Algoritmos de Machine Learning y sus aplicaciones

Trabajo de Fin de Grado

Javier Díaz Bustamante Ussia

3 de julio de 2015

Índice

Resumen

En este trabajo vamos a estudiar diferentes algoritmos de *Machine Learning* para la clasificación de sucesos. Para comprobar su funcionamiento, los utilizaremos sobre distintas bases de datos, una de reconocimiento de dígitos, una de supervivientes del Titanic y la última 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 detrás, 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*.

Al calor del 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 (también los hay de regresión). Estos cinco son *Logistic Regresion*, *Naïve Bayes*, *Support Vector Machine*, *Random Forest* y *K-Nearest Neighbors*, explicados en detalle en la seccion 3.

Existen dos tipos de problemas en *ML*, los de clasificación y los de regresión. Estos últimos tratan de predecir el valor de una variable para un conjunto de datos nuevos, como por ejemplo la regresión por mínimos cuadrados. Los problemas de clasificación consisten en tratar de asignar a cada entrada de datos nuevos una clase concreta, pudiendo ser una clasificación binaria (verdadero o falso) o una clasificación multiclase, como por ejemplo un algoritmo de reconocimiento de dígitos. A lo largo de este trabajo, por similitud con el problema del bosón de Higgs, trataremos únicamente con problemas de clasificación binaria.

2. Consideraciones previas

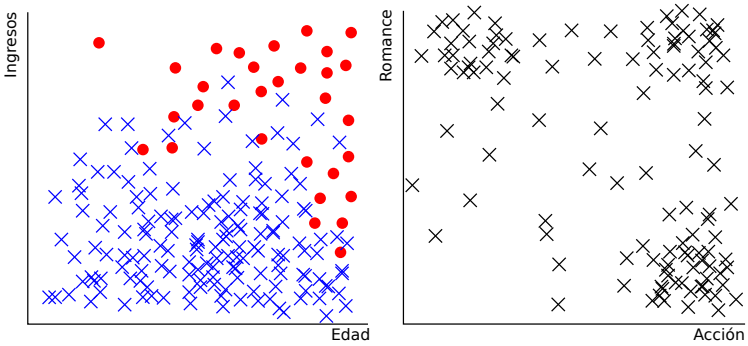
2.1. Aprendizaje supervisado y no supervisado

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 (ya veremos más adelante cómo). 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 training set 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 de training set ficticio 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 los romances un 80 %, mientras que las películas de acción sólo un 57 %. A otra, sin embargo, le gustan las primeras al 43 % y las segundas al 96 %. 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.

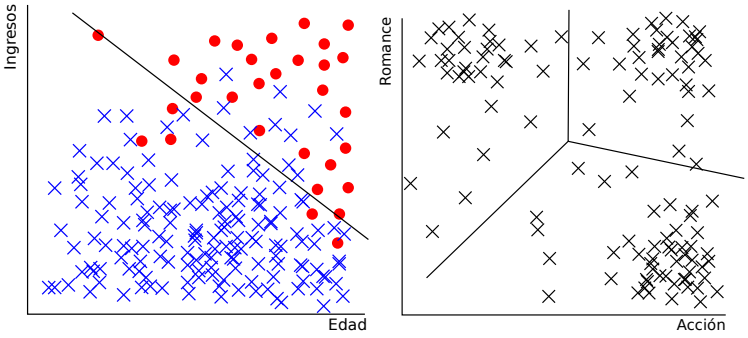


(a) Supervisado. En rojo, los amantes de la música clásica.

(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 training set, es hora de probarlo. Para ello se emplea otro conjunto de datos, llamado 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

[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.

y según la etiqueta predicha por el algoritmo, se pueden dar los casos de la tabla 1.

	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[2]:

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \tag{1}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2}$$

$$\text{Specificity} = \frac{TN}{FP + TN} \tag{3}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{4}$$

$$\text{F1-score} = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5}$$

La ecuación (2) (sensibilidad) nos da una medida de la exactitud de los casos positivos, mientras que (3) (especificidad) nos da la de los casos negativos. (4) (precisión) nos da la exactitud de los casos clasificados como positivos.

Aunque pudiera parecer que (1) (exactitud) es una buena medida de la bondad de la clasificación, 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, ecuación (5), 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

[2]Ver [1].

del test set, con lo que podremos calcular nuestras métricas como vimos en las ecuaciones 1 a 5. 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 training set, sino cualquier nuevo conjunto de datos del que dispongamos.

2.3. Cross Validation

La mayoría de los algoritmos de *ML* dependen de unos parámetros que hay que fijar manualmente. Por ejemplo, muchos de ellos utilizan el *parámetro de regularización*, C, que veremos a continuación. El problema es que los valores de estos parámetros influyen en gran medida en la bondad de la clasificación, y una mala elección de los parámetros nos puede hacer pensar que un algoritmo es malo, cuando la realidad es que lo estamos utilizando mal.

Para fijar correctamente estos parámetros, nos ayudamos del Cross Validation set. Hasta ahora el procedimiento a seguir era entrenar el algoritmo con el training set, y ver la bondad del mismo con el test set. Ahora, para fijar correctamente estos parámetros, usaremos un tercer conjunto, el Cross Validation set, o CV set. El procedimiento a seguir ahora es el siguiente:

Primero, elegimos valores para nuestros parámetros. Con esos valores, entrenamos el algoritmo con el training set. Evaluamos las métricas con el CV set, y volvemos a realizar estos pasos con valores distintos de los parámetros, eligiendo el conjunto de parámetros que mejor prediga las clases del CV set. Por último, con el algoritmo entrenado con los mejores parámetros, evaluamos las métricas en el test set. Estas métricas son las que nos dirán cómo de acertado será nuestro algoritmo al clasificar datos nuevos.

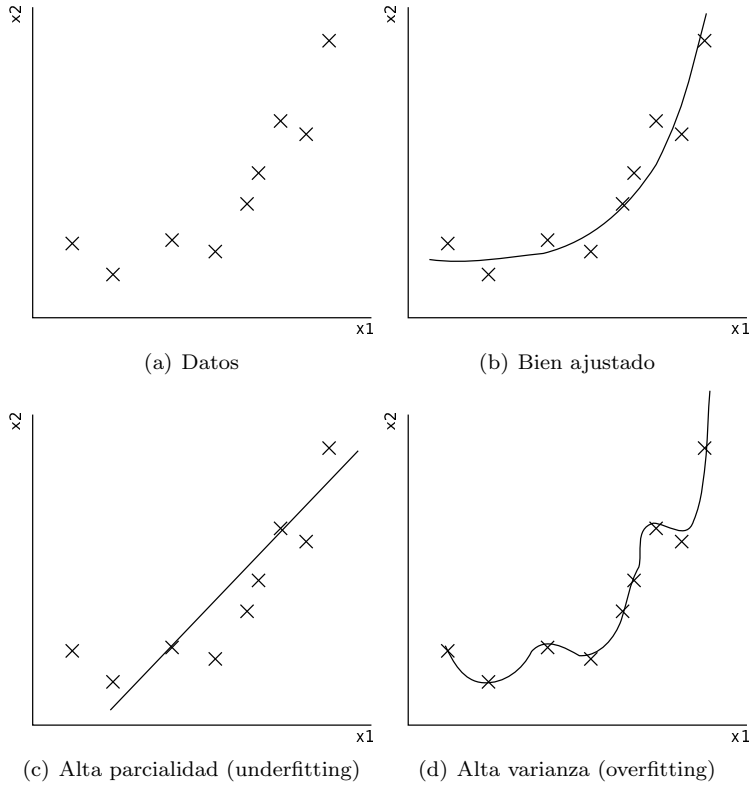
Al igual que antes, el proceso de evaluar métricas se realiza en conjuntos de datos que no hayamos utilizado para entrenar el algoritmo, aunque esta vez tampoco usamos el conjunto que nos ha servido para ajustar los parámetros. Ésto se debe a que, al usar el CV set para ajustar los parámetros, éste ya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

2.4. Varianza, Parcialidad y Regularización

Al entrenar un algoritmo nos enfrentamos a dos errores diametralmente opuestos, la *parcialidad* y la *varianza*. Tendremos alta parcialidad (*underfitting*) 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. 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 regularización, que veremos en seguida.

Por el contrario, tendremos alta varianza (*overfitting*) 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. Para solucionarlo podremos tomar más datos, simplificar la hipótesis o modificar el parámetro de regularización. Podemos ver un claro ejemplo de ambos problemas en la gráfica 3.



(a) Datos (b) Bien ajustado

(c) Alta parcialidad (underfitting) (d) Alta varianza (overfitting)

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

A continuación explicamos en qué consiste el parámetro de regularización. 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 regularización se encarga 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 regularización[3] C se introduce como un término adicional al error que se minimiza. Este término suele ser proporcional al inverso de C y a la suma cuadrática de los θ_j, de forma que un valor alto de C respetará más la hipótesis del algoritmo que un valor bajo de C. En resumen, el nuevo error sería:

$$J'(\boldsymbol{\theta}) = J(\boldsymbol{\theta}) + \frac{1}{C}\boldsymbol{\theta}^T\boldsymbol{\theta} \tag{6}$$

Reduciendo C se consigue suavizar la hipótesis del algoritmo, minimizando el vector de pesos $\boldsymbol{\theta}$, reduciendo 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 la parcialidad del mismo. Por lo tanto, el parámetro de regularización nos ayuda a mejorar nuestro algoritmo, y lo podremos fijar con el CV set, 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*), es decir, $h_{\boldsymbol{\theta}}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{Z}/p\mathbb{Z} : \boldsymbol{x} \mapsto y$, donde n es la dimensión del vector $\boldsymbol{x}$, p es el número de clases y $\boldsymbol{\theta}$ es el vector de pesos. 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, es decir, $J : \mathbb{R}^q \to \mathbb{R}^+ : \boldsymbol{\theta} \mapsto J(\boldsymbol{\theta})$, donde q es la dimensión del vector de pesos, que la mayor parte de las veces coincide con la dimensión de las variables. La forma concreta de la función objetivo también depende del algoritmo, así que vamos a estudiar los que utilizaremos.

3.1. Logistic Regression

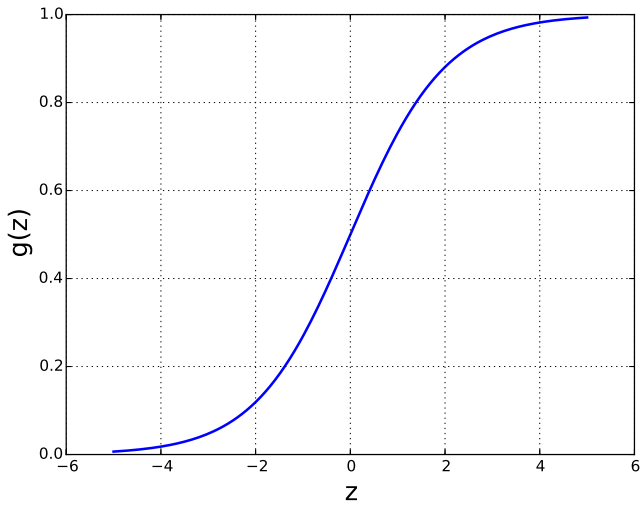
El algoritmo de regresión logística[4] es un algoritmo de modelo lineal, es decir, siempre producirá una superficie de decisión que dependa linealmente de las variables de entrada, aunque si queremos una superficie más compleja sólo tendremos que crear nuevas variables no lineales con las de entrada.

Se llama de regresión logística porque precisamente para la hipótesis hace uso de una función logística, que tiene la forma de la ecuación 7. 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.

[3] A veces se utiliza $\lambda \propto C^{-1}$.

[4] Ver el capítulo 1 de [3]

$$g(z) = \frac{1}{1 + e^{-z}} \tag{7}$$



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

Sea $\boldsymbol{x}$ el vector columna de variables de entrada, y $\boldsymbol{\theta}$ el vector columna 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, como ahora veremos. La hipótesis del algoritmo es (8), donde $g(z)$ es la función logística de (7). Ésta 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".

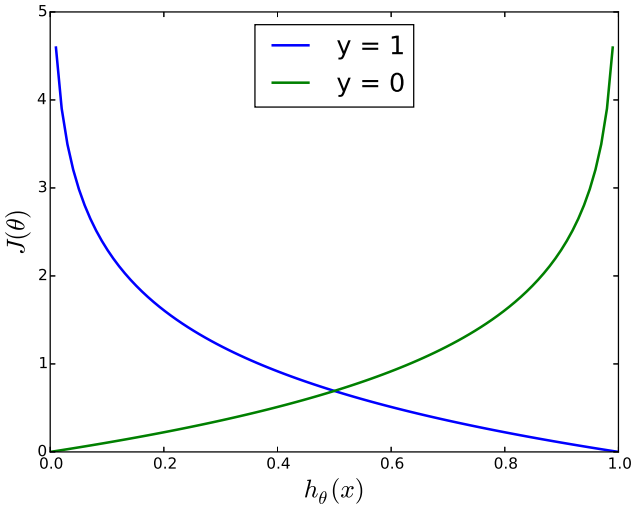
$$\boxed{h_{\boldsymbol{\theta}}(\boldsymbol{x}) = p(y = 1|\boldsymbol{x}; \boldsymbol{\theta}) = g(\boldsymbol{\theta}^T \boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \boldsymbol{x}}}} \tag{8}$$

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

La función de coste nos da el error cometido por la hipótesis, y en el caso de la regresión logística tiene la forma de la ecuación 9.

$$\boxed{J(\boldsymbol{\theta}) = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log \left(h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) + \left(1 - y^{(i)} \right) \log \left(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}) \right) \right]} \tag{9}$$

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



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[5]. Se basa en asumir ingenuamente que las variables de las entradas de datos de una clase determinada son independientes, lo que nos puede asustar al oirlo por primera vez, aunque veremos que, a pesar del calibre de la simplificación, funciona sorprendentemente bien a pesar de que las variables no sean realmente independientes[6]. Esta asunción se expresa con la siguiente expresión

$$p(\boldsymbol{x}|C) = \prod_{j=1}^{n} p(x_j|C) \tag{10}$$

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 $\boldsymbol{x}$ se de la clase C_l es:

$$\boxed{p(C_l|\boldsymbol{x}) = \frac{p(C_l)p(\boldsymbol{x}|C_l)}{p(\boldsymbol{x})}} \tag{11}$$

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 que maximice la probabilidad de la ecuación 11. Ésta es la hipótesis del algoritmo. Nótese que, dado un vector $\boldsymbol{x}$, $p(\boldsymbol{x})$ es una constante, por lo que sólo tenemos que preocuparnos por maximizar el numerador, teniendo en cuenta (10).

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

[5] Ver capítulo 20 de [4].
[6] Ver [5].

por ejemplo, el "Naive Bayes Gaussiano", el "Naive Bayes Multinomial" o el "Naive Bayes Binomial", todos ellos nombrados según la distribución de probabilidad de las variables. También se pueden hacer algoritmos de Naive Bayes mixtos, donde a distintas variables les correspondan distintas distribuciones de probabilidad.

3.3. Support Vector Machine

3.4. Random Forest

3.5. K-Nearest Neighbors

4. Bases de datos

4.1. Reconocimiento de dígitos

4.2. Supervivientes del Titanic

4.3. Búsqueda del bosón de Higgs

5. Resultados

6. Conclusiones

Referencias

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

[2] S. Jha, M. Guillen, J. C. Westland. *Expert System with Applications*, **39**, 12650-12657 (2012).

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

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

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