}$$

3.3. Support Vector Machine

Las Máquinas de Soporte Vectorial (*SVM* por sus siglas en inglés) son otro algoritmo de aprendizaje supervisado de *ML*[5]. Muchas veces son llamadas también *large margin separator* o *maximum margin separator*, porque la superficie de decisión que toman es la que más separa los datos del conjunto de aprendizaje. Como vemos en la gráfica 6, todas las líneas separan a la perfección los datos, pero intuitivamente esperamos que la que mejor los separe sea la curva b. ¿Por qué esperamos esto? La respuesta es muy sencilla, esperamos que al tomar nuevos datos, estos sigan la misma distribución que los antiguos, así que las curvas a y c, que tienen puntos muy cercanos, puede que empiecen a clasificar mal alguna entrada de datos. Sin embargo, la curva b, que está más separada de los datos actuales, seguramente no tenga problema al clasificar datos nuevos. Lo que hace que las *SVM* sean tan potentes es que no tratan de minimizar el error empírico, sino de minimizar el error de generalización que podremos cometer al tomar datos nuevos.



Gráfica 6: Distintas líneas de decisión. La que elegirá una *SVM* será la b, por ser la que deja mayor margen, minimiza el error de generalización.

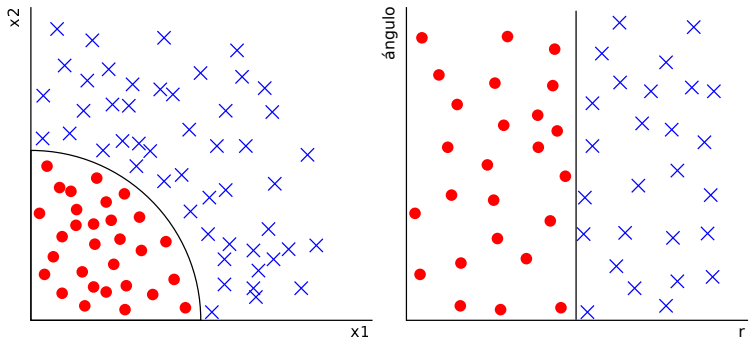
Las *SVM* tienen la característica de crear una curva de decisión lineal siempre en las variables que emplea. Esto no debería suponer un problema, pues utilizando *kernels* (cambios de variables) cambiamos nuestro espacio de variables a otro espacio que no sea lineal en el anterior, consiguiendo en el nuevo espacio superficies de decisión no lineales en el antiguo, como vemos en la gráfica 7, en la que se ha cambiado del espacio formado por x_1 y x_2 al formado por la distancia al origen y el ángulo con el eje x_1.

Para la formulación matemática de las *SVM* es importante notar que las etiquetas tradicionalmente no son $y^{(i)} \in \{0, 1\}$, sino $y^{(i)} \in \{-1, +1\}$. Tampoco añadimos el término independiente como una componente más del vector de pesos y la componente $x_0^{(i)}$ del de variables, sino como un término independiente aparte, quedando la superficie de decisión como (9).

$$\{\boldsymbol{x} : \boldsymbol{\theta} \cdot \boldsymbol{x} + b = 0\} \tag{9}$$

Para conseguir el máximo margen, trataremos de que los puntos más cerca-

[5]Ver [5], [6], capítulo 10 de [7], capítulo 4 de [8] y capítulo 18 de [3].



(a) Variables antiguas (b) Variables nuevas

Gráfica 7: Con un cambio de variables (*kernel*) podemos crear superficies de decisión no lineales.

nos a (9) cumplan que

$$\begin{aligned} \boldsymbol{\theta} \cdot \boldsymbol{x}^{(i)} + b = +1 \text{ si } y^{(i)} = +1 \\ \boldsymbol{\theta} \cdot \boldsymbol{x}^{(i)} + b = -1 \text{ si } y^{(i)} = -1 \end{aligned} \tag{10}$$

por lo que para cualquier punto tendremos que

$$y^{(i)} \left(\boldsymbol{\theta} \cdot \boldsymbol{x}^{(i)} + b \right) \geq 1 \quad \forall i = 1, ..., m \tag{11}$$

Los puntos que cumplen que $y^{(i)} \left(\boldsymbol{\theta} \cdot \boldsymbol{x}^{(i)} + b \right) = 1$ se llaman *vectores de soporte*, y son los que dan el nombre al algoritmo. Nuestra hipótesis será

$$\boxed{h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \text{signo}(\boldsymbol{\theta} \cdot \boldsymbol{x} + b)} \tag{12}$$

Si miramos la gráfica 8a vemos el hiperplano de separación y los hiperplanos soporte. Las distancias de cada uno de ellos al origen, nombrados como en la gráfica, son

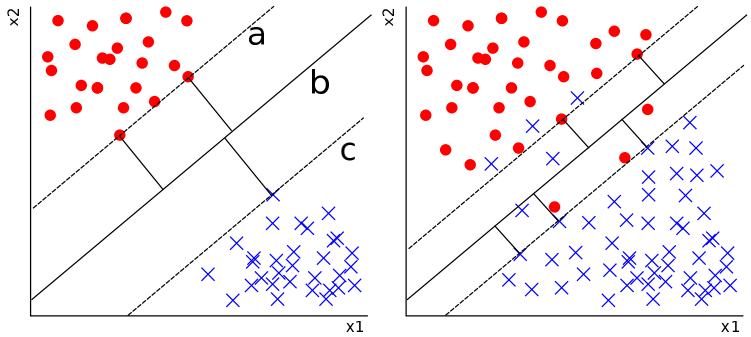
$$\begin{aligned} d_a &= \frac{|b-1|}{||\boldsymbol{\theta}||} \\ d_b &= \frac{|b|}{||\boldsymbol{\theta}||} \\ d_c &= \frac{|b+1|}{||\boldsymbol{\theta}||} \end{aligned} \tag{13}$$

por lo que la distancia entre los hiperplanos soporte, que es la que hay que maximizar para hacer lo propio con el margen, es

$$d = \frac{2}{||\boldsymbol{\theta}||} \tag{14}$$

Finalmente, tenemos que el algoritmo *SVM* se reduce a maximizar (14) sujeto a (11), que es lo mismo que

$$\boxed{\begin{array}{l} \text{Minimizar} \quad ||\boldsymbol{\theta}||^2 \\ \quad \text{sujeto a } y^{(i)} \left(\boldsymbol{\theta} \cdot \boldsymbol{x}^{(i)} + b \right) \geq 1 \quad \forall i = 1, ..., m \end{array}} \tag{15}$$



(a) High margin (b) Soft margin

Gráfica 8: Gráfica **a**: Hiperplano de separación (b), con ecuación $\{\boldsymbol{x} : \boldsymbol{\theta}\cdot\boldsymbol{x}+b = 0\}$ e hiperplanos soporte, (a) con ecuación $\{\boldsymbol{x} : \boldsymbol{\theta}\cdot\boldsymbol{x}+b = +1\}$ y (c) con ecuación $\{\boldsymbol{x} : \boldsymbol{\theta}\cdot\boldsymbol{x}+b = -1\}$. Los vectores de soporte son los más cercanos al hiperplano de separación. Gráfica **b**: Ejemplo de problema de *soft margin*. Hay algunos puntos que se clasifican mal con tal de mantener un margen amplio con la mayoría de los datos, reduciendo el error de generalización.

Como (15) es un problema de programación cuadrática, y este trabajo no intenta ser un tratado de programación no lineal, la manera de resolver este problema de optimización no se explica aquí, recomendando al lector que quiera profundizar en el tema la lectura de [7].

3.3.1. Margen blando

En gran parte de las bases de datos no podremos separar linealmente las variables (gráfica 8b), por lo que el método de *high margin* fallará. Para estos casos se utiliza el método de *soft margin* (margen blando), introduciendo las variables de holgura h_i tales que todos los puntos deben cumplir que

$$y^{(i)}\left(\boldsymbol{\theta}\cdot\boldsymbol{x}^{(i)}+b\right) \geq 1-h_i \quad \forall i = 1,...,m \tag{16}$$

Con esto, el problema de minimización se convierte en (17).

$$\boxed{\begin{array}{l}\text{Minimizar} \qquad ||\boldsymbol{\theta}||^2 + C\sum_{i=1}^{m} h_i \\ \quad \text{sujeto a } y^{(i)}\left(\boldsymbol{\theta}\cdot\boldsymbol{x}^{(i)}+b\right) \geq 1-h_i \quad \forall i = 1,...,m\end{array}} \tag{17}$$

3.3.2. Kernels

A continuación vamos a explicar los *kernels* que veíamos al principio de esta sección. Éstos son aplicaciones no lineales $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R} : (\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) \mapsto K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})$. Gracias a los kernels, podemos transformar nuestro espacio de variables $\mathbb{R}^n$ a uno de dimensión superior $\mathbb{R}^{n'}$ en el que se pueda hacer una separación lineal de los datos.

Los kernels tienen la propiedad de que se puede encontrar una transformación $\phi : \mathbb{R}^n \to \mathbb{R}^{n'} : \boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ tal que $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \phi(\boldsymbol{x}^{(i)}) \cdot \phi(\boldsymbol{x}^{(j)})$. Ésta es la

transformación del espacio de las variables que se lleva a cabo. La superficie de decisión será lineal en el nuevo espacio de variables, y con la transformación inversa de ϕ conseguimos una superficie no lineal en las variables antiguas. La gran ventaja de emplear kernels es que no necesitaremos calcular explícitamente $\phi(\boldsymbol{x})$.

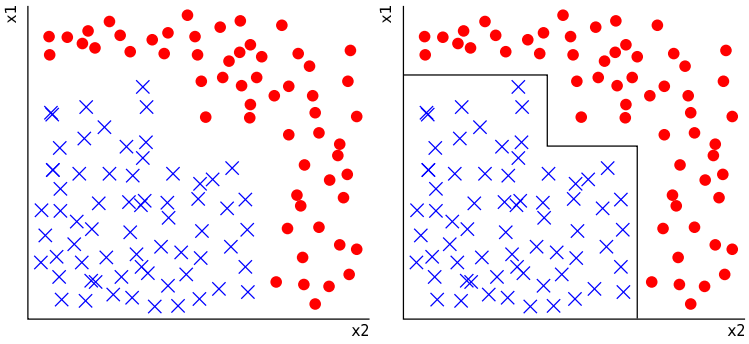
Hay muchos tipos de kernels. Los más comunes son: kernel polinomial homogéneo, $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = (\boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)})^d$, e inhomogéneo, $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = (\boldsymbol{x}^{(i)} \cdot \boldsymbol{x}^{(j)} + 1)^d$; kernel de función de base radial gaussiano, también llamado *RBF kernel*, $K(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)}) = \exp\left(-\frac{||\boldsymbol{x}^{(i)} - \boldsymbol{x}^{(j)}||^2}{2\sigma^2}\right)$; kernel lineal, igual que el polinomial homogéneo con $d = 1$, que corresponde a no hacer nada en el espacio de variables. No hay ninguno mejor que otro, cada base de datos tendrá uno de ellos que funcione mejor que los demás, pero en otras bases de datos ese kernel puede ser peor. Parte del "*cocinado*" al preparar un algoritmo *SVM* para una base de datos será encontrar qué kernel y con qué valores de los parámetros (d y σ, por ejemplo) funcionará mejor.

3.4. Random Forest

Los Random Forest son un ejemplo de los llamados *ensemble methods*, en los que se utiliza repetidas veces un algoritmo más sencillo, y se combinan los resultados de estas repeticiones para dar un resultado común (ver [9], [10] y [11]). En el caso de los Random Forest lo que se utiliza son árboles de decisión (*decision trees*).

3.4.1. Decision Trees

Podemos ver un ejemplo de base de datos en la gráfica 9a, mientras que en 9b vemos una posible línea de decisión. Ésta puede parecer muy artificial, muy simple, pero funciona bien y es la que hace un árbol de decisión.

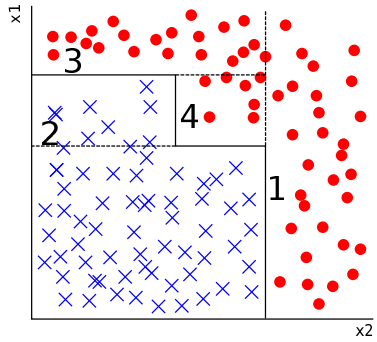


(a) Datos (b) Línea de decisión

Gráfica 9: Ejemplo de línea de decisión tomada por un árbol de decisión.

¿Cómo toma esa línea de decisión? El algoritmo consiste en ir separando los datos en regiones. En cada paso separará una región de los datos en dos, y el criterio de separación es sencillamente disminuir la imperfección (desorden) de las regiones. Para poner un ejemplo, en la gráfica 10, primero separaríamos los datos con la línea 1, consiguiendo una región sin imperfección a la derecha de la

línea. Luego la línea 2, que parte la región de la izquierda de la línea 1 en dos regiones, deja la región de abajo sin desorden. Con dos líneas más conseguimos separar perfectamente los datos.



Gráfica 10: Proceso para encontrar la línea de decisión de un Decision Tree.

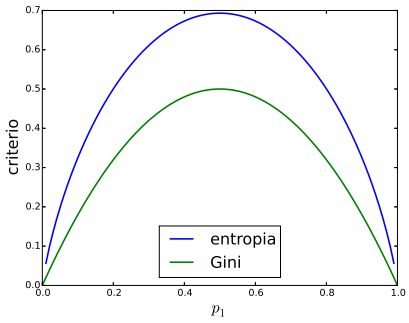
Hay varios criterios para medir el desorden de las regiones. El primero de ellos, la **entropía**, se define por

$$-\sum_k p_{mk} \log p_{mk} \tag{18}$$

donde p_{mk} es la proporción, en tanto por uno, de la clase k en la región m. Otro criterio es el criterio **Gini**, definido por

$$\sum_k p_{mk}(1 - p_{mk}) \tag{19}$$

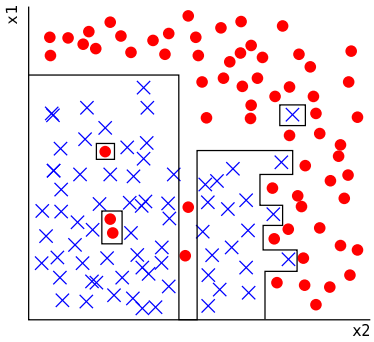
Hay más criterios, pero éstos son los más importantes y usados. El desorden medido por ambos criterios en función de la proporción de la clase 1 se muestra en la gráfica 11.



Gráfica 11: Desorden en función de p_1.

Uno de los mayores problemas de un árbol de decisión es su tendencia a sobreajustar los datos. Con unos datos como los usados en el ejemplo anterior no había ningún problema, porque eran fácilmente separables, pero con datos más realistas y complejos, podemos tener una línea de decisión como la de la

gráfica 12, con finas franjas para unos pocos datos y regiones aisladas para uno o dos puntos. Las bases de datos reales pueden ser aún más complejas que la de la gráfica, haciendo que la línea de decisión del árbol no tenga realmente ningún sentido general. Una de las maneras de resolver esto es exigir que el algoritmo no divida regiones con pocos datos, o que no deje regiones cerradas con menos de cierto número de datos. Estos pasos requieren de mucho "*cocinado*" para cada base de datos. Otra forma, bastante más extendida, es emplear varios árboles de decisión inicializados de forma aleatoria, creando un ***Random Forest***.



Gráfica 12: Los árboles de decisión son propensos al sobreajuste.

3.4.2. Random Forest

Ahora que ya sabemos lo que es un Árbol de Decisión, los *Random Forest* se construyen a partir de éstos, haciendo crecer varios árboles, cada uno de los cuales utiliza un conjunto aleatorio de N muestras del training set y M variables que irá separando como hemos visto. Con todos los árboles crecidos (entrenados), para cada entrada de datos nueva el *Random Forest* le asignará la clase más votada por los árboles.

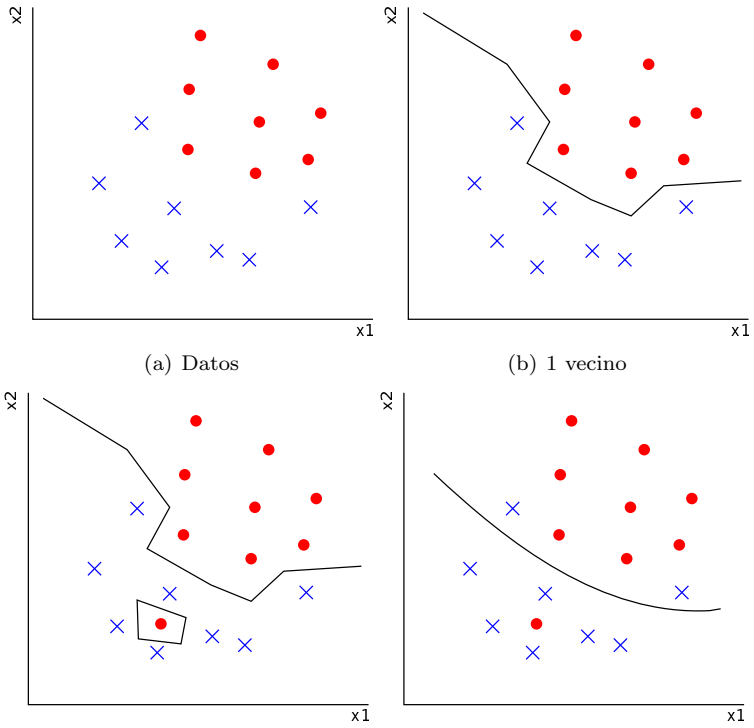
La forma de elegir las N muestras y M variables para cada árbol depende del tipo de *Random Forest* que hagamos, aunque siempre tienen que tener las mismas distribuciones de probabilidad para todos los árboles, y los árboles tienen que ser independientes entre sí, pues el error de generalización aumenta con la correlación entre árboles.

Aunque no vamos a entrar en detalles, por no hacer la explicación demasiado larga y tediosa, conviene saber que los *Random Forest* tienen un método para calcular las variables más importantes en nuestro training set, de forma que si tenemos muchas variables, podremos entrenar un primer *Random Forest*, ver qué variables han sido más importantes y entrenar un segundo *Random Forest* sólo con esas variables, ahorrando tiempo de cálculo y memoria. En el apartado 4, cada vez que hablemos de variables importantes en una base de datos nos estaremos refiriendo a las variables más importantes para un *Random Forest*.

3.5. K-Nearest Neighbors

Llegamos a otro algoritmo más sencillo aún que el *Naïve Bayes*, si esto puede ser posible. El *K-Nearest Neighbors* (KNN) se explica casi únicamente al leer el nombre(ver [12] y [13]). Se trata de un algoritmo que asigna a cada entrada

nueva de datos la clase más común entre las K entradas clasificadas más cercanas (del conjunto de aprendizaje), utilizando una métrica dada (normalmente la euclídea). Como parámetros a ajustar de este algoritmo, tenemos únicamente la métrica a utilizar y el número de vecinos.



(a) Datos (b) 1 vecino

(c) 1 vecino, *overfitting* (d) 3 vecinos, ya no hay *overfitting*

Gráfica 13: Con pocos vecinos, el algoritmo es propenso al *overfitting*.

En la grafica 13, vemos la sencillez del algoritmo. Hay que tener cuidado con el número de vecinos, porque si escogemos pocos podemos caer en *overfitting*. También es importante antes de entrenar el algoritmo, hacer un *feature scaling*[6], de manera que no haya una variable cuyas distancias entre datos sean mucho menores que el resto de variables, dando a todas ellas la misma importancia.

4. Aplicaciones

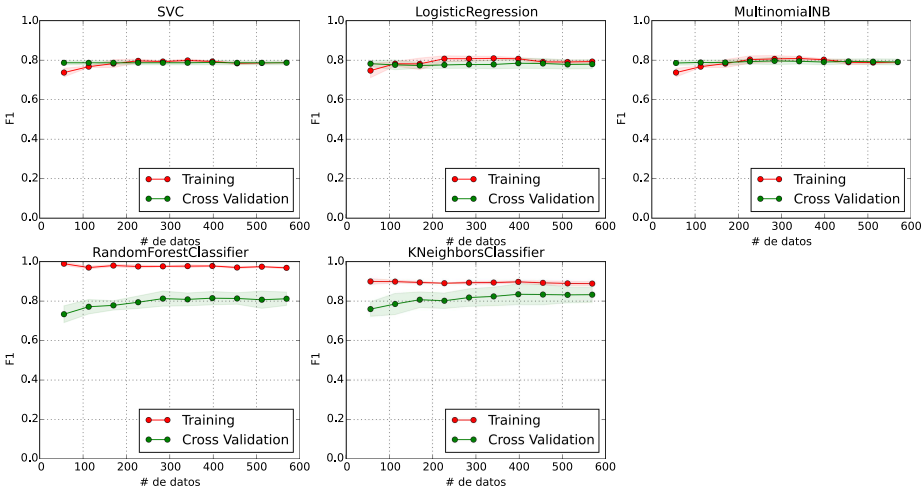
En esta sección veremos cómo se aplican los anteriores algoritmos a bases de datos reales. Para cada una de ellas explicaremos las variables de que se disponen, las peculiaridades de los datos y los resultados que extraemos.

[6]El *feature scaling* cambia todas las variables para que sus valores se encuentren en el intervalo [0,1]. Es importante hacerlo tanto para este algoritmo como para el *SVM* (sobre todo con el *kernel* gaussiano), aunque nosotros lo hacemos para todos los algoritmos por comodidad.

4.1. Supervivientes del Titanic

Esta base de datos se encuentra en la página web del Kaggle[7], una web en la que se proponen concursos abiertos de Machine Learning. Este ejemplo concreto consiste en un problema que proponen como iniciación para los nuevos usuarios, y consiste en conseguir detectar qué pasajeros sobrevivieron al accidente. Las variables con las que contamos son las siguientes: la categoría del billete, que es una variable categórica (1ª, 2ª o 3ª clase); el sexo, variable binaria; la edad; el número de parientes y esposas a bordo (sin contar padres ni hijos); el número de padres e hijos a bordo; el precio del billete; el lugar de embarque, variable categórica con valores C, Q y S, refiriéndose a Cherbourg, Queenstown y Southampton, y otra información no útil como el nombre del pasajero o un ID único para cada uno de ellos. Disponemos de los datos de 891 pasajeros ya clasificados como supervivientes o no, que usaremos para el training, CV y test set.

Al igual que con los dígitos, hacemos una primera curva de aprendizaje para asegurar que no vamos a necesitar más datos de entrenamiento, o que no tenemos demasiados. Esta curva se recoge en la gráfica 14. En ella podemos ver un efecto interesante: mientras que para casi todos los algoritmos las curvas del training set y las del CV set están próximas y son paralelas, en la gráfica del Random Forest vemos cómo en el training set el algoritmo consigue clasificar bastante bien los datos (valores de F1 en torno a 1), mientras que en el CV set está fallando mucho más (valores entre 0.7 y 0.85), aunque mejora ligeramente al aumentar los datos. Lo ideal sería que el valor F1 del CV set estuviese cercano al del training set, pero en este caso en el que hay tanta diferencia el algoritmo está sobreajustando los datos (overfitting), consigue métricas muy buenas para el training set, a costa de generalizar peor los datos del CV set. Sería bueno ver si, después de ajustar los parámetros con el procedimiento explicado en el apartado 2.3, seguimos cometiendo este overfitting, en cuyo caso habría que obtener más datos de pasajeros del Titanic o desechar directamente el algoritmo.



Gráfica 14: Curvas de aprendizaje del problema del Titanic.

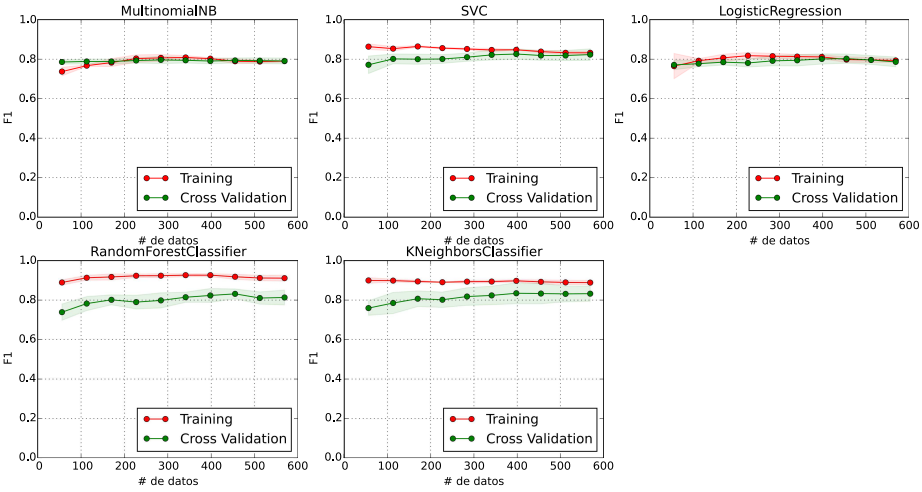
[7] `https://www.kaggle.com/c/titanic/data`

Realizamos ahora el Cross Validation. La malla inicial de parámetros, común a ambas bases de datos, es la siguiente: Para el SVC (Support Vector Classifier) usamos el kernel RBF con los valores de C 1, 10, 100, 1000, y los valores de γ 0.0001, 0.001, 0.01, 0.1, 1.0, y el kernel lineal con los mismos valores de C. Como el Naïve Bayes no depende de parámetros no podemos hacer sobre él ningún cross validation. Para el Logistic Regression usamos los mismos valores de C. Para el Random Forest usamos 5, 10 y 15 árboles, para el mínimo de datos requerido para poder separar una rama usamos 2, 4, 6, 8 y 10 datos, y para el mínimo de datos por hoja usamos 1, 2, 3, 4 y 5 datos. Para el KNN (K-Nearest Neighbors) usamos como número de vecinos 2, 3, 4, 5 y 6.

Después de hacer el proceso del Cross Validation, los mejores parámetros que hemos encontrado son los siguientes: Para el SVC, el kernel RBF con $C = 10$ y $\gamma = 1.0$; para el Logistic Regression, $C = 100$, para el Random Forest, 15 árboles, con 8 datos como mínimo para separar una rama y con 1 dato como mínimo para cada hoja; y para el KNN 3 vecinos. Las métricas medidas por cada algoritmo se recogen en la tabla 2.

Algoritmo	Accuracy	Precision	Recall	F1-Score
Multinomial NB	0.7821	0.7231	0.6912	0.7068
SVC	0.7933	0.7246	0.7353	0.7299
Logistic Regression	0.7709	0.6957	0.7059	0.7007
Random Forest	0.8156	0.7397	0.7941	0.7660
KNN	0.7821	0.6986	0.7500	0.7234

Tabla 2: Métricas del Titanic para los distintos algoritmos.



Gráfica 15: Curvas de aprendizaje del problema del Titanic después de hacer el Cross Validation.

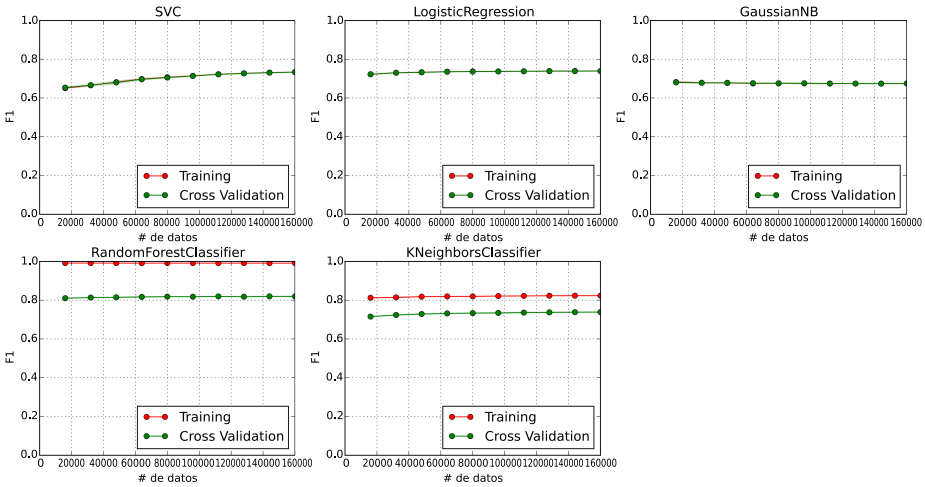
En la gráfica 15 vemos que no ha mejorado mucho el overfitting del Random Forest, lo cual se puede deber a que tenemos muy pocas variables, muchas de ellas con pocos valores posibles (categóricas), por lo que los árboles de decisión no tienen muchas posibilidades de separar los datos de muchas formas.

En la tabla 2 podemos elegir el mejor algoritmo que hayamos logrado. Podríamos pensar que es el Random Forest, con un valor F1 de 0.7660, pero, como sabemos que está sobreajustando los datos, no está generalizando bien **¿ESTOY SEGURO DE ESTO?**. Los dos mejores algoritmos, una vez despreciamos el Random Forest, son el Support Vector Machine y el K-Nearest Neighbors, con 0.7299 y 0.7234 como valores F1 respectivamente. Aunque estos valores no son muy diferentes, si nos volvemos a fijar en la gráfica 15 vemos que la curva del CV set y la del training set están más próximas en el SVC que en el KNN, por lo que el SVC tiene menos sesgo, generalizará mejor para datos nuevos. El KNN sería mejor si dispusiésemos de nuevos datos, como se refleja en la gráfica, pero con los datos que tenemos, el SVC es el mejor algoritmo para esta base de datos.

4.2. Búsqueda del bosón de Higgs

Por último, vamos a estudiar una base de datos grande, la del bosón de Higgs. Ésta también viene de un concurso del Kaggle[8] celebrado en 2012. Consta de 250000 datos, cada uno con 17 variables primitivas, medidas directamente, y 13 variables derivadas de las anteriores. Muchas entradas de datos tienen variables sin su valor, bien porque no se haya podido medir o porque no tiene ningún sentido físico, en esos casos introduciremos un valor nuevo en la variable, que será la media de todos los valores de esa variable.

Lo primero de todo, como la base de datos es bastante grande, es pintar las curvas de aprendizaje, recogidas en la gráfica 16.



Gráfica 16: Curvas de aprendizaje del bosón de Higgs.

En la gráfica 16 podemos ver cómo tanto el Random Forest como el KNN están sobreajustando los datos. El SVC parece que mejora al utilizar más datos, incrementando en 0.1 su valor F1 desde el 10 % de los datos hasta el 100 % de ellos. Tanto el Logistic Regression como el Gaussian NB no mejoran demasiado con más datos, así que utilizaremos sólo el 10 % de ellos, consiguiendo una

[8] `https://www.kaggle.com/c/higgs-boson/data`

notable aceleración en el tiempo de cálculo. Si tras el CV vemos que el Random Forest o el KNN dejan de sobreajustar, o que el SVC generaliza bien los datos, nos plantearemos la posibilidad de usar todos los datos para el entrenamiento de los algoritmos.

Tras el proceso de Cross Validation sobre el 10 % de los datos obtenemos las métricas de la tabla 3. Los mejores parámetros encontrados son: para el SVC, kernel lineal con $C = 1000$ y $\gamma = 0.1$; para el Logistic Regression $C = 100$; para el Random Forest 15 árboles, con un mínimo de 10 muestras para separar una rama, y con un mínimo de 4 muestras por hoja; para el KNN, 5 vecinos. Algunos de estos parámetros son los extremos de la malla inicial que estamos utilizando, por ejemplo el valor de C del SVC, el número de árboles o las muestras mínimas para separar una rama del Random Forest, lo cual nos anima a volver a hacer un Cross Validation con una malla más "fina" en torno a estos parámetros. Además, estos dos algoritmos son los que mejores métricas han dado, merece la pena perder un poco de tiempo en tratar de mejorarlos.

Algoritmo	Accuracy	Precision	Recall	F1-Score
Gaussian NB	0.6858	0.7367	0.8144	0.7736
SVC	0.8129	0.8413	0.8826	0.8614
Logistic Regression	0.7412	0.7810	0.8608	0.8190
Random Forest	0.8252	0.8450	0.9000	0.8716
KNN	0.7249	0.7855	0.8015	0.7934

Tabla 3: Métricas del Higgs para los distintos algoritmos.

La nueva malla que probamos para estos algoritmos es así: para el SVC, con el kernel rbf, $\gamma \in \{0.01, 0.03, 0.1, 0.3, 1.0\}$ y $C \in \{100, 300, 1000, 3000, 10000\}$; mientras que para el Random Forest usamos 10, 13, 15, 17 y 20 árboles, con 8, 9, 10 y 11 como mínimo de muestras para separar una rama, y con 2, 3, 4, 5 y 6 como mínimo de muestras por hoja.

Los nuevos parámetros encontrados son, para el SVC, $C = 10000$y $\gamma = 0.03$, y para el Random Forest, 20 árboles, con 11 muestras para separar una rama y con 5 muestras como mínimo por hoja. Las métricas conseguidas por estos algoritmos se recogen en la tabla 4.

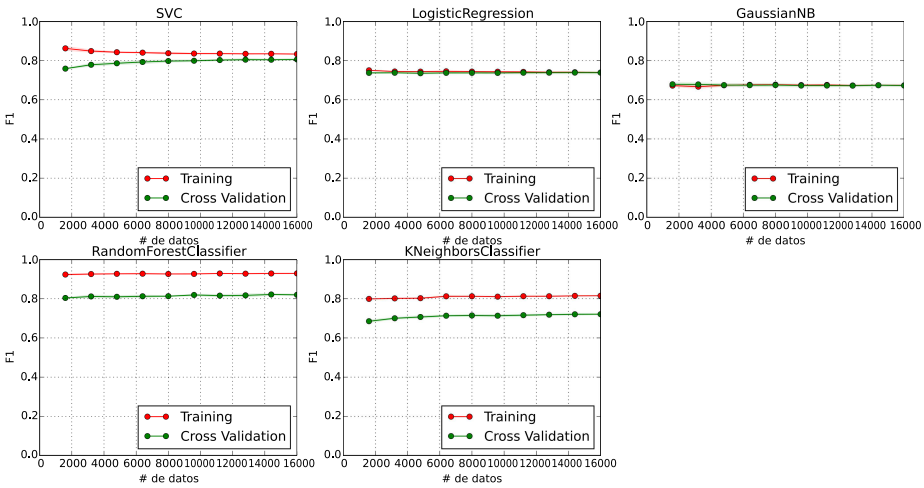
Algoritmo	Accuracy	Precision	Recall	F1-Score
SVC	0.8133	0.8408	0.8842	0.8619
Random Forest	0.8270	0.8476	0.8993	0.8727

Tabla 4: Métricas del Higgs para los nuevos algoritmos.

Como podemos ver, comparando las tablas 3 y 4, no han mejorado mucho nuestros algoritmos, y los parámetros vuelven a estar en los extremos de las mallas. Seguramente en la vida real estos valores de las métricas no serían suficientes, y tendríamos que seguir probando hasta tener valores mejores. En la práctica, debido al tiempo de cálculo, para el objetivo de este estudio no nos importa tanto tener el mejor algoritmo sino saber usarlo, así que nos quedaremos con éstos como los mejores parámetros.

Como vimos en la sección 3.4, el Random Forest nos proporciona un método para ver la importancia relativa de las variables. Gracias a ello vemos que las diez variables más importantes para esta base de datos son DER_mass_MMC, DER_mass_transverse_met_lep, DER_mass_vis, DER_met_phi_centrality, PRI_tau_pt, DER_deltar_tau_lep, DER_pt_ratio_lep_tau, PRI_met, DER_pt_h y DER_sum_pt[9], con importancias relativas de 0.1513, 0.1330, 0.0935, 0.0731, 0.0632, 0.0493, 0.0448, 0.0401, 0.0289 y 0.0277 respectivamente. No nos sorprende que la variable más importante sea precisamente la masa estimada del candidato a bosón, pues es una característica casi definitoria del bosón.

Representamos ahora en la gráfica 17 las curvas de aprendizaje con los nuevos algoritmos, para ver cómo generalizan, y si hace falta usar todos los datos.



Gráfica 17: Curvas de aprendizaje para el Higgs después de hacer el Cross Validation

Podemos ver que el KNN y el Random Forest siguen sobreajustando los datos, y no parece que esto se vaya a solucionar trabajando con más datos, por la escasa pendiente de ambas curvas. Tampoco van a mejorar significativamente el Logistic Regression ni el Gaussian NB, pues tampoco tienen mucha pendiente sus curvas. El SVC presenta una curva de aprendizaje "de libro", en la que el valor F1 del training set va descendiendo, mientras que el del CV set va aumentando, ambos con mayor pendiente al principio, disminuyendo ésta al usar más datos. En la gráfica podemos ver que el SVC tampoco va a mejorar mucho más usando más datos, porque al final de la gráfica las dos curvas se están estabilizando, así que descartamos la idea de volver a correr los algoritmos con todos los datos de los que disponíamos.

Para terminar, como el Random Forest seguía sobreajustando los datos, podemos decir que para esta base de datos el mejor algoritmo es el Support Vector Machine.

[9]Ver [14] para mayor información sobre estas variables

5. Conclusiones

Para concluir este trabajo, vamos a ver qué hemos conseguido. Primero hemos podido conocer los fundamentos del Machine Learning, cómo usar los algoritmos, qué casos nos podemos encontrar, si hay que preparar los datos antes de alimentar al algoritmo, etc. Luego hemos ido viendo las peculiaridades de cada algoritmo, en qué se basa, cómo funciona, qué tipo de datos necesita, si tiene alguna restricción especial que haya que considerar aparte, qué tipo de curvas de decisión realiza y cómo podemos utilizar esas curvas, etc. Hasta aquí la parte teórica del trabajo, nos quedaba mancharnos las manos con la práctica.

A la hora de utilizar realmente los algoritmos, hemos usado tres bases de datos, un par de ellas casi "de juguete", y otra más seria, la del Higgs. En la primera hemos visto que teníamos tres algoritmos que clasificaban a la perfección los datos, es decir, todas las muestras de imágenes que eran el dígito 0 las predecía como 0, y las que eran 1 las predecía como tal. Estos algoritmos eran el Logistic Regression, el Random Forest y el K-Nearest Neighbors.

En la segunda base de datos vimos que sólo éramos capaces de conseguir valores F1 en torno a 73-76 %. Ésto se debía a que tenemos pocas variables a estudiar, y además las clases no seguían una distribución clara, siendo más difícil diferenciarlas. Hay que tener en cuenta que el Machine Learning no es un truco de magia, son algoritmos matemáticos, así que si un experto humano es incapaz de sacar mucha información de una base de datos, un algoritmo de Machine Learning tampoco podrá. Pongamos el ejemplo de un dado perfectamente equilibrado. Si le dices a cualquier hombre qué números han salido en las últimas 100000 tiradas, éste no podrá predecir qué saldrá en las siguientes, aunque sepa datos como quién tira el dado, qué temperatura hace o hacia dónde lo tira. Los algoritmos jamás podrán obtener información que no pudiese obtener un experto en la materia.

Una vez aclarado esto, recordamos que en esta base de datos el algoritmo que mejor funcionó fue el Support Vector Machine, con un valor F1 del 73 %. Para elegir éste algoritmo tuvimos que descartar el Random Forest, con un mejor valor F1 (76 %), pero sobreajustaba los datos del training set, así que no nos podíamos fiar de su capacidad de generalización.

Por último, la base de datos del Higgs. Como la base de datos es bastante grande, aprovechamos la información que nos brindaban las curvas de aprendizaje[10] y pudimos usar sólo un 10 % de los datos. Luego vimos de forma un poco más seria cómo llevar a cabo el Cross Validation sobre una base de datos, probando mallas de parámetros cada vez más finas, hasta alcanzar el nivel de precisión deseado.

Vimos cómo el Random Forest acertaba al escoger las variables más importantes, siendo la masa estimada la mejor. Al final, decidimos que el mejor algoritmo para esta base de datos, al nivel de profundidad al que la estudiamos, era el SVC, aunque seguramente, si hubiéramos seguido probando mallas más finas, podría haber sido el Random Forest.

De este trabajo podemos concluir que no existe un algoritmo mejor que el resto, sino que cada base de datos tiene "su algoritmo", uno con el que es más fácil sacar toda la información posible de los datos. También concluimos, gracias

[10] Mi portátil tardó en hacerlas 3 días enteros, porque para hacerlas hay que entrenar cada algoritmo cada vez con distinto número de muestras

a la base de datos del Titanic, que donde no hay información no se puede sacar, que los algoritmos no inventan información, sólo tratan de sacarla de los datos.

Referencias

[1] S. Bhattacharyya, S. Jha, K. Tharakunnel y J. C. Westland. *Decision Support Systems*, **50**, 602-613 (2011).

[2] D. W. Hosmer, S. Lemeshow, R. X. Sturdivant. Applied Logistic Regression, *John Wiley & Sons* (2013).

[3] S. Russel, P. Norvig. Artificial Intelligence: A Modern Approach, *Prentice Hall* (2010).

[4] I. Rish. An empirical study of the naive Bayes classifier, *IBM Research Report* **No sé cómo citar esto...** (2001).

[5] J. E. Hernández, S. Salazar. *Scientia et Technica*, **31**, 47-52 (2006)

[6] C. Cortes, V. Vapnik. *Machine Learning*, **20**, 273-297 (1995)

[7] V. Vapnik. Statistical Learning Theory, *John Wiley & Sons* (1998).

[8] J. L. Martínez Pérez. *Comunicación con computador mediante señales cerebrales. Aplicación a la tecnología de la rehabilitación* (2009). Tesis doctoral de la ETSII (UPM). **Tampoco sé si se cita así...**

[9] L. Breiman. *Machine Learning*, **45**, 5-32 (2001).

[10] L. Rokach, O. Maimon. Data Mining with Decision Trees, Theory and Applications, *World Scientific* (2015).

[11] J. R. Quinlan. *Machine Learning*, **1**, 81-106 (1986).

[12] T. M. Cover, P. Hart. *IEEE Transactions on Information Theory*, **13**, 21-27 (1967)

[13] T. M. Cover. *IEEE Transactions on Information Theory*, **14**, 21-27 (1968)

[14] C. Adam-Bourdarios, G. Cowan, C. Germain, I. Guyon, B. Kégl y D. Rousseau. Learning to discover: the Higgs boson machine learning chalenge (2014)