

Maestría en Inteligencia Artificial

Aprendizaje Automático

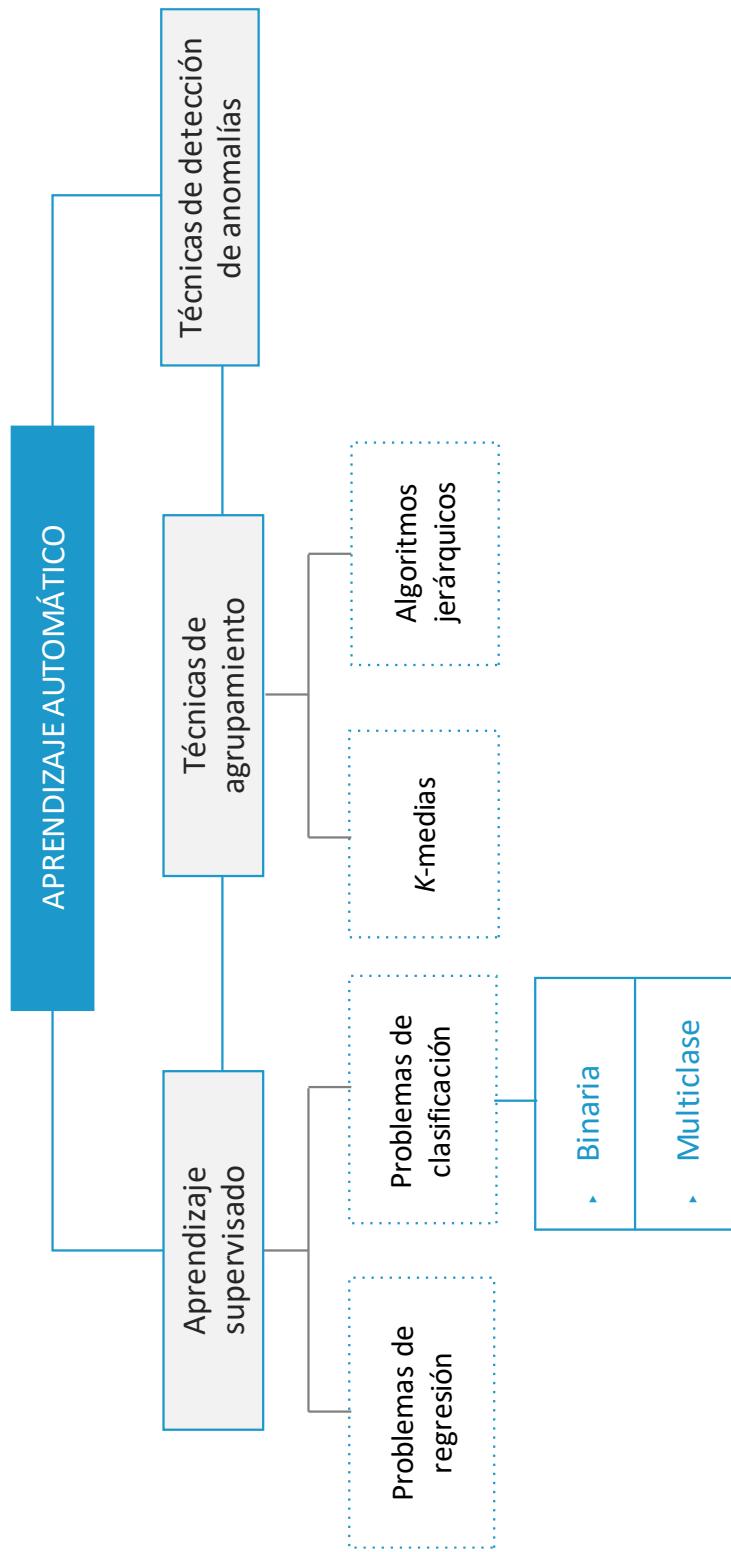
Aprendizaje Automático

Introducción al aprendizaje automático

Índice

Esquema	3
Ideas clave	4
1.1. Introducción y objetivos	4
1.2. Aprendizaje supervisado: problemas de regresión	5
1.3. Conjuntos de entrenamiento, test y validación cruzada	8
1.4. Técnicas de agrupamiento	11
1.5. Referencias bibliográficas	14

Esquema



1.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos del aprendizaje automático» a través del aula virtual

En este tema os introduciremos en los conceptos básicos del aprendizaje automático. Se entiende por **aprendizaje automático** aquellos algoritmos que se ejecutan en los ordenadores para aprender automáticamente en base a los datos proporcionados. Se trata de crear programas capaces de generalizar comportamientos a partir de los datos suministrados en forma de ejemplos. Es por tanto un proceso de inducción del conocimiento.

El aprendizaje automático, también conocido como *machine learning*, puede dividirse en algoritmos de **aprendizaje supervisado** y algoritmos de **aprendizaje no supervisado**. El aprendizaje supervisado utiliza ejemplos conocidos para obtener las inferencias mientras que el aprendizaje no supervisado no dispone de ejemplos con un objetivo o etiqueta conocido. A su vez, los **problemas** se pueden dividir en los siguientes cuatro sub-tipos:

- ▶ Aprendizaje supervisado: problemas de regresión.
- ▶ Aprendizaje supervisado: problemas de clasificación.
- ▶ Aprendizaje no supervisado: problemas de agrupamiento.
- ▶ Aprendizaje no supervisado: problemas de detección de anomalías.

En este tema se describen las **principales características y diferencias** de estos cuatro grupos de problemas que forman parte del aprendizaje automático. Además, se describen las diferencias entre los conjuntos de datos de entrenamiento, test y validación cruzada.

1.2. Aprendizaje supervisado: problemas de regresión



Accede al vídeo «Aprendizaje supervisado y no supervisado» a través del aula virtual

El aprendizaje automático **surge a mediados de los años 80** con la aplicación de las redes de neuronas y los árboles de decisión. El aprendizaje automático se empezó a utilizar en problemas de predicción complejos donde los modelos estadísticos clásicos no eran muy buenos como, por ejemplo, el reconocimiento de voz e imágenes, la predicción de series temporales no lineales, la predicción de los mercados financieros, el reconocimiento de texto escrito, etc.

El **aprendizaje supervisado** es un tipo de aprendizaje automático donde se realizan inferencias por medio de una función que establece una correspondencia entre las entradas y la salida del sistema. En el aprendizaje supervisado tenemos datos que son generados por una «caja negra» donde un vector de variables de entrada x (llamadas variables independientes) entran por un lado y por otro lado las variables respuesta y son obtenidas.

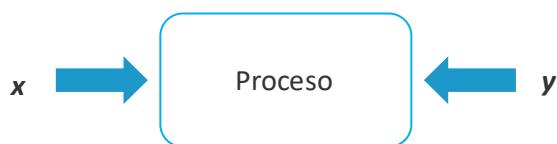


Figura 1. El aprendizaje automático busca el proceso que relaciona las variables de entrada x con las variables respuesta y .

En el aprendizaje supervisado para cada observación de las variables predictoras x existe una medida de la variable respuesta y . En el caso concreto de los problemas de regresión la variable respuesta y del sistema que se desea inferir o generalizar es una variable cuantitativa (numérica continua).

El objetivo del aprendizaje supervisado es **predecir las respuestas que habrá en el futuro** con nuevas variables de entrada. Para ello se utilizan algoritmos como modelos y se trata al mecanismo de generación de los datos como algo desconocido. Es decir, se considera el interior de la caja como algo complejo y desconocido. Por tanto, el enfoque es buscar una función $f(x)$ que opere con los datos x para producir las respuestas y .

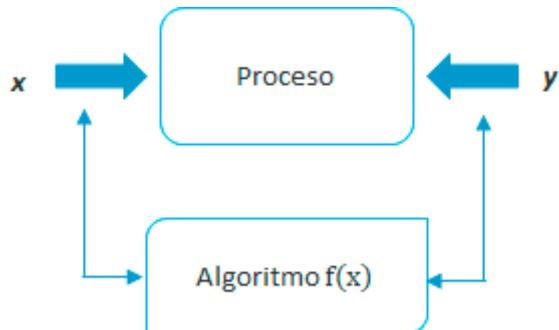


Figura 2. En el aprendizaje supervisado por medio de algoritmos se busca la función $f(x)=y$ que relaciona la entrada con la salida.

La evaluación de este modelo se lleva a cabo por medio de la capacidad predictiva del modelo.

El **aprendizaje supervisado** tiene como objetivo generalizar las respuestas sobre datos no observados, utilizando para ello **ejemplos** observados previamente. En el caso de los problemas de **regresión** la variable respuesta y es una variable numérica continua.

Aprendizaje supervisado: problemas de clasificación

El otro gran grupo de algoritmos de aprendizaje supervisado son los que están enfocados a problemas de clasificación. En los problemas de clasificación, el aprendizaje supervisado utiliza ejemplos conocidos para inferir la etiqueta (clasificar) de los vectores de entrada x eligiendo una de entre varias categorías o clases.

Estos algoritmos, al igual que en los problemas de regresión, utilizan ejemplos etiquetados previamente para «aprender» los patrones para llevar a cabo una clasificación.

En este tipo de problemas la variable respuesta *y* es una variable con dos o más categorías.

En los **problemas de clasificación**, utilizando aprendizaje supervisado, el objetivo es identificar a qué categoría pertenece una nueva observación utilizando para ello una serie de observaciones y categorías conocidas previamente.

Un ejemplo sería asignar a un correo electrónico la categoría de *spam* o no *spam* en función de los correos recibidos previamente. Otro ejemplo es realizar un diagnóstico a un paciente en función de sus características (sexo, presión sanguínea, colesterol, etc).

Los **problemas de clasificación** se dividen en dos grandes grupos: clasificación binaria y clasificación multi-clase. Los problemas de clasificación binaria buscan diferenciar las nuevas observaciones entre una de las dos clases posibles (ejemplo *spam* y no *spam*). Los problemas de clasificación multi-clase llevan asignar una nueva observación a una de entre varias clases posibles.

El aprendizaje supervisado tiene como objetivo generalizar utilizando para ello ejemplos conocidos. En el caso de los problemas de clasificación la variable respuesta y es una variable con 2 o más categorías o clases.



Accede a los ejercicios de autoevaluación a través del aula virtual

1.3. Conjuntos de entrenamiento, test y validación cruzada



Accede al vídeo «Conjunto de entrenamiento, test y validación cruzada» a través del aula virtual

La clave de cualquier modelo de aprendizaje automático es su capacidad de generalizar situaciones del futuro en función de los datos históricos observados. En la siguiente figura se describe el proceso de entrenamiento y predicción a alto nivel. En la fase de **entrenamiento** (a) se extraen las características o variables relevantes de los datos de entrada para construir un modelo por medio de algoritmos de aprendizaje automático. Posteriormente en la fase **predicción** (b) se realiza una extracción de variables similar sobre las que se aplica el modelo previamente entrenado para obtener el resultado estimado.

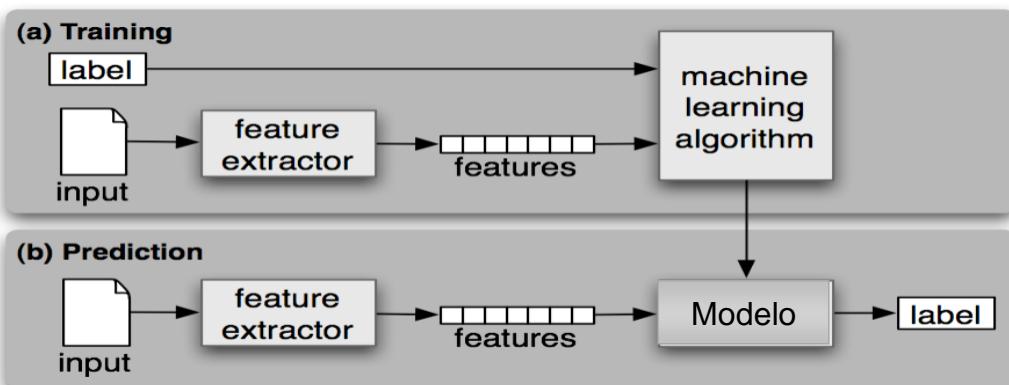


Figura 3. Ejemplo de las fases de entrenamiento (a) y predicción (b) en los procesos de aprendizaje automático. Fuente: <http://www.nltk.org/book/ch06.html>

Durante la fase de construcción de los modelos de aprendizaje automático las diferentes implementaciones software proporcionan métricas de error. Estas métricas suelen obtenerse con el **conjunto de datos** utilizado para realizar el entrenamiento. Este conjunto de datos se conoce con el nombre de *training set* o **conjunto de entrenamiento**. Las métricas obtenidas con los datos de entrenamiento

deben utilizarse solo como referencia, pues no son buenos indicadores del comportamiento futuro.

Un modelo puede ser capaz de tener un error mínimo con datos históricos (conjunto de entrenamiento) y no ser capaz de predecir bien los valores futuros. Por esta razón el error en el conjunto de entrenamiento suele ser un valor muy optimista y debe de ser interpretado con cautela.

Para solucionar el problema anterior, una buena práctica es utilizar un **conjunto de datos de test**. Este conjunto de datos de test puede estar formado con un subconjunto de los datos de entrenamiento.

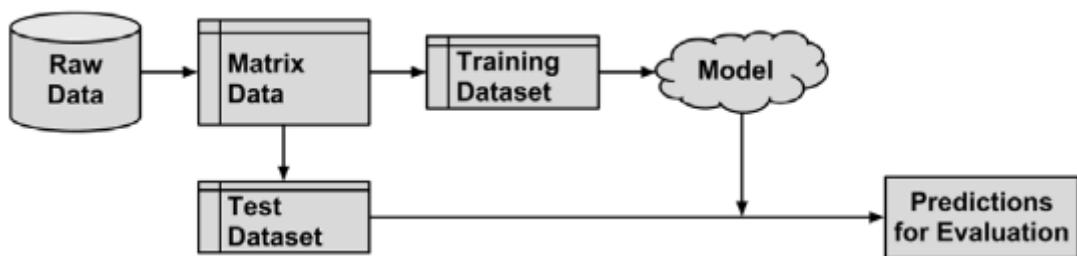


Figura 4. Ejemplo de utilización de un conjunto de test para evaluar el rendimiento de un modelo de aprendizaje supervisado.

El procedimiento de dividir los datos de los que se dispone entre conjuntos de entrenamiento y conjuntos de test, se conoce con el nombre de **hold-out**. En este procedimiento, el conjunto de entrenamiento se utiliza para crear el modelo que es evaluado utilizando el conjunto de test. Normalmente, se utiliza un 80 % de los datos para crear el conjunto de entrenamiento y un 20 % de los datos para generar conjunto de test. Para asegurarse que no existen diferencias sistemáticas entre los dos grupos es necesario obtener las instancias de forma aleatoria.

Es importante remarcar que para que este método sea riguroso, no se pueden utilizar observaciones o instancias del conjunto de test para crear el clasificador.

La repetición de la técnica de *hold-out* es la base para la técnica **de validación cruzada**. La técnica de validación cruzada es el estándar de la industria para estimar

el rendimiento de los modelos. Esta técnica también es conocida como ***k-fold***, donde *k* es un valor que indica que se han dividido los datos históricos en *k* particiones separadas llamadas *folds* o conjuntos.

Aunque *k* puede ser cualquier número, lo habitual es utilizar *k*=5 o *k*=10. Esto es debido a que la evidencia empírica indica que hay poco beneficio en utilizar más de 10 *folds*. En este caso, para cada uno de los 10 *folds* (que comprenden un 10 % del total de los datos) se crea un modelo en los 9 *folds* restantes (90 % de los datos) y se evalúa en ese 10 %. Este proceso se realiza 10 veces y la media y la desviación típica de las ejecuciones es reportada.

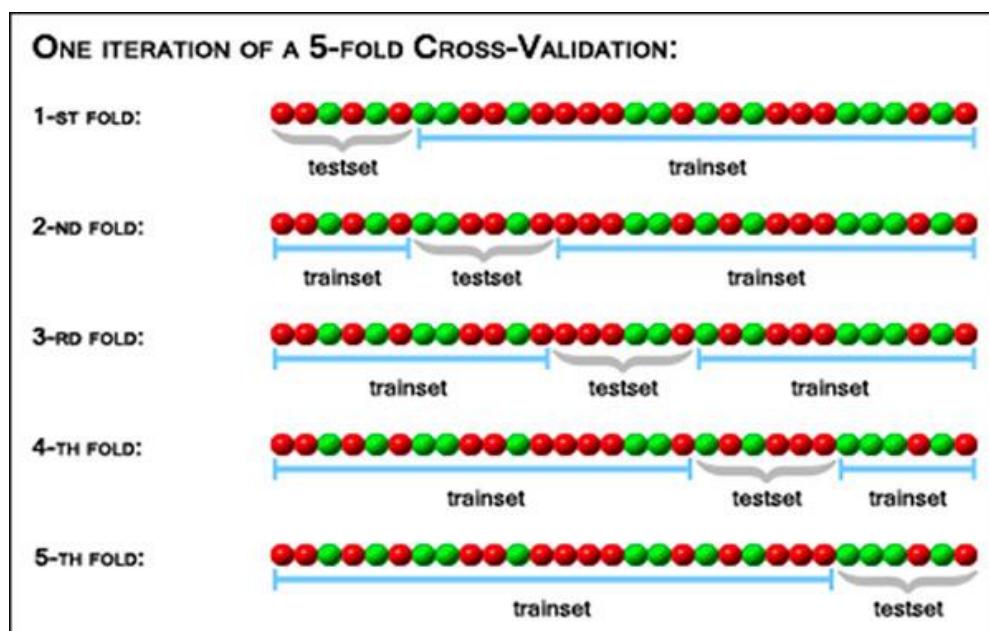


Figura 5. Ejemplo de división de un esquema de validación cruzada con 5 folds. Fuente:

<https://genome.tugraz.at/proclassify/help/pages/XV.html>

El conjunto de **entrenamiento** en aprendizaje supervisado proporciona un error que debe de ser evaluado con cautela. Es más riguroso evaluar a los modelos utilizando un conjunto separado e independiente llamado **conjunto de test**. Esta separación entre conjunto de test y entrenamiento se puede repetir a lo largo de los datos disponibles utilizando la técnica de **validación cruzada**.



Accede a los ejercicios de autoevaluación a través del aula virtual

1.4. Técnicas de agrupamiento



Accede al vídeo «Técnicas de agrupamiento y de detección de anomalías» a través del aula virtual

Las técnicas de agrupamiento o también conocidas como *clustering* son un ejemplo de **aprendizaje no supervisado** que se utilizan cuando desconocemos la etiqueta, clase o respuesta de las instancias de entrenamiento. En este tipo de problemas no se tiene conocimiento sobre las categorías o valores de los datos observados y se desea buscar la estructura oculta en los datos.

Por tanto, el proceso de modelado se lleva a cabo sobre un conjunto de ejemplos que **solo contiene las entradas del sistema**, y el algoritmo tiene que ser capaz de reconocer patrones para diferenciar entre los datos existentes. El resultado de estos algoritmos es una asignación de las observaciones a cada uno de los segmentos obtenidos.

La evaluación de este tipo de técnicas no es sencilla porque no se conoce el número de errores producidos. En concreto, cualquier método de evaluación sobre este tipo de técnicas no pueden calcularse con las etiquetas asignadas a las instancias sino con la separación que el algoritmo hace utilizando para ello métricas de similitud entre los miembros de cada una de las clases.

En la siguiente figura se muestra la separación entre clases que llevaría a cabo un algoritmo de agrupamiento para dos, tres y cuatro clases.

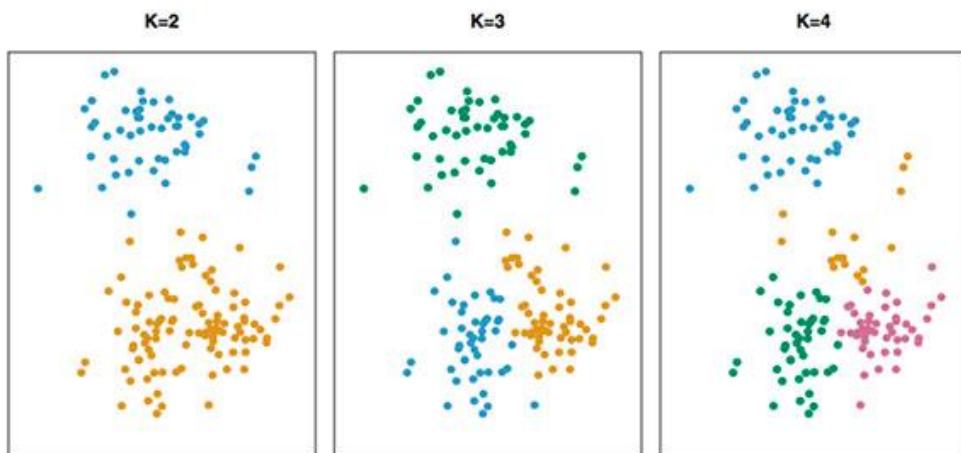


Figura 6. Ejemplos de agrupamiento para tres diferentes valores de k sobre los mismos datos. Fuente: James et al, 2013.

Las técnicas de agrupamiento son un ejemplo de aprendizaje no supervisado donde no se conocen las clases o valores de cada una de las instancias. El objetivo de estas técnicas es buscar la estructura oculta en los datos.

Técnicas de detección de anomalías

Una **anomalía** es una observación que es significativamente diferente del resto de observaciones. La **detección de anomalías** implica que esa observación es sospechosa de haber sido generada por un mecanismo diferente del resto de observaciones. La detección de anomalías es útil para dos objetivos diferentes. Por un lado, puede ser necesario eliminar estas anomalías de los datos para su posterior análisis. Por otro lado, el objetivo del análisis puede ser precisamente las anomalías y su origen.

Los algoritmos de detección de anomalías también son conocidos con el nombre de **detección de outliers**. Estos problemas tienen gran aplicación en entornos como el fraude, la detección de intrusos, la detección de defectos estructurales, problemas médicos o errores en texto.

Los algoritmos de detección de anomalías se pueden clasificar entre:

- ▶ Detección de anomalías supervisada.
- ▶ Detección de anomalías no supervisada.

La detección de anomalías de forma no supervisada busca anomalías en conjuntos de datos sin etiquetar con la hipótesis de que la mayoría de las instancias en el conjunto de datos son normales y observando aquellas que menos se parecen a la mayoría. Por otro lado, la detección de anomalías supervisada utiliza instancias o ejemplos previamente etiquetados como «normales» y «anormales» para entrenar un clasificador.

La detección de anomalías **busca observaciones** que son significativamente **diferentes** del resto de las observaciones. Esta detección puede hacerse utilizando **aprendizaje no supervisado**, si no conocemos ejemplos previos de anomalías. O bien utilizando **aprendizaje supervisado** para entrenar un clasificador que distinga entre ejemplos normales o anormales.



Accede al vídeo «Resumen del tema» a través del aula virtual



Accede a los ejercicios de autoevaluación a través del aula virtual

1.5. Referencias bibliográficas

James G., Witten, D., Hastie, T and Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Springer.

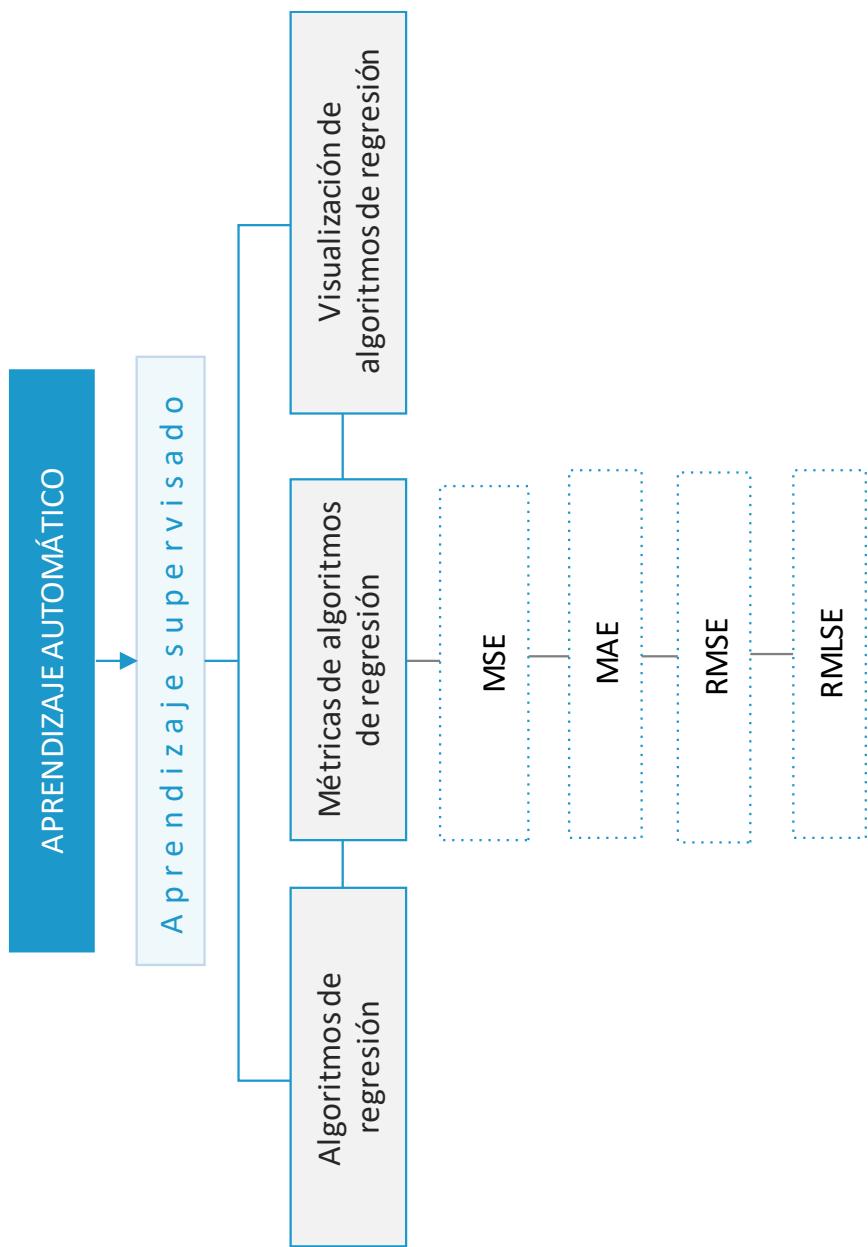
Aprendizaje Automático

Evaluación de algoritmos de regresión

Índice

Esquema	3
Ideas clave	4
2.1. Introducción y objetivos	4
2.2. Algoritmos de regresión	5
2.3. Métricas de error	10
2.4. Visualización de los errores	12

Esquema



2.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos: algoritmos de regresión» a través del aula virtual

Este tema profundiza en los algoritmos de aprendizaje supervisado, haciendo el foco en los **algoritmos de regresión** los cuales realizan una estimación o predicción de una variable numérica o cuantitativa.

Como ya se ha definido, el problema de regresión es aquel en el cual la variable a predecir y es una variable numérica o cuantitativa.

El objetivo de este tema es obtener una mayor comprensión sobre los algoritmos de regresión, utilizar las métricas de error más comunes y ser capaz de visualizar los errores de forma gráfica.

- ▶ En primer lugar, se describe el funcionamiento general de los algoritmos de regresión.
- ▶ A continuación, se describen las métricas comunes utilizadas por este tipo de algoritmos.
- ▶ Finalmente se describen visualización que permite obtener de forma gráfica una evaluación del funcionamiento de estos algoritmos.

2.2. Algoritmos de regresión



Accede al vídeo «Algoritmos de regresión» a través del aula virtual

Dentro del campo del aprendizaje automático, los algoritmos de regresión son un tipo concreto de **algoritmos de aprendizaje supervisado** y consisten en realizar una predicción de una variable numérica o cuantitativa. En el aprendizaje supervisado, para cada una de las observaciones de las variables predictoras (x_i) existe una medida de la variable respuesta (y_i). Se desea crear un modelo que relacione las variables y las respuestas con el objetivo de predecir las respuestas de observaciones en el futuro.

Existen numerosos problemas del mundo real donde la variable que se desea predecir o estimar es una variable numérica, por ejemplo, los ingresos de una persona, el precio de venta de un inmueble o algo tan dispar como la fuerza del cemento. De forma general, el aprendizaje supervisado se puede utilizar en todos aquellos casos en los cuales existe **conocimiento de un valor cuantitativo previo**.

La **terminología o notación más común** es la siguiente:

- ▶ Variable respuesta: también conocida como *outcome*, variable dependiente, variable objetivo, *target*, *class*, etc. Se suele denotar por y .
- ▶ Vector de p mediciones predictoras llamado x : también conocido como *inputs*, *regressors*, *features*, variables, variables independientes, etc.
- ▶ Se tienen datos de entrenamiento conocidos como *training data*: $(x_1, y_1), (x_N, y_N)$ que son observaciones, ejemplos o instancias de cada una de las medidas de entrada.

Para entender el funcionamiento de los algoritmos de regresión, observemos el siguiente gráfico:

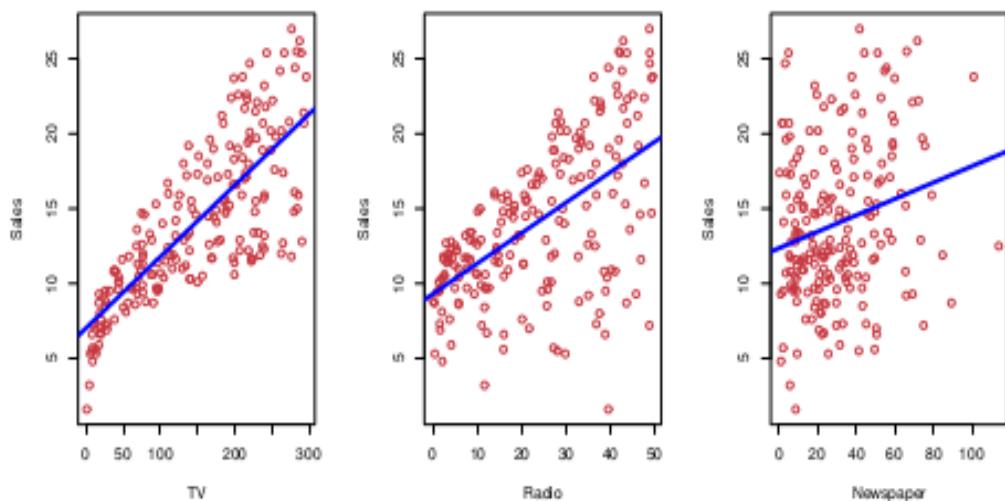


Figura 1. El gráfico muestra las ventas en unidades de miles en función el gasto en TV, radio y periódico en miles de dólares para 200 mercados diferentes. La línea azul representa un modelo simple que se puede utilizar para predecir las ventas utilizando cada una de estas variables más importantes. Fuente: James et al., 2013.

El gráfico anterior muestra la distribución de las ventas en función del gasto publicitario en TV, radio o prensa escrita. Es posible determinar cada uno de los valores del eje y en función del eje x utilizando cada una de las gráficas de forma independiente, o bien combinar las tres para obtener una función de las ventas obtenidas en función de las tres entradas.

$$Sales \approx f(TV, Radio, Newspaper)$$

Lo cual se puede definir utilizando notación vectorial de la siguiente forma:

$$X = \begin{pmatrix} X_1 \\ X_2 \\ X_3 \end{pmatrix} \quad Y = f(x) + \varepsilon$$

Supongamos que estamos en un plano de dos dimensiones. Se define como función $f(x)$ ideal a la curva que mejor aproxima los puntos.

Supongamos que tenemos una situación como la de la gráfica de abajo:

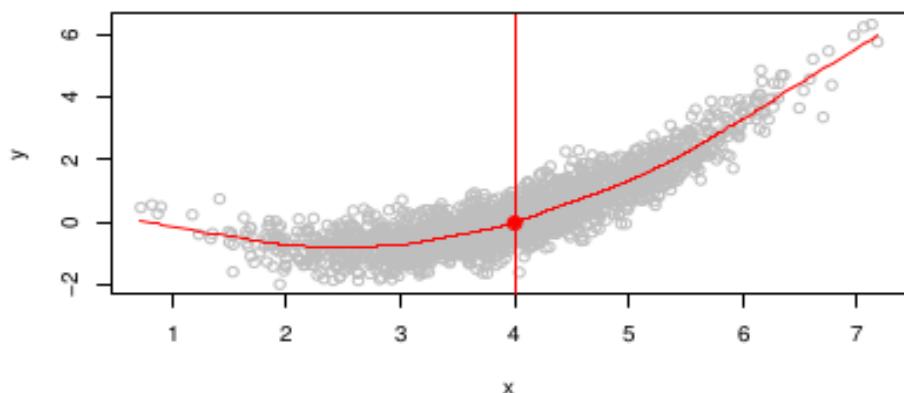


Figura 2. Función de regresión pintada en color rojo sobre la nube de puntos, donde se observa que existen diversos valores de y para un mismo valor del eje x . Fuente:
<https://lagunita.stanford.edu/courses/HumanitiesSciences/StatLearning/Winter2016/about>

Ahora, supongamos que queremos obtener el valor de y en el punto $x=4$, ¿Qué sucede? Pues que tenemos varios puntos con valores de y diferentes para el punto $x=4$. Este valor en el punto $x=4$ se define de la siguiente forma:

$$f(4) = E(Y|X = 4)$$

A la función $f(x)$ ideal también se la conoce con el nombre de función de regresión, la cual para todos los puntos del eje x se define de la siguiente forma:

$$F(x) = E(Y|X = x)$$

Esta función también se encuentra definida para un vector de x :

$$f(x) = f(x_1, x_2, x_3) = E(Y|X_1 = x_1, X_2 = x_2, X_3 = x_3)$$

La función predictora óptima de y es aquella con un error cuadrático medio (MSE) menor:

$$f(x) = E(Y|X = x)$$

Por tanto, que minimiza:

$$E \left[(Y - g(X))^2 \mid X = x \right]$$

Para todas las funciones g en todos los puntos $X=x$

Este cálculo se divide en dos términos distintos conocidos como error reducible y error irreducible:

$$E \left[(Y - g(X))^2 \mid X = x \right] E[Y - \hat{f}(X)]^2 \mid X = x = [f(x) - \hat{f}(x)]^2 + Var(\epsilon)$$

¿Cómo estimar el $f(x)$ ideal?

El problema es que normalmente tendremos pocos o ningún punto en $x=4$, por tanto no se puede calcular directamente $E(Y|X = x)$ y la solución es relajar la definición de $f(x)$ ideal, siendo por tanto necesario calcular: $\hat{f}(x) = Ave(Y|X \in N(x))$, donde $N(x)$ es un vecindario de puntos cercanos de x .

Un ejemplo ilustrativo de esta situación se puede observar en la siguiente gráfica:

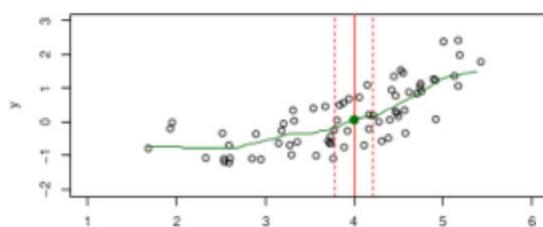


Figura 3. Estimación del valor de un punto en función del valor de sus vecinos (denotados por las líneas punteadas). Fuente:

<https://lagunita.stanford.edu/courses/HumanitiesSciences/StatLearning/Winter2016/about>

Esto es lo que se conoce como estimación *nearest-neighbor average*. Por lo general, el problema suele ser que estimar los puntos cercanos utilizando *nearest-neighbor average* funciona bien para problemas con pocas dimensiones (p pequeña) y muchos ejemplos (N muy grande), pues de lo contrario se produce el problema de la *curse of dimensionality*.

El problema de la *curse of dimensionality* implica que puntos cercanos en dimensiones pequeñas se van alejando de forma exponencial a medida que se incrementa el número de dimensiones.

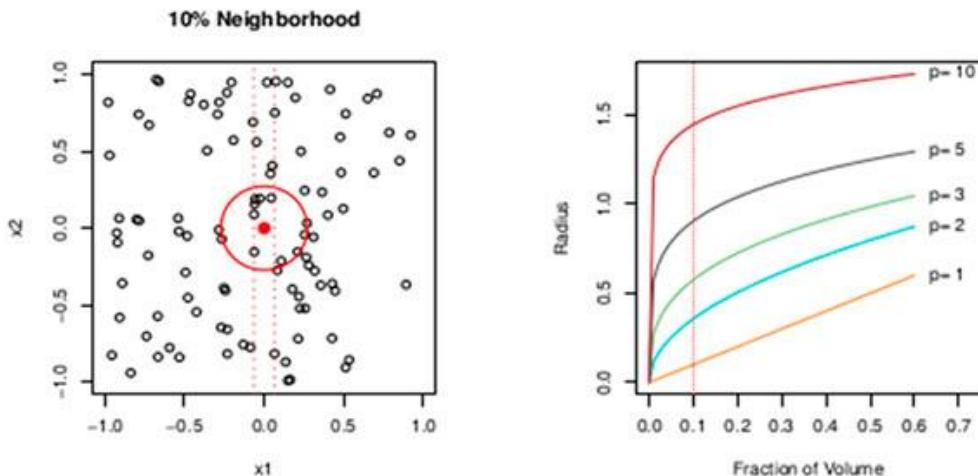


Figura 4. Efecto del problema de *curse of dimensionality*. Fuente:
<https://lagunita.stanford.edu/courses/HumanitiesSciences/StatLearning/Winter2016/about>

Este efecto se observa en la figura anterior: en el gráfico de la izquierda en una dimensión es posible cubrir el 10 % de los puntos con las líneas punteadas del eje x1. En el caso de dos dimensiones, el 10 % de los puntos se cubren con el área interior del círculo rojo. En el gráfico de la derecha se observa este efecto para los casos hasta diez dimensiones ($p=10$). El efecto consiste en que cada vez que se incrementa una dimensión para tener p dimensiones, los puntos que estaban cercanos en $p-1$ dimensiones pasan a estar más alejados.

En los algoritmos de regresión se utiliza una variable respuesta, un vector de p mediciones y datos de entrenamiento para construir una función de regresión. La función de regresión consta de un error reducible y un error irreducible. La estimación de la función $f(x)$ ideal puede conllevar los problemas de *curse of dimensionality*.



Accede a los ejercicios de autoevaluación a través del aula virtual

2.3. Métricas de error



Accede al vídeo «Métricas de error» a través del aula virtual

Existen diferentes métricas de error que se pueden aplicar en un problema de regresión para obtener la función $f(x)$ ideal. Las más comunes o habituales y sus definiciones matemáticas son:

- ▶ Error cuadrático medio, *mean square error (MSE)*: se define como la media de la diferencia entre el valor real y el valor predicho o estimado al cuadrado.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- ▶ Error absoluto medio, *mean absolute error (MAE)*: se define como la diferencia en valor absoluto entre el valor real y el valor predicho.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- ▶ Raíz del error cuadrático medio, *root mean square Error (RMSE)*: se define como la raíz cuadrada de la media de la diferencia entre el valor real y el valor predicho o estimado al cuadrado.

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

Esta métrica comparada con el error absoluto medio (MAE) amplifica y penaliza los errores grandes. Por otro lado, en MAE cada error contribuye al total del error

en función de su valor absoluto. El siguiente código muestra un ejemplo de cálculo en R:

```
RMSE <- sqrt(mean((y-y_pred)^2))
```

Por otro lado, el siguiente código muestra un ejemplo de cálculo en Python:

```
from sklearn.metrics import  
mean_squared_error  
RMSE = mean_squared_error(y,  
y_pred)**0.5
```

- ▶ Logaritmo de la raíz del error cuadrático medio, *root mean logarithmic square error (RMLSE)*:

$$RMLSE = \sqrt{1/n \sum_{i=1}^n (\log (y_i + 1) - \log (\hat{y}_i + 1))^2}$$

Esta métrica penaliza una *under-prediction* más que una *over-prediction*. El siguiente código de Python muestra un ejemplo de cómo calcular esta métrica.

```
import math  
  
def rmsle(y, y_pred):  
    assert len(y) == len(y_pred)  
    terms_to_sum = [(math.log(y_pred[i] + 1) -  
                    math.log(y[i] + 1)) ** 2.0 for i, pred in  
                    enumerate(y_pred)]  
    return (sum(terms_to_sum) * (1.0 / len(y))) ** 0.5
```

Las métricas más comunes de los problemas de regresión, son el error cuadrático Medio (**MSE**), error absoluto medio (**MAE**), raíz del error cuadrático medio (**RMSE**) y logaritmo de la raíz del error cuadrático medio (**RMLSE**).



Accede a los ejercicios de autoevaluación a través del aula virtual

2.4. Visualización de los errores



Accede al vídeo «Visualización de errores» a través del aula virtual

Una de las mejores formas de evaluar un modelo de regresión es utilizando gráficos. Uno de los gráficos que más información proporcionan visualmente es un **diagrama de dispersión de dos dimensiones** donde el eje x son los valores reales y el eje y los valores predichos o estimados (o viceversa). En el siguiente gráfico se muestra un ejemplo.

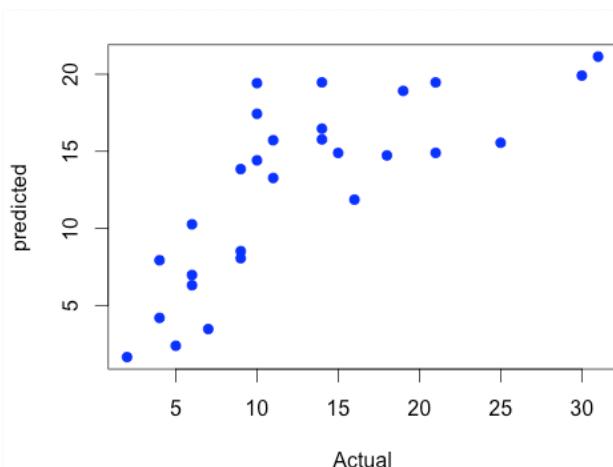


Figura 5. Gráfico de dispersión que muestra la relación y el error entre las predicciones y los valores reales.
Fuente: <https://stats.stackexchange.com/questions/104622/what-does-an-actual-vs-fitted-graph-tell-us>

Cada uno de los puntos corresponde a una de las estimaciones o predicciones realizadas. En el caso de que las **predicciones o estimaciones fueran perfectas**, caerían directamente sobre la diagonal del gráfico. Por otro lado, los puntos que caen por encima de la diagonal son **sobre estimaciones de las predicciones**, mientras que los puntos que caen por debajo de la diagonal son predicciones que se quedan cortas.

Una métrica común que se utiliza para medir la dispersión en este tipo de diagramas es el **coeficiente de correlación**, el cual cuantifica la relación lineal entre dos variables. Este coeficiente toma valores entre -1 y 1, donde 1 implica correlación lineal positiva de forma completa, -1 correlación lineal negativa de forma completa; y cuanto más cercano de 0 sea el valor menor correlación entre las dos variables.

Una correlación lineal positiva implica que cuando el valor en el eje *x* crece, el valor en el eje *y* lo hace en la misma proporción. Por el contrario, una correlación lineal negativa implica que cuando el valor en el eje *x* crece el valor en *y* decrece de forma proporcional. Esta relación se define matemáticamente con la siguiente función.

Esta bondad de ajuste también se puede medir utilizando el **coeficiente de determinación r^2** . Este coeficiente indica la fracción de la variación explicada por la recta de regresión respecto a la variación total. Coincide con el cuadrado del coeficiente de correlación y puede tomar valores entre 0 y 1, siendo 1 el mejor ajuste y 0 el peor. De forma matemática se define como:

Una de las mejores formas de visualización del resultado de un modelo de regresión es el obtenido por medio de un *scatter plot*, donde en el eje x aparezca la predicción y en el eje y el valor real.



Accede al vídeo «Resumen del tema» a través del aula virtual



Accede a los ejercicios de autoevaluación a través del aula virtual

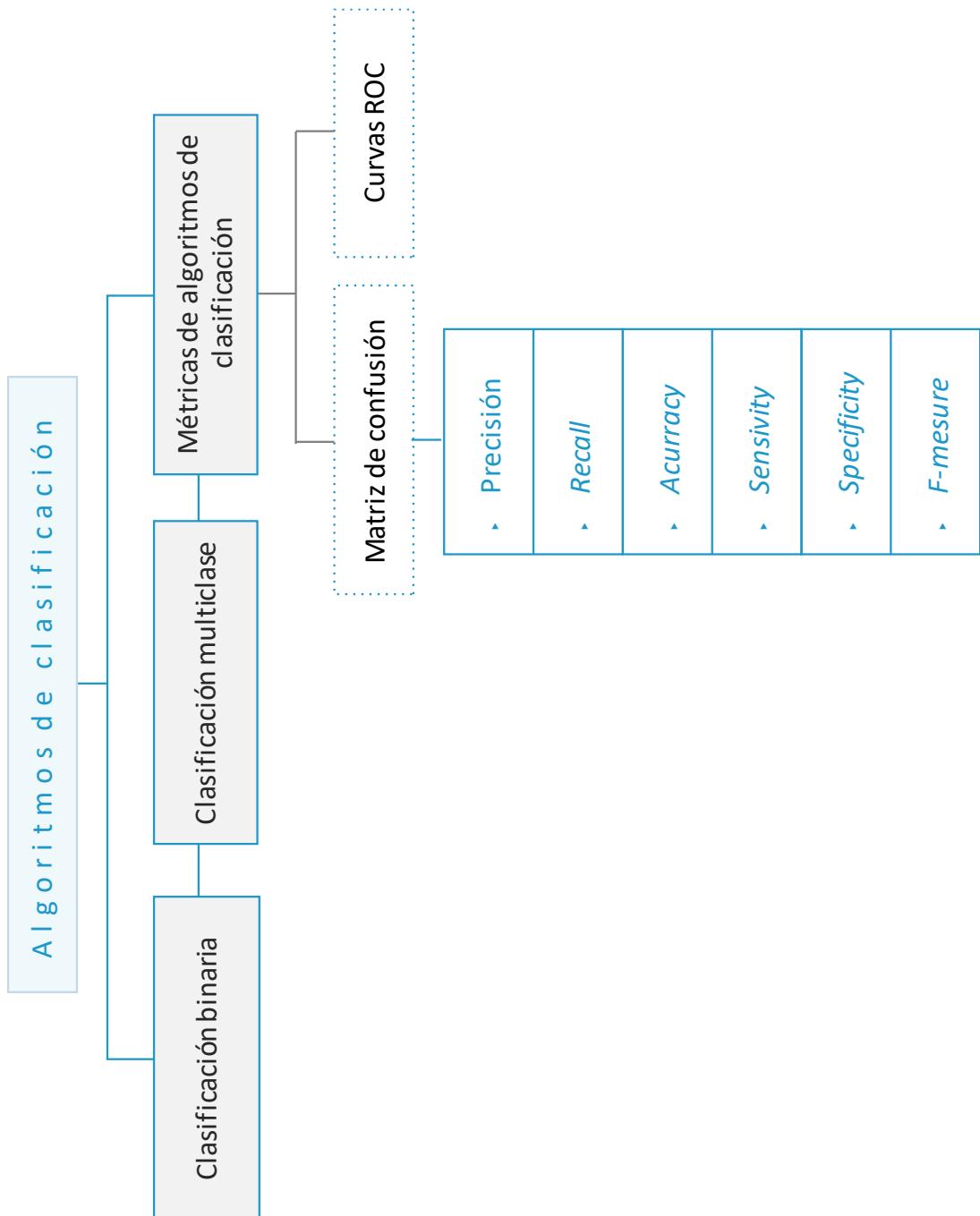
Aprendizaje Automático

Evaluación de algoritmos de clasificación

Índice

Esquema	3
Ideas clave	4
3.1. Introducción y objetivos	4
3.2. Algoritmos de clasificación	5
3.3. Métricas de evaluación: matriz de confusión	6
3.4. Métricas de evaluación: curvas ROC, AUC	14

Esquema



3.1. Introducción y objetivos

En este tema veremos los **algoritmos de clasificación**, los cuales son un tipo de algoritmos de aprendizaje supervisado donde la variable a predecir es categórica.

- ▶ En primer lugar, se verán las diferencias entre un problema de clasificación binaria y un problema de clasificación multiclas. Un ejemplo de clasificador binario sería un algoritmo para determinar la probabilidad de que un mensaje de correo electrónico sea *spam* y no *spam*; o por ejemplo un algoritmo para predecir el impago de un crédito bancario. Por otro lado, un ejemplo de clasificador multiclas puede ser un algoritmo que busque predecir en qué país de doce posibles un cliente va a realizar la siguiente reserva.

Este tipo de algoritmos de aprendizaje supervisado tienen la ventaja competitiva de **poder adelantarte a situaciones futuras** de la forma más adecuadas y gestionarlas mejor.

Cualquier evaluación corre el riesgo de ser subjetiva, por tanto, es necesario establecer los criterios de evaluación más objetivos y rigurosos posibles. Por tanto, a continuación, se describen las diferentes métricas de evaluación de los algoritmos de clasificación más utilizadas.

3.2. Algoritmos de clasificación



Accede al vídeo «Algoritmos de clasificación» a través del aula virtual

Dentro del aprendizaje supervisado, a continuación de los algoritmos de regresión, el siguiente gran grupo de algoritmos son los de clasificación. A diferencia de los algoritmos de regresión cuyo objetivo es predecir un valor numérico, los algoritmos de clasificación tienen como objetivo **obtener la clase más probable para cada una de las instancias**.

Este tipo de técnicas pueden utilizarse para predecir o estimar la probabilidad de pertenencia a una clase de entre dos posibles, lo que se conoce como **clasificación binaria**. Por otro lado, también se pueden utilizar para predecir o estimar la probabilidad de pertenencia de una clase de entre varias posibles (más de dos), lo cual se conoce como **clasificación multiclasa**.

A continuación, veremos las métricas de evaluación que se utilizan centrándonos en la matriz de confusión, precisión/recall, accuracy/sensitivity y f-measure. Posteriormente, veremos las **curvas ROC y la métrica de área bajo la curva**.

Dentro del **aprendizaje supervisado** los algoritmos de clasificación se utilizan para obtener la clase o categoría más probable de entre dos o más posibles.



Accede a los ejercicios de autoevaluación a través del aula virtual

3.3. Métricas de evaluación: matriz de confusión



Accede al vídeo «Métricas de evaluación: matriz de confusión» a través del aula virtual

La mejor métrica de rendimiento de un algoritmo de clasificación es ver si el clasificador tiene **éxito para su propósito**. Para evaluar un clasificador se pueden utilizar los valores predichos de las clases, los valores reales de las clases o bien la probabilidad estimada de la predicción.

En la práctica lo habitual es **mantener dos vectores de datos**. Por un lado, un vector que contiene la verdad o los valores observados y otro vector que contiene los valores predichos o estimados. Es importante resaltar que ambos vectores deben tener el mismo tamaño y los datos en el mismo orden. En este tipo de algoritmos de aprendizaje supervisado la clase verdadera se conoce de antemano y esta variable se utiliza para evaluar el algoritmo en el conjunto de test comparándola con el resultado predicho.

Es decir, los valores predichos **se obtienen por medio de la aplicación de un modelo sobre el conjunto de test**, el cual ha sido entrenado previamente con datos de entrenamiento.

En la mayoría de los paquetes del lenguaje *R*, esto se realiza por medio de invocar a la función `predict()` sobre un objeto que es el resultado de un modelo entrenado previamente y sobre un conjunto o *data.frame* de test. Ejemplo:

```
pred_valores <- predict(my_model test_data)
```

La función `predict()` tiene parámetros para indicar el tipo de predicción, normalmente se utiliza el valor `prob` para obtener la probabilidad de pertenencia a cada una de las clases.

Incluso en el caso de los clasificadores binarios, que realizan la predicción de una clase en función de dos posibles categorías, es muy útil conocer con qué certeza o confianza se predice cada una de las clases.

Por ejemplo, un mensaje de correo electrónico puede ser predicho como *spam* con una probabilidad de 0,9 o de 0,59, etc. De forma general si dos algoritmos de clasificación diferentes producen los mismos errores, pero uno de ellos es más capaz de tener en cuenta la incertidumbre, sería un mejor modelo. Por tanto, para evaluar correctamente el resultado de un clasificador es necesario **considerar el valor de probabilidad obtenido** en lugar de utilizar únicamente la clase más probable.

En resumen, el objetivo es obtener modelos que tienen mucha confianza en las predicciones correctas y sean temerosos en las predicciones dudosas. Este **balance entre confianza y prudencia** es la clave de la evaluación de modelos.

Ejemplo. Clasificación binaria

En la siguiente tabla se muestra la distribución de probabilidad obtenida para cada una de las dos clases en un problema de clasificación binaria. En este caso es importante tener claro que indica la clase *yes* y la clase *no*, por ejemplo, la clase *yes* puede indicar que el cliente ha realizado un impago de su crédito o por el contrario que el cliente ha realizado un pago con éxito del crédito. Por regla general, a la clase positiva (*yes* o 1 en este caso) se hace corresponder con el evento interesante a predecir.

Es importante tener claro la definición de cada una de las clases para no obtener conclusiones contradictorias.

```

> head(predicted_prob)
      no      yes
1 0.0808272 0.9191728
2 1.0000000 0.0000000
3 0.7064238 0.2935762
4 0.1962657 0.8037343
5 0.8249874 0.1750126
6 1.0000000 0.0000000

```

Figura 1. Fuente: Brett, 2013.

Los valores anteriores son el **resultado de la predicción**, pero al tratarse de una clasificación binaria se puede obtener solo uno de ellos, pues la suma de los dos valores debe ser **1**. A partir de esta probabilidad y aplicando un umbral o punto de corte a partir del cual se establece que la predicción es de una clase o de otra, se pueden realizar las evaluaciones correspondientes.

Por ejemplo, en la siguiente tabla se muestran los resultados obtenidos de un modelo de clasificación binaria donde la variable a predecir es si un correo electrónico es *spam* o no. El primer ejemplo se ha predicho como *ham* puesto que tiene una probabilidad de *spam* muy baja y en realidad era *ham*. El quinto ejemplo se ha predicho como *spam* con una certeza muy alta pues tiene una probabilidad de 1 y en realidad era *spam*.

```

> head(sms_results)
  actual_type predict_type    prob_spam
1        ham         ham 2.560231e-07
2        ham         ham 1.309835e-04
3        ham         ham 8.089713e-05
4        ham         ham 1.396505e-04
5      spam         spam 1.000000e+00
6        ham         ham 3.504181e-03

```

Figura 2. Fuente: Brett, 2013.

En el ejemplo anterior, cuando los valores predichos son de la clase *ham*, los valores de *spam*, son muy cercanos a 0 y por el contrario cuando los valores predichos son *spam* este valor es 1. Este comportamiento sugiere que el modelo tiene mucha confianza en sus clasificaciones.

En cualquier caso, lo habitual es que los **clasificadores o los modelos tengan errores** y sus predicciones difieran del valor real, como por ejemplo en la siguiente tabla donde todas las predicciones han sido *ham* y el valor real era *spam*.

```
> head(subset(sms_results, actual_type != predict_type))
   actual_type predict_type    prob_spam
1      spam        ham 0.0006796225
2      spam        ham 0.1333961018
3      spam        ham 0.3582665350
4      spam        ham 0.1224625535
5      spam        ham 0.0224863219
6     184        ham 0.0320059616
```

Figura 3. Fuente: Brett, 2013.

Sin embargo, en algunos casos las probabilidades con las que se ha asignado la clase predicha no son muy extremas. Por ejemplo, la instancia 73 de la tabla anterior ha sido asignada a la clase *ham*, pero tenía una probabilidad de 0,35 o 35 % de ser *spam*.

Estos ejemplos muestran errores puntuales o particulares, pero para saber si un clasificador es útil se deben tener en cuenta métricas más completas sobre el conjunto de datos completo.

En el caso de los algoritmos de clasificación una métrica muy común es la **matriz de confusión**. Una matriz de confusión es una tabla que organiza las predicciones en función de los valores reales de los datos. Una de las dimensiones de la tabla hace referencia a la categoría de los valores predichos, la otra dimensión a las categorías reales.

Las instancias clasificadas correctamente caen en la diagonal de la matriz. Los valores fuera de la diagonal indican las instancias clasificadas incorrectamente, se trata de predicciones incorrectas. Las métricas de rendimiento obtenidas con la matriz de confusión se basan en cuantas instancias caen dentro y fuera de la diagonal. La mayoría de las métricas de rendimiento consideran la capacidad que tiene un clasificador de discernir una categoría respecto de las demás. La categoría de interés se conoce como **clase positiva**, mientras que las otras categorías se conocen como **clase negativa**.

En la siguiente imagen se muestra cómo se obtienen los cuatro valores de una matriz de confusión para el caso de una clasificación binaria.

		Condition (as determined by "Gold standard")		
		Condition Positive	Condition Negative	
Test Outcome	Test Outcome Positive	True Positive	False Positive (Type I error)	Positive predictive value = $\frac{\sum \text{True Positive}}{\sum \text{Test Outcome Positive}}$
	Test Outcome Negative	False Negative (Type II error)	True Negative	Negative predictive value = $\frac{\sum \text{True Negative}}{\sum \text{Test Outcome Negative}}$
		Sensitivity = $\frac{\sum \text{True Positive}}{\sum \text{Condition Positive}}$	Specificity = $\frac{\sum \text{True Negative}}{\sum \text{Condition Negative}}$	

Tabla 1. Matriz de confusión.

Fuente: https://en.wikipedia.org/wiki/Confusion_matrix

Cada uno de estos cuatro valores se pueden **definir** de la siguiente forma:

1. Tasa de verdaderos positivos (*true positives*): se trata de las clasificaciones correctas de las instancias que corresponden a la clase positiva.
2. Tasa de falsos positivos (*false positives*): se trata de las clasificaciones de la clase negativa que han sido incorrectamente clasificadas como clase positiva.
3. Tasa de verdaderos negativos (*true negatives*): se trata de las clasificaciones correctas de las instancias que corresponden a la clase negativa.

4. Tasa de falsos negativos (*false negatives*): se trata de las clasificaciones de la clase positiva que han sido incorrectamente clasificadas como clase negativa.

Matriz de confusión en R

Una forma sencilla de obtener la matriz de confusión utilizando el lenguaje de programación *R* es por medio de la función `table()`, la cual devuelve las ocurrencias de cada combinación de valores, por ejemplo:

```
> table(sms_results$actual_type, sms_results$predict_type)
    ham  spam
ham   1202     5
spam    29   154
```

Figura 4. Fuente: Brett, 2013.

Para obtener directamente las probabilidades se puede utilizar:

```
prop.table(table(sms_results$actual_type, sms_results$predict_type),
           margin = 2)
```

Por medio de la matriz de confusión se pueden obtener también las siguientes métricas:

- ▶ *Accuracy*: esta métrica también se conoce como el ratio de éxito. Representa la proporción del número de predicciones correctas entre el número total de predicciones y se define así:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- ▶ *Sensitivity/Specificity* (sensibilidad/especificidad): una clasificación siempre conlleva un balance entre ser conservador y agresivo. Por ejemplo, un sistema de

clasificación de *spam* podría eliminar todos los mensajes de *spam* a costa de eliminar también los mensajes *ham*.

Por otro lado, es necesario una garantía de que ningún mensaje *ham* sea clasificado como *spam*. Este balance se captura con las métricas de sensibilidad y especificidad.

La sensibilidad de un modelo es el ratio de los ejemplos positivos correctamente clasificados. La especificidad de un modelo indica la proporción de los ejemplos negativos correctamente clasificados.

Estas métricas tienen valores de 0 a 1, siendo 1 lo más deseable y se definen así:

$$\text{especificidad} = \frac{TN}{TN + FP}$$

$$\text{sensibilidad} = \frac{TP}{TP + FN}$$

- ▶ *Precision/recall*: estas métricas hacen referencia a los compromisos hechos en la clasificación. Se utilizan mucho en el campo de *information retrieval* e indican lo interesantes y relevantes que son los resultados de un modelo.

La **métrica de precisión** también se conoce como *positive predictive value* (PPV) e indica la proporción de ejemplos que son verdaderamente positivos. Es decir, cuando el modelo predice la clase positiva, ¿cuántas veces está en lo cierto? Por otro lado, la métrica de *recall* indica que tan completos son los resultados. La métrica de *recall* equivale a la métrica de sensibilidad. Un modelo con gran *recall* captura un gran porcentaje de ejemplos positivos. La definición de ambas métricas es la siguiente:

$$\text{precision} = \frac{TP}{TP + FP}$$

$$recall = \frac{TP}{TP + FN}$$

- *F-measure*: también conocida como *F1*, es una métrica que combina precisión y *recall* utilizando la media armónica. Se utiliza la media armónica porque los valores indican proporciones entre 0 y 1. Se utiliza con mucha frecuencia puesto que simplifica el rendimiento de un algoritmo de clasificación a una única métrica. De forma matemática se define así:

$$F1 = \frac{2 * precision * recall}{recall + precision} = \frac{2 * TP}{2 * TP + FP + FN}$$

La métrica por defecto asume que precisión y *recall* tienen la misma importancia, pero es posible ponderarlos para darle mayor peso a uno u otro

A la hora de evaluar los algoritmos de clasificación es importante diferenciar entre una clasificación binaria o multiclase. Una de las métricas más comunes es la matriz de confusión que nos proporciona información acerca de los aciertos y errores de cada una de las clases. A partir de la matriz de confusión se pueden obtener métricas como *accuracy*, *sensitivity/specifity*, *precisión/recall* y *f-measure*.



Accede a los ejercicios de autoevaluación a través del aula virtual

3.4. Métricas de evaluación: curvas ROC, AUC



Accede al vídeo «Métricas de evaluación: curvas ROC, AUC» a través del aula virtual

La clave de cualquier modelo de aprendizaje automático es su **capacidad de generalizar situaciones del futuro** en función de los datos históricos observados. Las métricas anteriores que se obtienen a partir de la matriz de confusión conllevan que se establece un punto de corte determinado sobre la distribución de probabilidad para determinar si una observación es clasificada como una clase determinada. Es decir, por ejemplo, aquellos valores por encima de 0,5 se les asigna la clase positiva (1) y aquellos valores iguales o por debajo de 0,5 la clase negativa (0).

En el caso de los algoritmos de clasificación binaria, se puede utilizar una métrica obtenida a partir de las curvas *receiver operating characteristic (ROC)* conocida como *area under the curve (AUC)* o en castellano área bajo la curva. Esta métrica se utiliza para determinar el balance entre la detección de verdaderos positivos y evitar los falsos positivos. Para ello, se muestra la proporción de detección de los verdaderos positivos en el eje vertical y la proporción de los falsos positivos en el eje horizontal, para un umbral o punto de corte determinado.

Los puntos a lo largo de la curva indican el ratio de los verdaderos positivos a medida que se incrementan los valores de los falsos positivos. En la siguiente imagen se muestra un ejemplo de cuatro curvas ROC distintas.

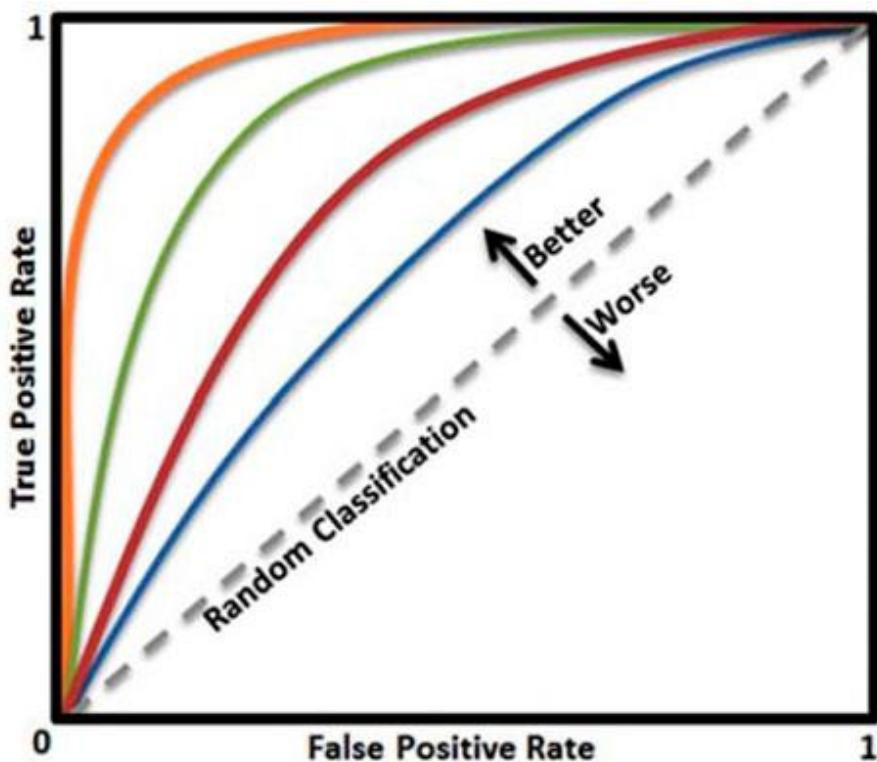


Figura 5. Ejemplo de una curva ROC.

Fuente: https://openi.nlm.nih.gov/imgs/512/261/3861891/PMC3861891_CG-14-397_F10.png

A medida que la curva este más cercana de la esquina superior izquierda, mejor será. En el ejemplo, el clasificador de la curva naranja es mejor que el verde, y este mejor que el rojo y el azul. Una clasificación completamente aleatoria coincide con la línea punteada.

A partir de estas curvas se puede obtener el **área bajo la curva**. Esta métrica va desde valores de 0,5 para un clasificador sin potencia predictiva (completamente aleatorio) hasta 1 para un clasificador perfecto. Cuanto más cercanos están los resultados del clasificador perfecto, mejor.

Es posible que para un mismo valor de AUC haya diferentes curvas ROC, por lo que suele ser buena práctica mostrarlas de forma gráfica.

Visualizando el rendimiento en R

El paquete **ROCR** (<https://rocr.bioinf.mpi-sb.mpg.de/>) proporciona una serie de funciones para visualizar y entender el rendimiento de un clasificador.

Para crear visualizaciones con ROCR se necesitan dos vectores: uno con la clase real de los valores predichos y otro con la probabilidad estimada de la clase positiva. Estos valores se utilizan en la función `prediction()` que devuelve un objeto que se puede utilizar para obtener métricas o visualizar el resultado. El siguiente código de ejemplo muestra el uso:

```
library(ROCR)
pred      <- prediction(predictions      =
pred_results$prob,
                           labels = pred_results$actual_value)
#Curva ROC
perf <- performance(pred, "tpr", "fpr")
plot(perf)

#Curva Precision-Recall
perf2 <- performance(pred, "prec", "rec")
plot(perf2)

#Curva sensivity-specificity
perf3 <- performance(pred, "sens", "spec")
plot(perf3)
```



Accede al vídeo «Resumen del tema» a través del aula virtual



Accede a los ejercicios de autoevaluación a través del aula virtual

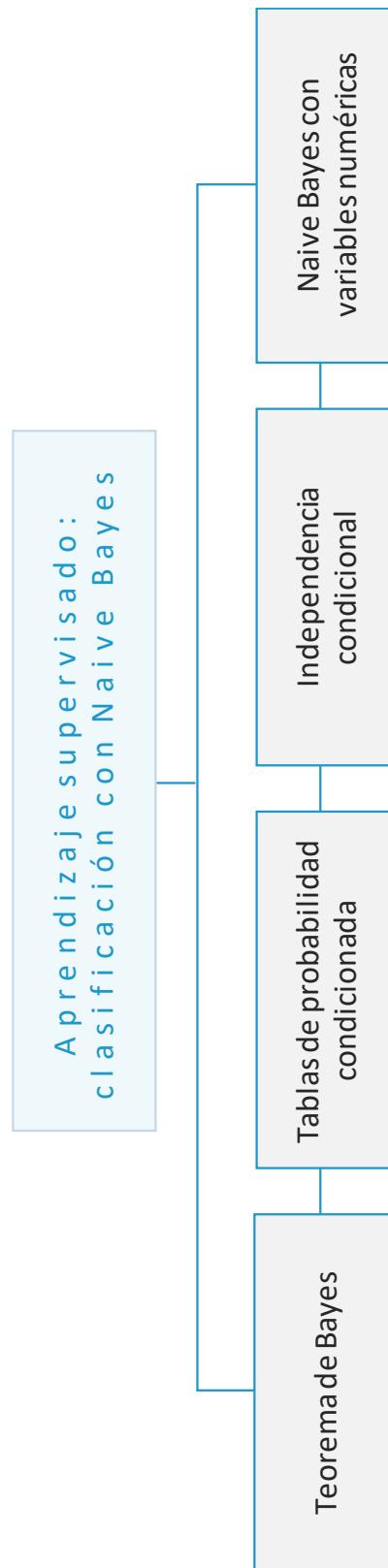
Aprendizaje Automático

Aprendizaje supervisado: clasificación con Naive Bayes

Índice

Esquema	3
Ideas clave	4
4.1. Introducción y objetivos	4
4.2. Teorema de Bayes y tablas de probabilidad	5
4.3. Independencia condicional en el clasificador	
Naive Bayes	10
4.4. Clasificador Naive Bayes	12
4.5. Referencias bibliográficas	15

Esquema



4.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos: clasificador Naive Bayes» a través del aula virtual

E

n este tema se va a introducir el clasificador Naive Bayes, el cual está basado en el teorema de Bayes para calcular las probabilidades *a posteriori* de los eventos a predecir.

Vamos a ver los siguientes puntos:

- ▶ En primer lugar, veremos el teorema de Bayes.
- ▶ A continuación, se describe la forma de calcular las tablas de probabilidad condicionada.
- ▶ Posteriormente se verá por qué es importante asumir independencia condicional en el clasificador Naive Bayes.
- ▶ El clasificador Naive Bayes y finalmente cómo se puede utilizar con variables numéricas.



Accede a los ejercicios de autoevaluación a través del aula virtual

4.2. Teorema de Bayes y tablas de probabilidad

Teorema de Bayes



Accede al vídeo «Teorema de Bayes y tablas de probabilidad» a través del aula virtual

El Teorema de Bayes es una proposición planteada por el filósofo inglés Thomas Bayes (1702-1761) en el año 1763 en su artículo «An Essay towards solving a Problem in the Doctrine of Chances» publicado en la revista *Philosophical Transactions of the Royal Society of London*.

Este teorema expresa la **probabilidad condicional** de un evento aleatorio A dado B en términos de la distribución de probabilidad condicional del evento B dado A y la distribución de probabilidad marginal de solo A.



Foto: Reverendo Thomas Bayes.

Fuente: https://upload.wikimedia.org/wikipedia/commons/d/d4/Thomas_Bayes.gif

El teorema de Bayes es bastante relevante porque **relaciona la probabilidad de dos eventos A y B utilizando la dependencia condicional de uno de ellos**. Es decir, relaciona la probabilidad de que ocurra el evento A y sabemos de antemano que ha

ocurrido B, utilizando la probabilidad de que ocurra el evento B sabiendo que ha ocurrido A.

Este teorema **permite relacionar, entre otras cosas, síntomas y enfermedades**. Por ejemplo, sabiendo la probabilidad de tener dolor de garganta dado que se conoce que se tiene gripe, se puede obtener la probabilidad de tener gripe dado que se tiene dolor de garganta.

El teorema de Bayes **relaciona la comprensión de la probabilidad de aspectos causa-efecto dados los eventos dependientes observados**. Un evento dependiente es aquel cuyo resultado se ve afectado por el resultado de otro evento o serie de eventos. Los **eventos dependientes** son la base del modelado predictivo puesto que se busca obtener la probabilidad de que ocurra un suceso teniendo en cuenta la existencia de una serie de eventos dependientes.

En el caso de dos eventos dependientes A y B, el teorema de Bayes describe su relación de la siguiente manera:

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Es decir, la probabilidad de A dado que ocurre B, es igual a la probabilidad de que ocurra B dado que ha ocurrido A, multiplicado por la probabilidad de que ocurra A, dividido por la probabilidad de que ocurra B.

Este teorema se puede **generalizar para más de dos eventos** de la siguiente forma: en primer lugar, se entiende por evento mutuamente excluyente cuando dos resultados diferentes de un evento no pueden ocurrir al mismo tiempo. En el caso además de que los eventos sean exhaustivos, por lo menos uno de ellos tiene que ocurrir.

De forma matemática, se puede definir el teorema de Bayes para n eventos: Sea $\{A_1, A_2, \dots, A_i, \dots, A_n\}$ un conjunto de sucesos mutuamente excluyentes y exhaustivos, y tales que la probabilidad de cada uno de ellos es distinta de cero. Sea B un suceso cualquiera del que se conocen las probabilidades condicionales $P(B|A_i)$. Entonces, la probabilidad $P(A_i|B)$ viene dada por la expresión:

$$P(A_i|B) = \frac{P(B|A_i) * P(A_i)}{P(B)}$$

Donde:

- ▶ $P(A_i)$ son las probabilidades *a priori*.
- ▶ $P(B|A_i)$ es la probabilidad de B en la hipótesis A_i .
- ▶ $P(A_i|B)$ son las probabilidades *a posteriori*.
- ▶ $P(B)$ es la verosimilitud marginal.

Ejemplo:

Supongamos que deseamos estimar la probabilidad de que un mensaje de correo electrónico sea *spam*. Sin tener evidencias adicionales y, únicamente, por los mensajes previos obtenidos la probabilidad *a priori* de que un mensaje sea *spam* es 0,2.

Ahora bien, si tenemos evidencia de que el mensaje contiene la palabra Viagra, por medio de conocer la probabilidad de que la palabra Viagra haya sido utilizada en mensajes de *spam* previos (lo cual se conoce como verosimilitud o *likelihood*) y por medio de la probabilidad de que la palabra Viagra aparezca en cualquier mensaje, lo que se conoce como verosimilitud marginal. Aplicando el teorema de Bayes se puede calcular la probabilidad *a posteriori* y si esta es mayor que 0,5 es más probable que el mensaje sea *spam*. En la siguiente formula se muestra el cálculo anteriormente descrito.

$$P(Spam_i|Viagra) = \frac{P(Viagra|Spam) * P(Spam)}{P(Viagra)}$$

El **teorema de Bayes** relaciona la probabilidad de dos o más eventos utilizando la dependencia condicional de cada uno de ellos. Los eventos deben ser dependientes y mutuamente excluyentes.

Tablas de probabilidad condicionada

Para obtener cada uno de los componentes de las formulas anteriores es necesario construir una **tabla de frecuencias** que indica el número de veces que el evento aparece en cada una de las situaciones. En nuestro ejemplo de *spam*, es necesario calcular el número de veces que la palabra Viagra ha aparecido en los mensajes de *spam*. Esta tabla de frecuencias se utiliza posteriormente para calcular las tablas de verosimilitud o de probabilidad condicionada.

Siguiendo con el ejemplo de los mensajes de *spam*, en el caso hipotético de que tuviéramos la siguiente distribución histórica de 100 mensajes para la palabra Viagra...

Frecuencia	Si	No	Total
<i>Spam</i>	4	16	20
<i>Ham</i>	1	79	80
Total	5	95	100

Tabla 1. Ejemplo de distribución histórica de mensajes *spam* y *ham* para la palabra Viagra.

...Obtendríamos la correspondiente tabla de verosimilitud:

Verosimilitud	Si	No	Total
<i>Spam</i>	4/20	16/20	20
<i>Ham</i>	1/80	79/80	80
Total	5/100	95/100	100

Tabla 2. Ejemplo de tabla de verosimilitud para mensajes *spam* y *ham* en función de la palabra Viagra.

Con estos datos, para calcular la probabilidad *a posteriori* de que un mensaje sea *spam* dado que nos ha llegado la palabra Viagra, tendríamos que hacer el siguiente cálculo:

$$P(\text{Spam}/\text{Viagra}) = [(4/20) * (20/100)] / (5/100) = 0.8$$

Es decir, con los datos anteriores, la probabilidad de que un correo electrónico que contenga la palabra Viagra sea *spam* es del 0,8.

Ahora supongamos que deseamos añadir a este cálculo otros términos más comunes que aparecen en los mensajes *spam*, como pueden ser: *money*, *groceries* y *unsubscribe*.

En este caso, tendríamos la siguiente tabla de verosimilitud:

	Viagra (W1)		Money (W2)		Groceries (W3)		Unsubscribe (W4)		
Verosimilitud	Si	No	Si	No	Si	No	Si	No	Total
<i>Spam</i>	4/20	16/20	10/20	10/20	0/20	20/20	12/20	8/20	20
<i>Ham</i>	1/80	79/80	14/80	66/80	8/80	71/80	23/80	57/80	80
Total	5/10	95/10	24/10	76/10	8/100	91/100	35/100	65/100	100

Tabla 3. Ejemplo de tabla de verosimilitud para los mensajes *spam* y *ham* en función de las palabras Viagra, *money*, *groceries* y *unsubscribe*.

De esta forma, si llega un nuevo mensaje que contiene las palabras Viagra y *unsubscribe*, pero no *money* ni *groceries*; utilizando el teorema de Bayes habría que calcular la siguiente formula:

$$P(\text{Spam}|\text{Viagra}) = \frac{P(\text{Viagra}|\text{Spam}) * P(\text{Spam})}{P(\text{Viagra})}$$

El cálculo de la fórmula anterior es **computacionalmente costoso**, puesto que a medida que se añaden nuevos términos son necesarias grandes cantidades de memoria para almacenar todas las combinaciones.



Accede a los ejercicios de autoevaluación a través del aula virtual

4.3. Independencia condicional en el clasificador Naive Bayes



Accede al vídeo «Independencia condicional y clasificador con variables numéricas» a través del aula virtual

Debido a que el cálculo riguroso de la fórmula del teorema de Bayes, como en el ejemplo anterior, es computacionalmente costoso, el clasificador Naive Bayes se basa en una **modificación sencilla**. Básicamente asume **independencia condicional** entre los eventos, a pesar de que si todos los eventos fueran independientes sería imposible predecir ningún evento con los datos observados por otro. Formalmente, **dos eventos son independientes** si el resultado del segundo evento no es afectado por el resultado del primer evento. Si A y B son eventos independientes, la probabilidad de que ambos eventos ocurran es el producto de las probabilidades de los eventos individuales.

Por otro lado, los **eventos dependientes son la base del modelado predictivo**, puesto que permiten predecir la presencia de un evento en función de otro. Por ejemplo, la presencia de nubes suele ser un evento predictivo de un día lluvioso, o la presencia de la palabra *viagra* en un correo electrónico suele ser un evento predictivo de *spam*.

No obstante, al no poder asumir dependencia condicional por el alto coste computacional, el clasificador Naive Bayes asume **independencia condicional** entre los eventos condicionados al mismo valor de la clase. Este hecho es el que le ha dado el adjetivo de *naive* al clasificador.

En nuestro ejemplo anterior, asumiendo independencia condicional de las palabras para obtener la probabilidad de *spam*, tendríamos la siguiente formula:

$$\begin{aligned} P(Spam|W_1 \cap \neg W_2 \cap \neg W_3 \cap W_4) \\ = \frac{P(W_1|Spam)P(\neg W_2|Spam)P(\neg W_3|Spam)P(W_4|Spam) * P(Spam)}{P(W_1)P(\neg W_2)P(\neg W_3)P(W_4)} \\ = (4/20) * (10/20) * (20/20) * (12/20) * (20/100) = 0.012 \end{aligned}$$

Por otro lado, para obtener la probabilidad de *ham*, tendríamos:

$$\begin{aligned} P(Ham|W_1 \cap \neg W_2 \cap \neg W_3 \cap W_4) \\ = \frac{P(W_1|Ham)P(\neg W_2|Ham)P(\neg W_3|Ham)P(W_4|Ham) * P(Ham)}{P(W_1)P(\neg W_2)P(\neg W_3)P(W_4)} \\ = (1/80) * (66/80) * (71/80) * (23/80) * (80/100) = 0.002 \end{aligned}$$

Como $0,012/0,002 = 6$, se puede afirmar que es 6 veces más probable que el mensaje sea *spam* que *ham*.

Si queremos calcular la probabilidad de que el mensaje sea *spam*, sería igual a la verosimilitud de que el mensaje sea *spam* dividido por la verosimilitud de que sea *spam* o *ham*: $0,012 / (0,012 + 0,002) = 0,857$

Análogamente, la probabilidad de ser *ham* es: $0,002 / (0,002 + 0,012) = 0,143$

Por tanto, podemos estimar que, dadas las palabras del mensaje, hay una probabilidad de 0,857 de que sea *spam* y de 0,143 de que sea *ham* y como son eventos mutuamente excluyentes suman 1.



Accede a los ejercicios de autoevaluación a través del aula virtual

4.4. Clasificador Naive Bayes



Accede al vídeo «Clasificador Naive Bayes» a través del aula virtual

Como hemos comentado previamente, el teorema de Bayes es la **base** para el clasificador Naive Bayes. Este clasificador utiliza las probabilidades *a priori* de los eventos para estimar la **probabilidad de eventos futuros** por medio del teorema de Bayes.

Este clasificador **utiliza datos históricos o de entrenamiento** para calcular la probabilidad observada de cada evento en función de su vector de características. Para realizar una predicción, el clasificador es utilizado con datos que tienen la clase desconocida y se utilizan las probabilidades observadas para estimar la clase más probable.

La fórmula general del clasificador Naive Bayes se puede definir de la siguiente manera: la probabilidad del nivel L de la clase C, dada la evidencia proporcionada por las variables de F₁, ..., F_n, es igual al producto de las probabilidades de cada evidencia condicionada al nivel de la clase, la probabilidad *a priori* del nivel de la clase y un factor de escala 1/Z que convierte el resultado en probabilidad:

$$P(C_L | F_1, \dots, F_n) = \frac{1}{Z} p(C_L) \prod_{i=1}^n p(F_i | C_L)$$

Este clasificador se utiliza principalmente para **clasificar texto, para detección de intrusos en redes de computadores, diagnósticos médicos**, etc. Por ejemplo, se puede utilizar la frecuencia de las palabras de los correos electrónicos para identificar nuevos correos *spam* en el futuro.

Combinaciones desconocidas

Supongamos que ahora recibimos un mensaje que contiene las palabras *Viagra*, *groceries*, *money* y *unsubscribe*, y queremos estimar la probabilidad de que el mensaje sea *Viagra*. En este caso la verosimilitud de *spam* es:

$$(4/20) * (10/20) * (0/20) * (12/20) * (20/100) = 0$$

Por otro lado, la verosimilitud de *ham* es:

$$(1/80) * (14/80) * (8/80) * (23/80) * (80/100) = 0.00005$$

La probabilidad de *spam* es:

$$0/(0 + 0.0099) = 0$$

Y la probabilidad de *ham* es:

$$0.00005 / (0 + 0.00005) = 1$$

Este problema sucede cuando un evento nunca ha ocurrido para una o más categorías de las clases. Por ejemplo, si nunca se ha visto el término *groceries* en un mensaje *spam* $P(\text{Spam} / \text{groceries}) = 0$.

La solución es añadir un pequeño número a todas las clases en la tabla, para asegurarse que no existe ninguna combinación con probabilidad de ocurrir igual a 0, esto se conoce con el nombre de **estimador de Laplace**.

Por ejemplo, si usamos un valor de 1, la verosimilitud de *spam* y *ham* quedaría:

$$(5/24) * (11/24) * (1/24) * (13/24) * (20/100) = 0.0004$$
$$(2/84) * (15/84) * (9/84) * (24/84) * (80/100) = 0.0001$$

Lo que indica que la probabilidad de que el mensaje sea *spam* es del 0,8 y de que sea *ham* del 0,2.

Clasificador Naive Bayes con variables numéricas

Debido a que el clasificador Naive Bayes utiliza tablas de frecuencias para calcular las probabilidades, cada una de las variables utilizada debe de ser **categórica** y no se pueden utilizar de forma directa variables numéricas.

Una solución sencilla es **discretizar las variables numéricas en N conjuntos, agrupamientos o bins**. Este método es ideal cuando hay grandes cantidades de datos. Una cuestión importante aquí es considerar el punto de corte óptimo para hacer cada uno de los agrupamientos. Una buena solución suele ser explorar los datos para observar los puntos de corte en la distribución de los datos.

Por ejemplo, el siguiente histograma:

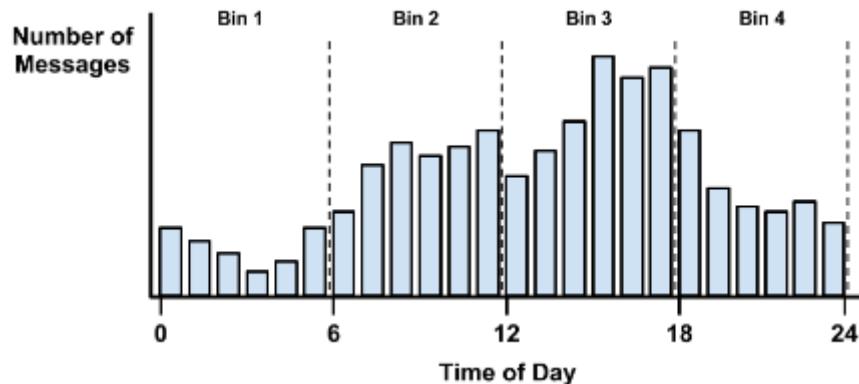


Gráfico 1. Fuente: Lantz, 2013.

Sugiere realizar una división en cuatro *bins*.

La **discretización** siempre se traduce en una reducción de la información, ya que la granularidad inicial se reduce. Por tanto, es importante mantener un balance entre el número de *bins*, puesto que con muy pocas se pierda mucha información y con muchas el proceso es muy costoso.



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Resumen del tema» a través del aula virtual

4.5. Referencias bibliográficas

Lantz, B. (2013). *Machine Learning with R*. Packt

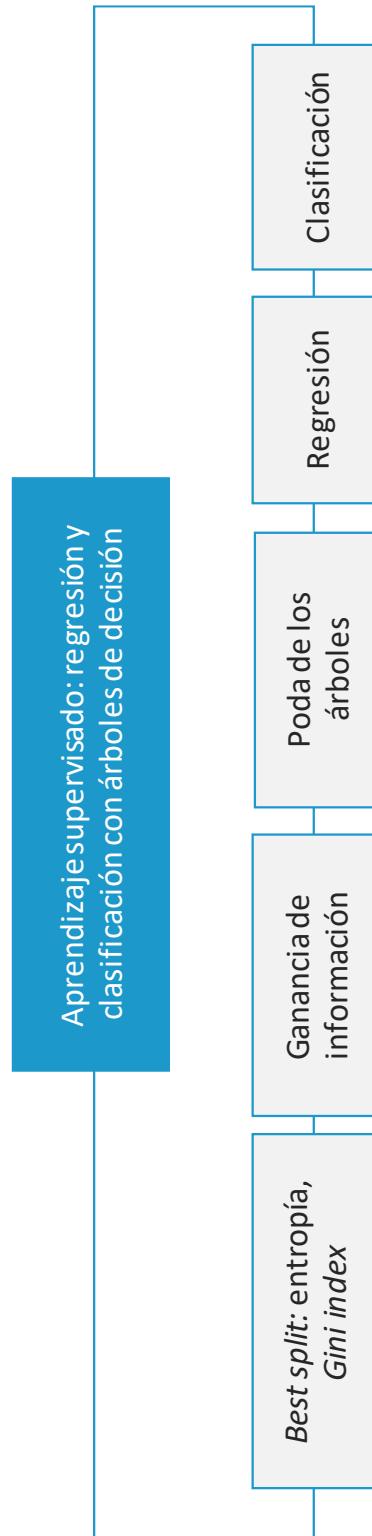
Aprendizaje Automático

Aprendizaje supervisado: regresión y clasificación con árboles de decisión

Índice

Esquema	3
Ideas clave	4
5.1. Introducción y objetivos	4
5.2. Introducción a los árboles de decisión	5
5.3. Poda de los árboles	9
5.4. Árboles para clasificación	13
5.5. Referencias bibliográficas	14

Esquema



5.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos a los árboles de decisión» a través del aula virtual

E

n este tema veremos los modelos de clasificación y regresión basado en los árboles de decisión.

- ▶ En primer lugar, veremos una introducción a los árboles de decisión.
- ▶ A continuación, se hará énfasis en cómo se decide el mejor corte, cubriendo para ello los conceptos de índice *Gini* y ganancia de información.
- ▶ Más adelante, se describe la necesidad de realizar la poda de los árboles.
- ▶ Posteriormente, se describen brevemente las diferencias con los árboles utilizados para clasificación.
- ▶ Finalmente se muestran las diferencias entre los árboles y un modelo de regresión lineal.



Accede a los ejercicios de autoevaluación a través del aula virtual

5.2. Introducción a los árboles de decisión



Accede al vídeo «Entropía, *gini*, ganancia de información» a través del aula virtual

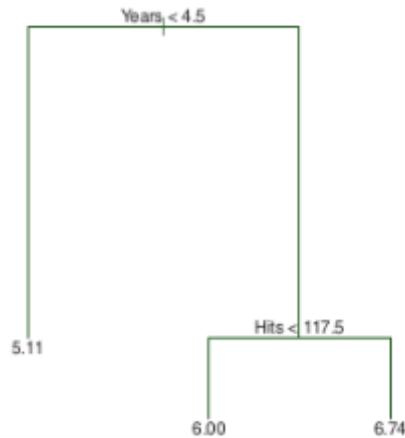
Los árboles de decisión son una de las **técnicas más populares dentro del campo del aprendizaje automático**. Se llevan utilizando durante muchos años en el área de la minería de datos. Se trata de métodos **sencillos** y fáciles de **interpretar**.

Estos modelos **dividen o segmentan** el espacio de las variables predictoras en una serie de regiones. Una vez creado el árbol de decisión es utilizado para predecir observaciones futuras. Para este propósito se utiliza la **moda** en el caso de que la variable a predecir sea categórica o bien la **media** en el caso de que sea numérica.

Como el conjunto de las reglas para separar las variables predictoras se pueden resumir en forma de árbol, a estos métodos se les conoce popularmente con el nombre de **árboles de decisión**.

Los árboles de decisión **dividen el espacio en rectángulos** que minimizan el error de la predicción. Debido a que es computacionalmente imposible considerar cualquier posible partición del espacio de entrada se utiliza un enfoque *top-down greedy* conocido como *recursive binary splitting*.

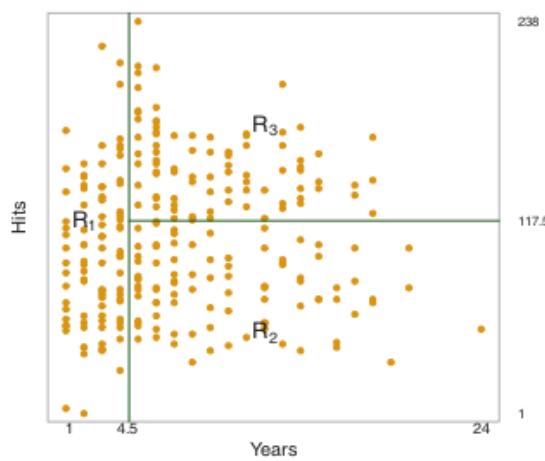
En la siguiente imagen se muestra un **árbol de decisión sencillo** para obtener el logaritmo del salario de los jugadores de béisbol. Este valor se obtiene por medio de las variables de años en la liga (*years*) y de golpes (*hits*) y utilizando unas reglas de decisión en base a los umbrales de estas variables. Así, por tanto, aquellos jugadores con más de 4,5 años de experiencia y más de 117,5 golpes estarían en 6,74, mientras que si tuvieran menos de 117,5 golpes estarían en 6,0.



Gráfica 1. Ejemplo de un árbol de decisión sencillo para estimar el salario de los jugadores de béisbol en función de los años y de los golpes. Fuente: James et.al., 2017.

El árbol de decisión anterior se obtiene por medio de la representación en un espacio de dos dimensiones de los puntos donde están el eje de años (*years*) y el de los golpes (*hits*).

A cada uno de esos puntos le corresponde un salario determinado y el objetivo es agrupar aquellos puntos que tienen un salario similar utilizando la información de las otras dos variables (golpes y años de experiencia).



Gráfica 2. Representación en dos dimensiones de la distribución de los golpes y años en la liga de los jugadores. Fuente: James et.al., 2017.

El árbol de decisión se obtiene **por medio de un algoritmo** que elige primero aquella variable que es más predictiva de la variable objetivo. A continuación, los ejemplos

de entrenamiento se dividen en grupos con distintos valores para las clases de esta primera variable. El algoritmo continúa dividiendo los nodos con la elección de la mejor variable en cada iteración hasta que se alcance el criterio de parada. En cada iteración el algoritmo elige aquella variable que mejor predice la variable objetivo, por tanto, se trata de un algoritmo de tipo *greedy*. El criterio de parada puede venir dado por algunas de estas situaciones:

- ▶ Todos, o casi todos, los ejemplos del nodo son de la misma clase.
- ▶ No existen variables para distinguir entre los ejemplos.
- ▶ El árbol ha alcanzado un tamaño predefinido.

Existen numerosas implementaciones de los árboles de decisión disponibles en el mercado, no obstante, las más famosas son:

- ▶ C.5.0 desarrollado por Ross Quinlan.
- ▶ C 4.5.
- ▶ ID3 (*Iterative Dichotomiser 3*).
- ▶ J48, una alternativa basada en Java *open source* del C.4.5.

De todos ellos el algoritmo C.5.0 está considerado como el estándar en la industria.

Los árboles de decisión dividen o segmentan el espacio de las variables predictoras en una serie de regiones. En el caso de los árboles utilizados para modelos de regresión se utiliza la **media** para estimar los valores que se encuentran en una determinada región. En el caso de los modelos de clasificación se utiliza la **moda** de la clase.

Best split: entropía, Gini index, ganancia de información

Cada una de las divisiones del árbol da como resultado dos grupos. Es decir, los datos resultantes de la división por la variable elegida y con el punto de corte determinado

están en uno de los dos grupos. Cuando los segmentos de una división contienen valores de una única clase se considera que es una **división pura**.

A la hora de identificar los criterios para realizar el *split* o corte existen varias métricas de pureza. Muchos algoritmos, por ejemplo, el C.5.0 utilizan el concepto de **entropía**. En este caso un valor de 0 indica que la muestra es completamente homogénea, mientras que un valor de 1 indica desorden completo.

La **entropía** se define con la siguiente fórmula matemática:

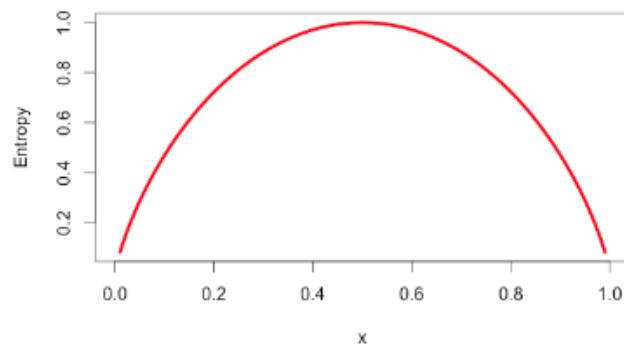
$$\text{Entropy}(S) = \sum_{i=1}^c -p_i \log_2(p_i)$$

Por ejemplo, si se ha realizado una partición utilizando una variable y umbral determinado y los datos de dos clases se dividen en un 60 % en un lado de la rama y en un 40 % en otro lado de la rama, tendríamos el siguiente valor de entropía:

```
> -0.60 * log2(0.60) - 0.40 * log2(0.40)
[1] 0.9709506
```

Si conocemos que la proporción de ejemplos de una clase es x , se puede examinar la entropía de todas las posibles divisiones de dos clases con el lenguaje de programación R utilizando la función `curve()`:

```
> curve(-x * log2(x) - (1 - x) * log2(1 - x), col = "red", xlab = "x", ylab = "Entropy", lwd = 4)
```



Gráfica 3. Ejemplo de los posibles valores de entropía de dos clases en función de las divisiones de cada una de las clases. Cuando la división entre las dos clases es cercana a 0,5 el valor de entropía es cercano a 1. Cuanto más cercana sea la división hacia la mayoría de una de las clases más cercano a cero es el valor de la entropía.

Fuente: James et.al., 2017.

Utilizando la medida de pureza el algoritmo tiene que decidir con que variable hacer el corte. Una opción es **utilizar la entropía** para calcular el cambio resultante de hacer el corte en esa variable.

A continuación, se calcula la ganancia de información (*information gain o IG*) que es la diferencia entre la entropía en el segmento antes de hacer el corte (S_1) y la partición resultante de hacer el corte (S_2), es decir:

$$\text{InfoGain}(F) = \text{Entropy}(S_1) - \text{Entropy}(S_2)$$

Después de un corte los datos se pueden dividir en más de una partición, por tanto, se considera la entropía a lo largo de las N particiones, ponderando la entropía de cada partición con el número de instancias de esa partición:

$$\text{Entropy}(S) = \sum_{i=1}^n W_i \text{Entropy}(P_i)$$



Accede a los ejercicios de autoevaluación a través del aula virtual

5.3. Poda de los árboles



Accede al vídeo «Poda de los árboles; árboles vs. modelos lineales» a través del aula virtual

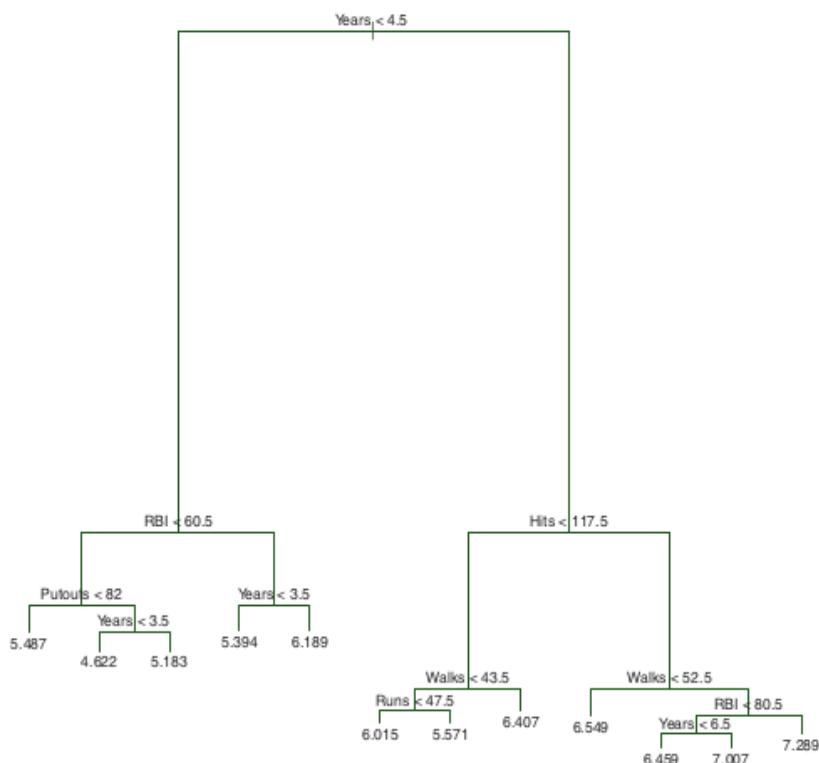
El proceso anterior de generación de cortes recursivos denominado *top-down greedy* puede generar buenas predicciones en el conjunto de entrenamiento, pero también sufre el **problema del sobre ajuste**.

Este motivo se debe a que el árbol resultante puede llegar a ser muy complejo y ajustarse muy bien a los datos de entrenamiento. Un árbol mucho más pequeño puede dar lugar a una menor **varianza** y mejor interpretación con el coste de un pequeño sesgo.

La estrategia para generar estos árboles más pequeños y con una menor varianza suele ser **generar un árbol muy grande** y después podarlo para obtener un sub-árbol (*post-prunning*).

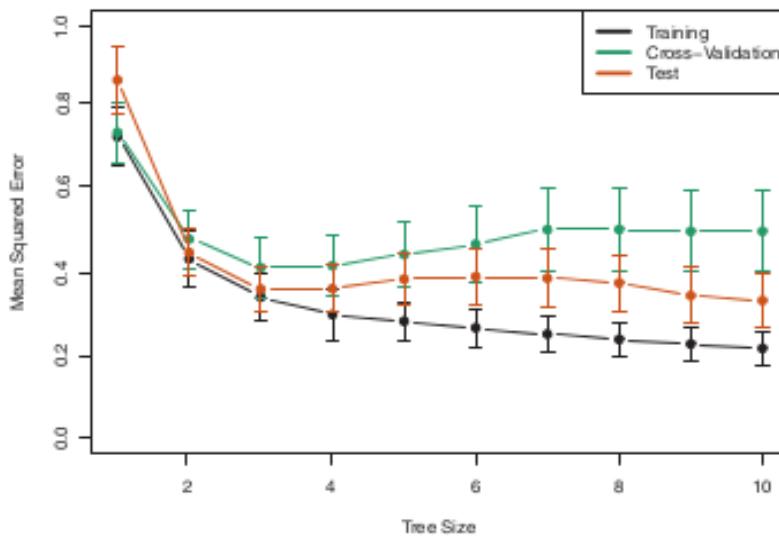
Ahora bien, la cuestión es, ¿cómo seleccionamos el sub-árbol? Una buena solución es utilizar aquel sub-árbol que proporcione un menor error de test en validación-cruzada (*cross validation*) o bien en el conjunto de validación.

Por ejemplo, en la gráfica 4 se muestra la obtención de un árbol sin podar. En la gráfica 5 se muestra el error obtenido por este árbol en función de la profundidad para los conjuntos de entrenamiento (*train*), test y validación cruzada (*cross validation*). Se observa que el punto óptimo es utilizar tres niveles.



Gráfica 4. Ejemplo de un árbol sin podar con bastantes ramas.

Fuente: James et.al., 2017.



Gráfica 5. Ejemplo de la evolución del error del árbol de la figura 4 en función del número de niveles y para 3 conjuntos de datos. Fuente: James et.al., 2017.

Árboles vs. modelos lineales

Una regresión lineal asume un modelo que tiene la siguiente forma:

$$f(X) = \beta_0 + \sum_{j=1}^p X_j \beta_j$$

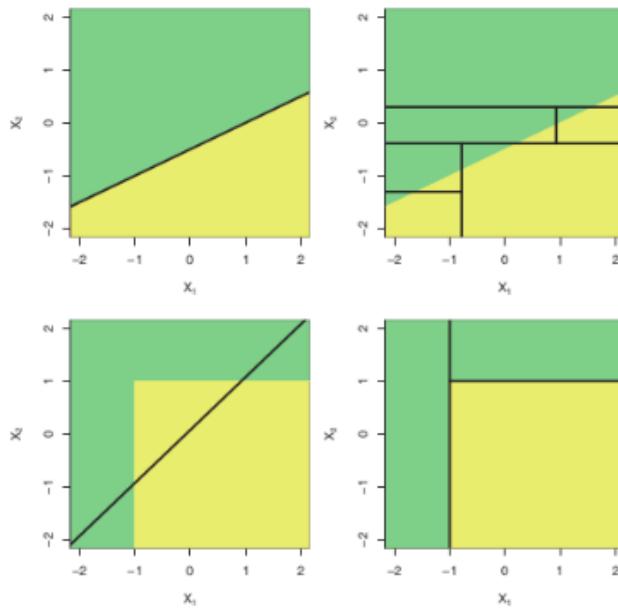
Mientras que un árbol de regresión, asume un modelo:

$$f(X) = \sum_{m=1}^M c_m \cdot 1(x \in R_m)$$

¿Qué modelo es mejor? Depende de lo que se esté **modelando**. Si las relaciones entre las variables de entrada y la variable respuesta se pueden aproximar por un modelo lineal, una regresión lineal funciona bien y supera a un árbol de regresión. Esto se debe a que un árbol de regresión no es capaz de modelar esta estructura.

Sin embargo, si la naturaleza del problema es no lineal y con relaciones complejas entre las variables, los árboles funcionan mejor.

Esta situación se refleja en la siguiente figura:



Gráfica 7. Ejemplo de ajustes a una región utilizando un modelo basado en árboles y uno basado en una regresión lineal. Si los puntos se distribuyen como en la imagen superior una regresión lineal (imagen izquierda superior) proporciona el mejor ajuste, sin embargo, un modelo en árbol (imagen izquierda derecha) se comporta mal. En el caso de una separación no lineal como ocurre en las imágenes inferiores esta situación se invierte.

Fuente: James et.al., 2017.



Accede a los ejercicios de autoevaluación a través del aula virtual

5.4. Árboles para clasificación



Accede al vídeo «Árboles para clasificación» a través del aula virtual

Los árboles se pueden utilizar para realizar una predicción numérica en el caso de regresión o bien para realizar una clasificación. Los árboles de clasificación (*classification trees*) son similares a los *regression trees* pero predicen la respuesta de una variable categórica.

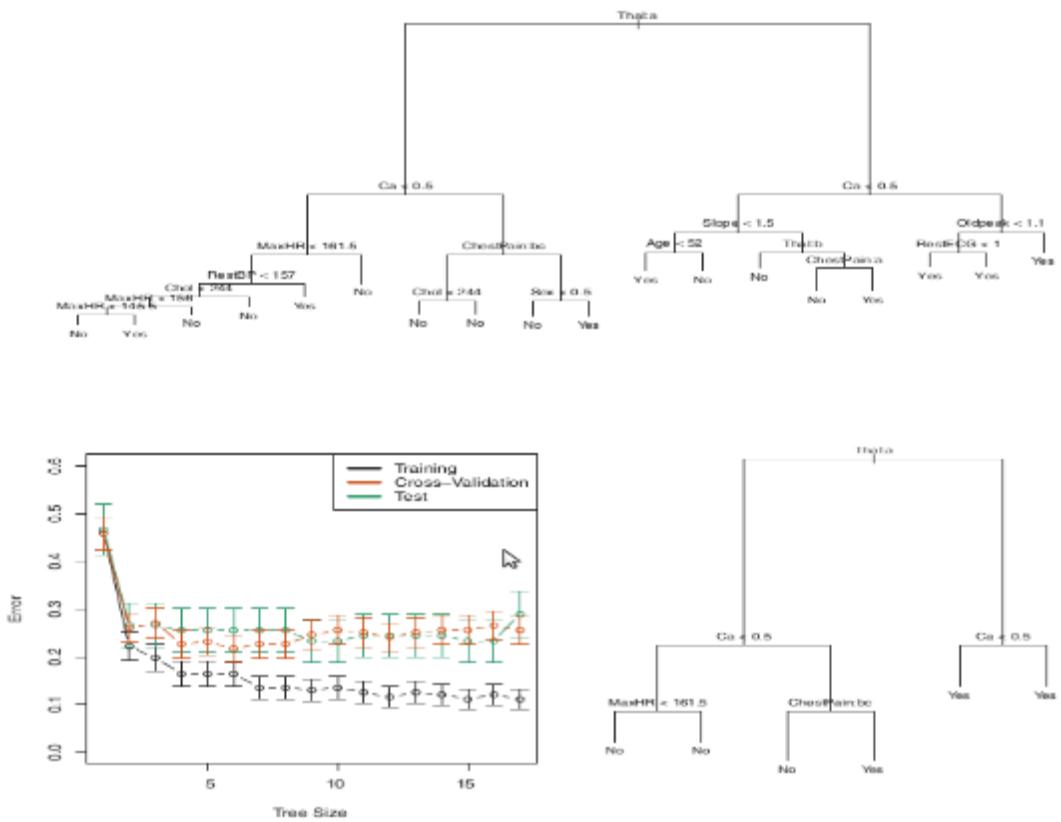
En *regression trees* la predicción corresponde a la media de las observaciones de ese nodo terminal. En *classification trees* se trata de la clase que tiene un mayor número de ocurrencias en ese nodo terminal.

En el caso de clasificación, el criterio que se suele utilizar para realizar los cortes o *splits* es el *Gini index*, que se define:

$$G = \sum_{K=1}^K \widehat{p_{mk}} (1 - \widehat{p_{mk}})$$

O bien la entropía cruzada, la cual se define como:

$$D = - \sum_{K=1}^K \widehat{p_{mk}} \log \widehat{p_{mk}}$$



Gráfica 6. Ejemplo de un árbol utilizado para realizado para clasificación. En la imagen superior se muestra el árbol completo. En la imagen inferior izquierda se muestra la evolución del error en función de la profundidad del árbol. En la imagen inferior derecha se observa el árbol podado resultante. Fuente: James et.al., 2017.



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Resumen del tema» a través del aula virtual

5.5. Referencias bibliográficas

James, G., Witten, D., Hastie, T. y Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer.

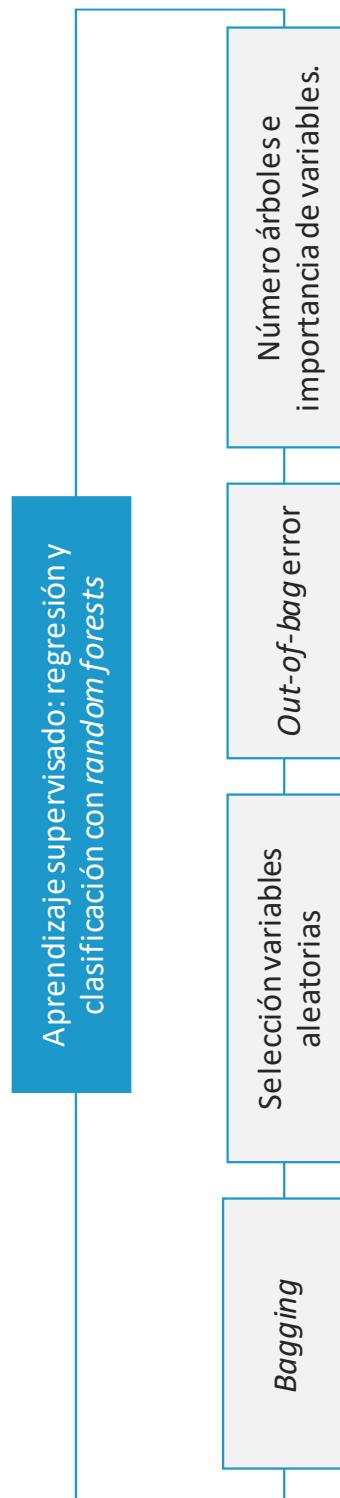
Aprendizaje Automático

Aprendizaje supervisado: regresión y clasificación con *random forests*

Índice

Esquema	3
Ideas clave	4
6.1. Introducción y objetivos	4
6.2. Explotando la diversidad: <i>bagging</i> y selección de variables	5
6.3. Interpretación de <i>out-of-bag</i> error	8
6.4. Evolución del número de árboles e importancia de variables	9
6.5. Referencias bibliográficas	11

Esquema



6.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos: clasificador *random forests*» a través del aula virtual

R

andom forests es un **método basado en ensambles** de árboles de decisión que fue inventado por Leo Breiman y Adele Cutler en el año 2001 (Breiman, 2001).

En este tema se describe como los modelos de *random forests* explotan la diversidad y son capaces de generalizar y predecir mejor que un árbol de decisión. Esta diversidad se consigue por medio de entrenar modelos con el método *bagging* y de realizar una selección de variables aleatoria.

A continuación, se discute la interpretación del *out-of-bag error*. Posteriormente nos enfocamos en determinar el número de árboles y en evaluar la importancia de las variables.

6.2. Explotando la diversidad: *bagging* y selección de variables



Accede al vídeo «Explotando la diversidad: *bagging* y selección de variables» a través del aula virtual

Uno de los inconvenientes principales de los árboles de decisión es su **baja capacidad predictiva**. Este inconveniente se puede solventar por medio de utilizar un ensemble o combinación de modelos de árboles de decisión. El modelo *random forests* es en esencia un ensemble de árboles de decisión. Los ensembles de árboles se pueden combinar utilizando los métodos de *bagging* o *boosting*. Los modelos de *random forests* se basan en combinar los árboles por medio de los métodos de *bagging*.

En el caso de problemas de clasificación en cada observación de test se almacena la clase predicha por cada uno de los B árboles y la clase final se obtiene por medio del voto de la mayoría. Es decir, la **predicción global** es la clase que más veces ocurre a lo largo de las B predicciones. Por otro lado, en el caso de los problemas de regresión la predicción global es la media de las predicciones de cada uno de los árboles.

El modelo de *random forests* combina los principios de *bagging* con selección de variables aleatorias para añadir diversidad a los árboles de decisión. Una vez es generado el ensemble de árboles (*forest*) el modelo utiliza el mecanismo de votación o la media para generar las predicciones.

Se trata de un modelo que **combina versatilidad y potencia en un enfoque**. A la hora de construir cada uno de los árboles se utiliza una porción pequeña y aleatoria de las variables de entrada disponibles, por lo general, este número se define como \sqrt{p} siendo p el número de variables de entrada.

Al ser un modelo que genera cada árbol con un subconjunto de los registros de entrada y una selección aleatoria de las variables, puede trabajar con conjuntos de datos bastante grandes y no se encuentra afectado por los problemas de *curse of dimensionality*. Además, debido a que los árboles son modelos con mucho ruido y muy inestables se pueden beneficiar de forma notoria de ser promediados.

En cada uno de los cortes de cada uno de los árboles se elige una serie de variables candidatas de entre todas las posibles. El valor por defecto, \sqrt{p} variables aleatorias de entre todas las posibles, suele dar buenos resultados.

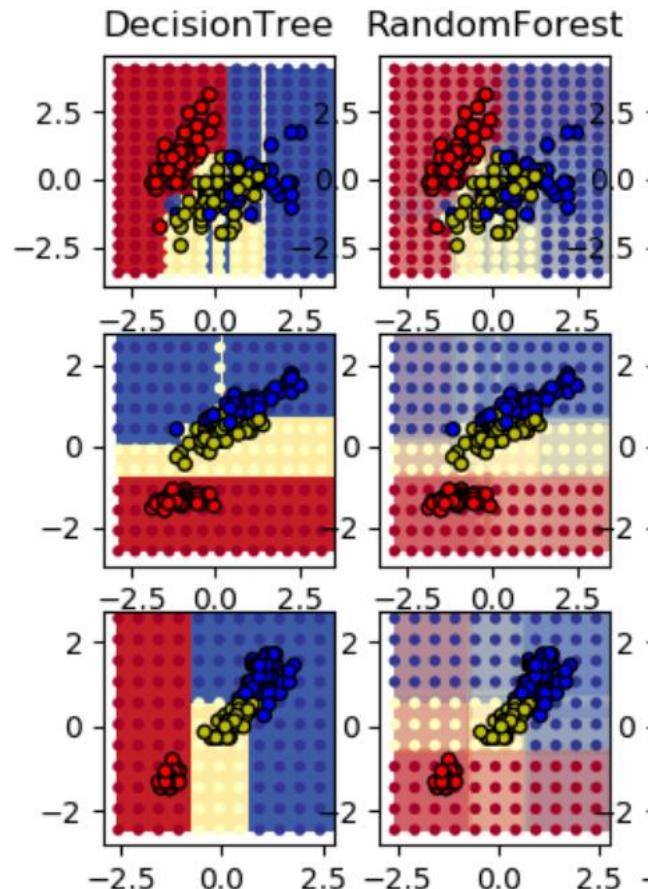
Este modelo se basa en la **utilización de un gran número de árboles**. La razón de utilizar un gran número de árboles es para que cada variable, de entre todas las posibles, tenga la oportunidad de aparecer en varios modelos.

La ventaja principal de los modelos de *random forests* frente a un modelo *bagged* de árboles es debido a que las variables de cada *split* de los árboles se obtienen por medio de **seleccionar un subconjunto de todas las variables de forma aleatoria**. De esta forma, se consigue decorrelar los árboles generados. Además, se reduce la varianza porque el resultado se obtiene a partir de calcular el promedio de los árboles y se controla el sobreajuste.

Por tanto, cada árbol de decisión se construye utilizando *bootstrapped training samples* y en cada *split* se utiliza una variable a partir de una selección aleatoria de m predictores del total p . Finalmente, los distintos árboles se combinan utilizando el voto de la mayoría para clasificación y la media para regresión.

En los *random forests* debido a la selección de variables aleatorias sobre el conjunto de entrada completo, el sesgo de los árboles se suele incrementar (con respecto al sesgo un único árbol de decisión); no obstante, debido a promediar los árboles la varianza se reduce y, por lo general, en una proporción mayor que el incremento del sesgo dando lugar a un mejor modelo por lo general. En muchos problemas el rendimiento de los *random forests* es similar a los modelos *boosting* a pesar de ser

más sencillos de entrenar. Como consecuencia de este hecho, los *random forests* se han convertido en una técnica muy popular.



Gráfica 1. Regiones detectadas por un árbol de decisión y un modelo de *random forests* sobre el dataset de iris. Fuente: http://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_iris.html



Accede a los ejercicios de autoevaluación a través del aula virtual

6.3. Interpretación de *out-of-bag* error



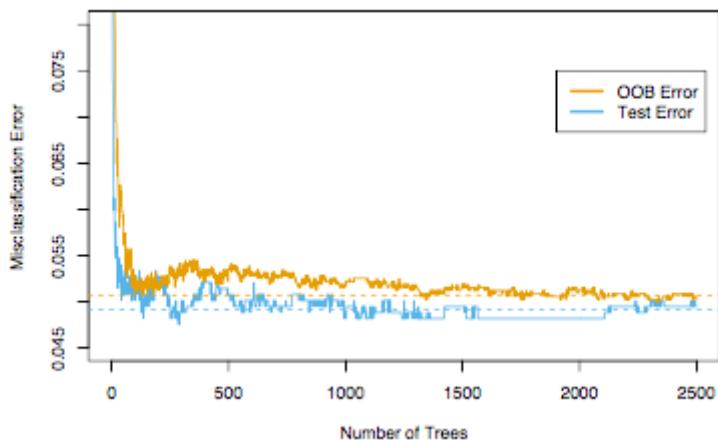
Accede al vídeo «Interpretación de *out-of-bag* error» a través del aula virtual

Una característica importante de los modelos *random forests* es el error *out-of-bag*. El *out-of-bag* error es una forma sencilla de estimar el error de test en un modelo *bagged*. Se puede demostrar que de media cada árbol construido con un modelo *bagged* utiliza 2/3 de las observaciones del conjunto de entrenamiento. El 1/3 restante de las observaciones de entrenamiento no se utilizan para generar el árbol *bagged* y son llamadas *out-of-bag*.

Para cada observación se puede obtener la respuesta de los *bagged tres*, donde esta observación era *out-of-bag*. Esto proporciona alrededor de $B/3$ predicciones para cada una de las observaciones.

Si el valor de B (número de árboles) es grande esta estimación es equivalente a un modelo de tipo *leave-one-out* de validación cruzada. Este modelo de *leave-one-out* es la forma más rigurosa y extrema de evaluar un modelo de aprendizaje automático.

Un modelo *random forests* se puede entrenar de forma secuencial, donde no es necesario utilizar validación cruzada, pues las muestras *out-of-bag* proporcionan una estimación similar. Para ello, es necesario observar la evolución del *out-of-bag* error y una vez que este error se haya estabilizado el algoritmo puede parar el entrenamiento.



Gráfica 2. Evolución del *out-of-bag error*. La gráfica muestra la evolución de 2500 árboles, pero se observa que 200 árboles serían suficientes. Fuente: Hastie et al., 2001.



Accede a los ejercicios de autoevaluación a través del aula virtual

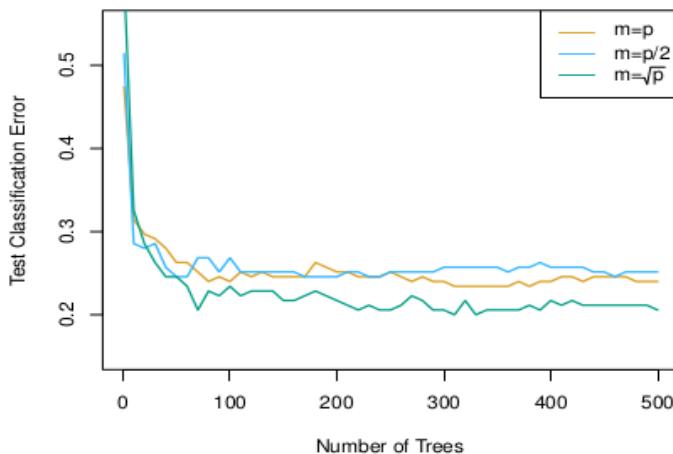
6.4. Evolución del número de árboles e importancia de variables



Accede al vídeo «Evolución del número de árboles e importancia de variables» a través del aula virtual

Los modelos basados en *random forests* principalmente tienen **dos parámetros para optimizar**: el número de árboles y el número de variables que se evalúan en cada tirada. Existen otros parámetros como la profundidad de los árboles, pero en la práctica no presentan muchos cambios.

En la siguiente gráfica se observa la evolución del error en el conjunto de test en función del número de árboles y del número de variables que se prueban en cada *split*. Se observa que el valor de \sqrt{p} **proporciona un error menor que los otros valores**.



Gráfica 3. Tres diferentes resultados de un *random forests* para un problema de clasificación de 15 clases y que tiene 500 predictores. Fuente: James et. al., 2017.

En principio se puede pensar que cuanto más grande sea el número de árboles mejor. Sin embargo, por lo general existe un **punto óptimo** donde a pesar de añadir más árboles el error no se reduce de forma significativa.

Una de las ventajas de los *random forests* es que pueden generar buenos resultados sin necesidad de muchos ajustes manuales. Además, permiten construir de forma sencilla gráficos de importancia de variables.

En cada decisión de corte de cada árbol la mejora en el criterio de corte es la medida de importancia que se atribuye al **corte de esa variable** y se agrega para todos los árboles del *forest* para cada variable.

Random forest también utiliza las muestras fuera del *bag* (OOB) para construir una medida de importancia de variables que mide la capacidad predictiva de cada una de las variables.



Accede al vídeo «Resumen del tema» a través del aula virtual



Accede a los ejercicios de autoevaluación a través del aula virtual

6.5. Referencias bibliográficas

Breiman, L. (2001). Random Forests. *Machine Learning*. 45(1), 5-32.

Hastie, T., Tibshirani, R. y Friedman, J. (2001). *The elements of Statistical Learning*. Springer.

James, G., Witten, D., Hastie, T. y Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer.

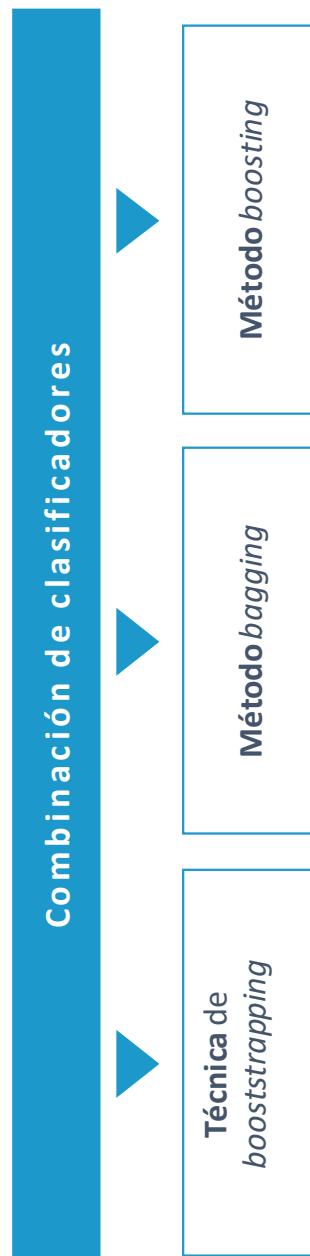
Aprendizaje Automático

Combinación de clasificadores: *bootstrapping, bagging y boosting*

Índice

Esquema	3
Ideas clave	4
7.1. Introducción y objetivos	4
7.2. Técnica de <i>bootstrapping</i>	5
7.3. Método <i>bagging</i>	6
7.4. Método de <i>boosting</i>	8
7.5. Referencias bibliográficas	12

Esquema



7.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos» a través del aula virtual

En este tema se van a estudiar los métodos de ensemble. Estos métodos combinan las predicciones de varios estimadores base, que han sido construidos con un algoritmo de aprendizaje automático, para mejorar la generalización/robustez de un único modelo.

Por lo general los métodos de ensemble o combinación de clasificadores se pueden dividir en dos grandes familias:

1. *Averaging methods*: los cuales se basan en construir varios estimadores de forma independiente y luego promediar las predicciones. En media, el estimador combinado se suele comportar mejor que cualquier estimador individual porque la varianza se reduce. Los métodos de *bagging* funcionan de esta forma.
 2. *Boosting methods*: los estimadores base se construyen de forma secuencial con el objetivo de reducir el sesgo del estimador combinado. La motivación es combinar varios modelos débiles para producir un modelo combinado potente.
-
- ▶ En primer lugar, a partir del capítulo de introducción se va a introducir la técnica estadística de *bootstrapping*.
 - ▶ A continuación, se describe el método *bagging*, el cual se apoya en la técnica de *bootstrapping* para ensamblar modelos.
 - ▶ Finalmente, veremos el método de *boosting*, uno de los métodos más populares para llevar a cabo la combinación de modelos débiles para construir un modelo más robusto y con mayor precisión.

7.2. Técnica de bootstrapping



Accede al vídeo «Técnica de bootstrapping» a través del aula virtual

Bootstrapping es una **técnica estadística** que consiste en cualquier test o métrica sobre muestreo aleatorio con remplazamiento. *Bootstrapping* **permite asignar medidas de precisión** definidas en términos de sesgo, varianza, intervalos de confianza o cualquier otra métrica sobre estimaciones muestrales. Esta técnica permite la estimación de la distribución de la muestra de cualquier estadístico utilizando métodos de muestreo. Generalmente, se suele clasificar como una clase de método de muestreo.

Bootstrapping es la **práctica de estimar las propiedades de un estimador** (así como su varianza) por medio de medir las propiedades al muestrear una distribución aproximada. Una elección habitual para aproximar la distribución es utilizar la función de distribución empírica de los datos observados. En el caso de que las observaciones se puedan asumir que se generan a partir de una población independiente e idénticamente distribuidas, pueden ser obtenidas por medio de muestreos con remplazamiento del conjunto de datos observados.

La idea sobre la que se basa la técnica de *bootstrapping* es que la inferencia sobre una población a partir de una muestra de datos puede ser **modelada por medio de re-muestrear los datos de la muestra** y llevar a cabo inferencias sobre un remuestreo de la muestra. Como la población es desconocida el error verdadero en una muestra estadística sobre el valor de la población es desconocido. En el remuestreo en *bootstrapping*, la población es en realidad la muestra, la cual es conocida. Por tanto, la calidad de la inferencia de la muestra verdadera a partir de los datos muestreados es medible.

Como ejemplo, supongamos que nos interesa conocer la **media de la altura de la población mundial**. Debido a que es complejo medir la altura de todas las personas

en el mundo, en su lugar podemos muestrear una parte de ellas y medir la altura en ellas. Supongamos que el tamaño de esta muestra es N , entonces tenemos la altura de N personas. Con esta muestra solo podemos tener una estimación de la altura media de la población. Para obtener una mejor imagen de la población necesitamos obtener algún tipo de variabilidad sobre los datos obtenidos. El método de *bootstrap* más sencillo para obtener esta variabilidad consiste en muestrear con remplazamiento una muestra de tamaño N . Por ejemplo, si muestreamos sobre [1,2,3,4,5] podríamos obtener [2,5,3,3,1]. Cuando el tamaño N es suficientemente grande para todos los efectos prácticos existe una probabilidad casi cero de que sea una muestra idéntica a la muestra inicial. Si se repite este proceso miles de veces y para cada muestra se obtiene la media, se obtiene un histograma de medias obtenido con *bootstrap*.

El método *bagging*, descrito a continuación, **consiste en un meta-algoritmo** para ponderar los resultados de múltiples muestras de *bootstrapping*.



Accede a los ejercicios de autoevaluación a través del aula virtual

7.3. Método *bagging*



Accede al vídeo «Método de *bagging*» a través del aula virtual

El método de *bagging* es una clase de algoritmos de ensemble o combinación de clasificadores. El término *bagging* proviene de la contracción de *bootstrap aggregation*. Se trata de un procedimiento general que permite reducir la varianza de un método de *machine learning*. Es un método para **combinar varias instancias de estimadores de caja negra** que se han construido sobre muestras aleatorias del conjunto de entrenamiento original y que agregan las predicciones individuales para obtener una predicción única.

Dado un conjunto de n observaciones independientes $Z_1 \dots Z_n$, cada una con una varianza σ^2 , la varianza de la media Z de las observaciones es σ^2/n .

Es decir, la media de un conjunto de observaciones reduce la varianza. Sin embargo, como lo habitual es no tener muchas observaciones se suele hacer *bootstrap* tomando muestras repetidas del *data set* de entrenamiento.

Para ello, se entranan B conjuntos de entrenamiento distintos y se obtiene la media de las predicciones:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^B \hat{f}^{*b}(x)$$

Estos métodos se utilizan como una forma para reducir la varianza de un estimador base (ejemplo: árboles de decisión) por medio de introducir aleatoriedad en el procedimiento de construcción y a continuación construir el ensemble.

En muchos casos, los métodos de *bagging* son un método sencillo de mejorar, un único modelo sin la necesidad de tener que modificar el algoritmo de entrenamiento base. Como se trata de un método para reducir el *overfitting*, los métodos de *bagging* funcionan mejor cuando se utilizan con modelos complejos (ejemplo: árboles de decisión profundos), en contraste con los métodos de *boosting* que funcionan mejor con modelos simples (ejemplo: shallow decisión trees).

Existen **diferentes variantes** de los métodos de *bagging*, pero dependen principalmente de la forma en que obtienen las muestras del conjunto de entrenamiento:

- ▶ Cuando se obtienen muestras aleatorias del conjunto de datos como subconjuntos aleatorios de las muestras, el algoritmo se conoce como *pasting* (Breiman, 1999).
- ▶ Cuando se obtienen los conjuntos para entrenar los algoritmos por medio de muestreo con remplazamiento del conjunto de entrenamiento, el método se conoce como *bagging* (Breiman, 1998).

- ▶ Cuando se obtienen subconjuntos aleatorios como subconjuntos del espacio de las variables, el método se conoce como *random subspaces* (Kam, 1998).
- ▶ Por último, cuando los estimadores base se construyen con subconjuntos de muestras y variables, el método se conoce como *random patches* (Louppe. y Geurts, 2012).

Out-of-bag (OOB) error

Existe una forma sencilla de estimar el error de test en un modelo *bagged*, **por medio del conjunto de datos que se queda fuera de la muestra**. Este conjunto de datos fuera de la muestra, por lo general, suele ser 1/3 del conjunto de datos total. Por tanto, para este 1/3 del conjunto total se puede predecir la respuesta que se hubiera obtenido. Cuando el valor B (número de conjuntos de entrenamiento distintos) es grande, la estimación del *out-of-bag* error es equivalente al *leave-one-out cross validation*.



Accede a los ejercicios de autoevaluación a través del aula virtual

7.4. Método de *boosting*



Accede al vídeo «Método de *boosting*» a través del aula virtual

Al igual que *bagging*, se trata de un **método de combinación de modelos** que se puede aplicar a los modelos de regresión y de clasificación.

En *bagging* se crean múltiples copias del conjunto de entrenamiento original utilizando el muestreo *bootstrap* y entrenando un modelo en cada copia, para posteriormente combinar todos los modelos en uno solo. En *bagging* cada modelo se construye en un *bootstrap data set* independiente de los otros. Sin embargo, en el

caso de *boosting* los modelos se generan de forma secuencial, es decir, cada modelo utiliza información de los modelos anteriores.

Boosting es la técnica seguida por los modelos de *generalized boosted models* (GBMs). En estos modelos, en lugar de entrenar un gran árbol de decisión sobre los datos, lo cual es complicado y puede dar lugar a *overfitting*, se utiliza el enfoque *boosting* para aprender poco a poco. Cada uno de los árboles puede ser bastante pequeño, con pocos nodos terminales, pero todos ellos se combinan.

Por tanto, el método *boosting* puede verse como un **meta-algoritmo de ensemble** para reducir sesgo y varianza el cual se aplica sobre algoritmos de *machine learning* supervisados con el objetivo de convertir modelos simples en modelos más precisos. Este método se basa en una pregunta formulada por Kearns y Valiant (año 1989): «¿Puede un conjunto de modelos débiles crear un modelo fuerte?».

En este contexto un **modelo débil** se define como un clasificador que está poco correlado con la clase real. Se trata de ejemplos ligeramente mejor que el acierto aleatorio. Por otro lado, un modelo fuerte se define como un clasificador que está bien correlado con la clase verdadera de la clasificación.

La respuesta a la pregunta de Kearns y Valiant ha tenido una gran repercusión en el mundo de *machine learning* dando lugar al desarrollo de los modelos de *boosting*. Cuando se introdujo esta hipótesis de problemas *boosting* dio lugar a que los algoritmos que consiguieran mejorar por medio de la combinación de modelos débiles fueran llamados *boosting*.

A pesar de que los algoritmos *boosting* no tienen ninguna restricción algorítmica, la mayoría de estos algoritmos consisten en ir aprendiendo modelos débiles iterativamente con respecto a una distribución. Cada vez que se añade un nuevo modelo débil, los datos son re-calibrados: los ejemplos que estaban correctamente clasificados pierden peso y los ejemplos incorrectamente clasificados ganan peso.

Tipos de algoritmos *boosting*

Existen diversos algoritmos de tipo *boosting*. El algoritmo original fue propuesto por Yoav Freund y Robert Schapire en 1997, el cual era no adaptativo y no explotaba de forma completa la ventaja de los modelos débiles. Posteriormente, Freund y Schapire desarrollaron el algoritmo *AdaBoost*, un algoritmo de *boosting* adaptativo que ganó el prestigioso premio Gödel.

La principal variación entre los algoritmos de *boosting* es el **método de ponderar los datos de entrenamiento y las hipótesis**. *AdaBoost* es muy popular e históricamente el primero que fue capaz de adaptarse a los modelos débiles. Sin embargo, hay más algoritmos de *boosting* como: el *LPBoost*, *BrownBoost*, *TotalBoost*, *LogitBoost* y *xgboost*.

Gradient Boosting (Simple Version)

(Why is it called “gradient”?)
(Answer next slides.)

(For Regression Only)

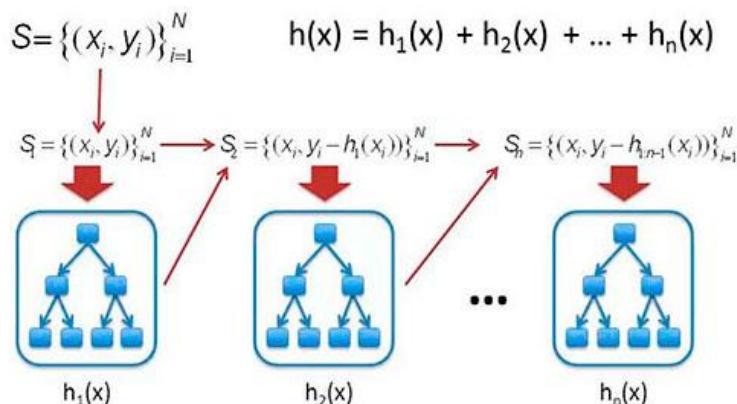


Figura 1. Ejemplo ilustrativo del funcionamiento de gradiente *boosting* para problemas de regresión. Fuente:
<http://statweb.stanford.edu/~jhf/ftp/trebst.pdf>

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \dots, B$, repeat:
 - 2.1 Fit a tree \hat{f}^b with d splits ($d + 1$ terminal nodes) to the training data (X, r) .
 - 2.2 Update \hat{f} by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$

- 2.3 Update the residuals,



$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x).$$

Figura 2. Algoritmo *boosting*. Fuente: James et. al., 2017.



Accede al vídeo «Resumen del tema» a través del aula virtual



Accede a los ejercicios de autoevaluación a través del aula virtual

7.5. Referencias bibliográficas

Breiman, L. (1998). Bagging predictors. *Machine Learning*, 24(2), 123-140.

Breiman, L. (1999). Pasting Small votes for classification in large databases and on-line. *Machine Learning*, 36(1), 85-103.

Freund, Y. y Schapire, R. (1997). A decisión-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119-139.

James, G., Witten, D., Hastie, T. y Tibshirani, R. (2017). *An Introduction to Statistical Learning with Applications in R*. Springer.

Kam, T. (1998). The random subspace method for constructing decision forests. *Pattern Analysis and Machine Intelligence*, 20(8), 832-844.

Louppe, G. y Geurts, P. (2012). Ensembles on Random Patches. *Machine Learning and Knowledge Discovery in Databases*, 346-361.

Quinlan, R. (2006). Bagging, boosting and C4.5. University of Sidney. Recuperado de <http://www.cs.ecu.edu/~dingq/CSCI6905/readings/BaggingBoosting.pdf>

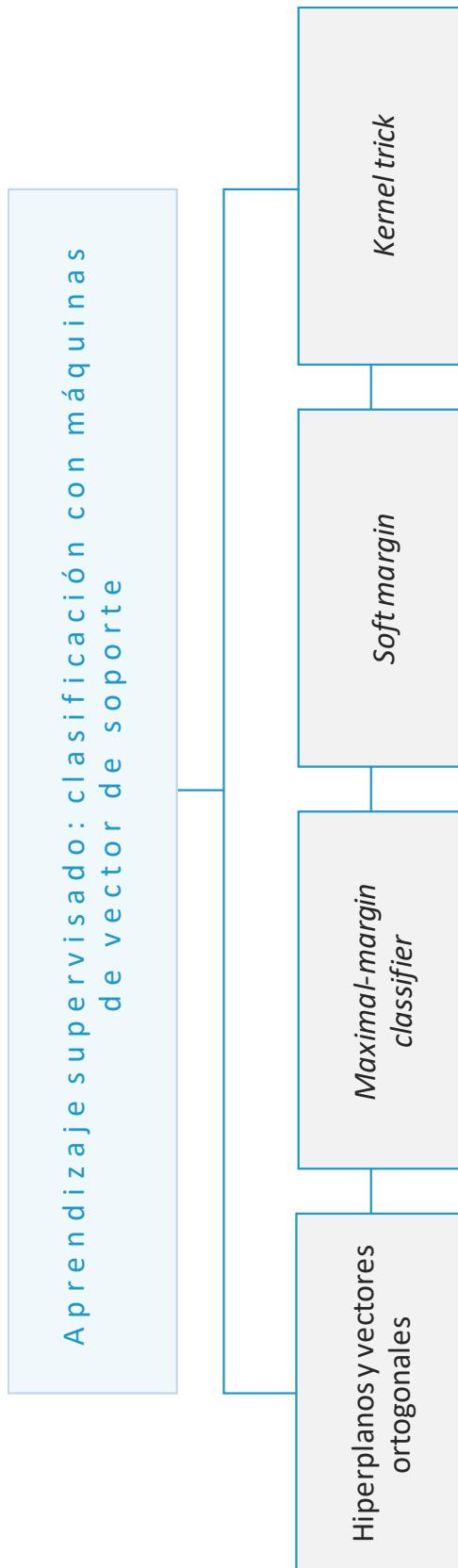
Aprendizaje Automático

Aprendizaje supervisado: clasificación con máquinas vector de soporte

Índice

Esquema	3
Ideas clave	4
8.1. Introducción y objetivos	4
8.2. Introducción a las máquinas de vector de soporte: hiperplanos	4
8.3. Separando por hiperplanos	8
8.4. <i>Maximal-margin classifier</i>	9
8.5. Referencias bibliográficas	17

Esquema



8.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos» a través del aula virtual

En este tema vamos a estudiar las máquinas de vector de soporte y cómo se

pueden aplicar en problemas de clasificación dentro del aprendizaje supervisado.

Las **máquinas de vector de soporte** han sido el primer modelo que utilizaba un enfoque no probabilístico para realizar la estimación de datos no observados.

- ▶ En primer lugar, vamos a ver los conceptos geométricos necesarios para entender el funcionamiento y la base que hay por detrás de las máquinas de vector de soporte.
- ▶ En segundo lugar, veremos los problemas existentes al separar por hiperplanos.
- ▶ A continuación, veremos cómo solucionar el problema de separar por hiperplanos.
- ▶ Finalmente, las técnicas conocidas como *kernel trick*.

8.2. Introducción a las máquinas de vector de soporte: hiperplanos



Accede al vídeo «Introducción a las máquinas de vector de soporte: hiperplanos» a través del aula virtual

Las máquinas vectores de soporte conocidas como *support vector machines* fueron inventadas por el científico de origen ruso Vladimir Vapnik y su equipo en el año 1963.

La gran contribución de Vapnik al campo del aprendizaje automático ha consistido precisamente en proponer uno de los **primeros modelos de estimación/predicción que no está basado en ningún modelo probabilístico**, lo cual es un gran cambio de mentalidad. Puesto que todos los modelos que se usaban hasta ese momento estaban basados en la teoría estadística y de probabilidad.

Sin embargo, las **máquinas de vector de soporte** (SVM) se pueden considerar el primer modelo que utiliza un enfoque completamente distinto y que está **basado en un modelado geométrico** y que se resuelve mediante un problema de optimización con restricciones.

El objetivo de las redes de vectores de soporte es buscar un plano que separe las clases en *feature space*. Por *feature space* se entiende un nuevo espacio de dimensiones diferente (por lo general con un mayor número de dimensiones) al espacio original.

Debido a que este objetivo de buscar un plano que separe las clases por lo general no es posible obtener, en las SVM se proponen las siguientes modificaciones:

- 1.** Relajar la definición de «separar».
- 2.** Mejorar y enriquecer el *feature space* para que la separación sea posible.

Antes de entrar en detalle en el funcionamiento de las máquinas de vector de soporte es necesario definir los conceptos de hiperplanos y vectores ortogonales.

Hiperplanos

En un espacio de dos dimensiones el hiperplano es una recta. Pero, un hiperplano se puede definir para un espacio de p dimensiones. La ecuación general del hiperplano para un espacio de p dimensiones es:

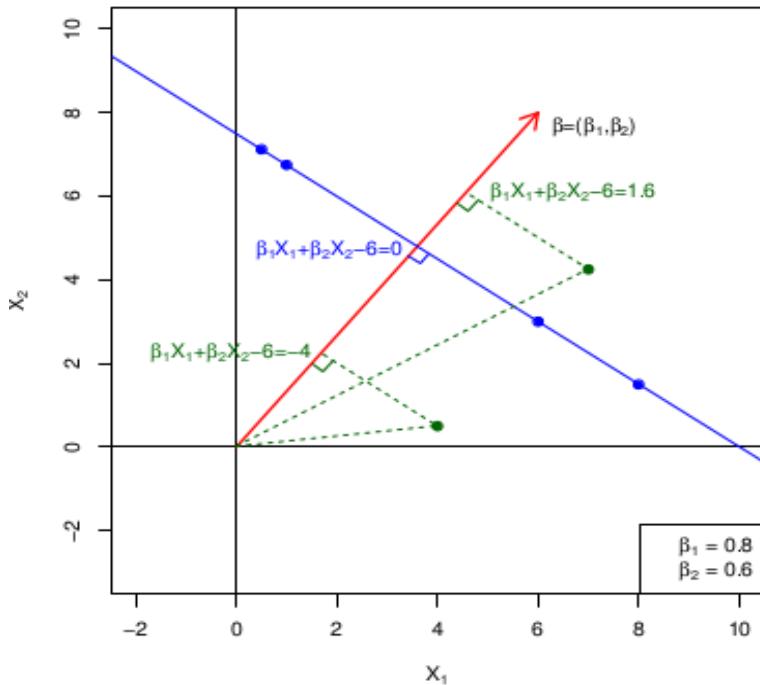
$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + B_p X_p = 0$$

donde el vector:

$$\beta = (\beta_1, B_2, \dots, B_p)$$

Se llama vector normal, y β es un vector unitario en el cual la suma de los cuadrados es 1. Este vector apunta a una dirección ortogonal a la superficie del hiperplano (vector de color rojo de la figura 1.).

Si proyectamos cada uno de los puntos sobre el vector normal, los puntos que caen sobre el hiperplano tienen valor 0 al proyectarse. Los puntos por encima del hiperplano tienen un valor positivo, los puntos por debajo del hiperplano un valor negativo y además estos valores son mayores cuanto más lejanos están del hiperplano. Es decir, el valor que se obtiene al proyectar los puntos sobre el vector normal es proporcional a la distancia de los puntos al hiperplano. Esta característica geométrica hace posible el uso de las máquinas de vector de soporte para buscar los patrones necesarios.



Gráfica 1. Representación de un hiperplano, un vector normal y la distancia de los puntos proyectados en el vector normal. Fuente:

<https://lagunita.stanford.edu/courses/HumanitiesSciences/StatLearning/Winter2016/about>

La cuestión es que el **hiperplano** busca separar dos conjuntos de puntos en dos regiones distintas. Por ejemplo, en un problema de clasificación el hiperplano puede separar aquellos puntos que corresponden a personas con un tumor determinado de aquellas personas sanas.

Sin embargo, para un conjunto de puntos determinados existen múltiples hiperplanos posibles que nos separan los puntos en dos regiones.

El objetivo de las máquinas de vector de soporte es buscar un hiperplano que separe las clases en *feature space*. Por *feature space* se entiende un nuevo espacio de dimensiones diferente al espacio original.



Accede a los ejercicios de autoevaluación a través del aula virtual

8.3. Separando por hiperplanos



Accede al vídeo «Separando por hiperplanos. *Kernel trick*» a través del aula virtual

Dado una serie de puntos en un espacio geométrico que se desea clasificar, se puede establecer que aquellos puntos que al proyectar sobre el vector normal a un hiperplano determinado sean > 0 , correspondan a una clase y aquellos que sean < 0 , a otra clase.

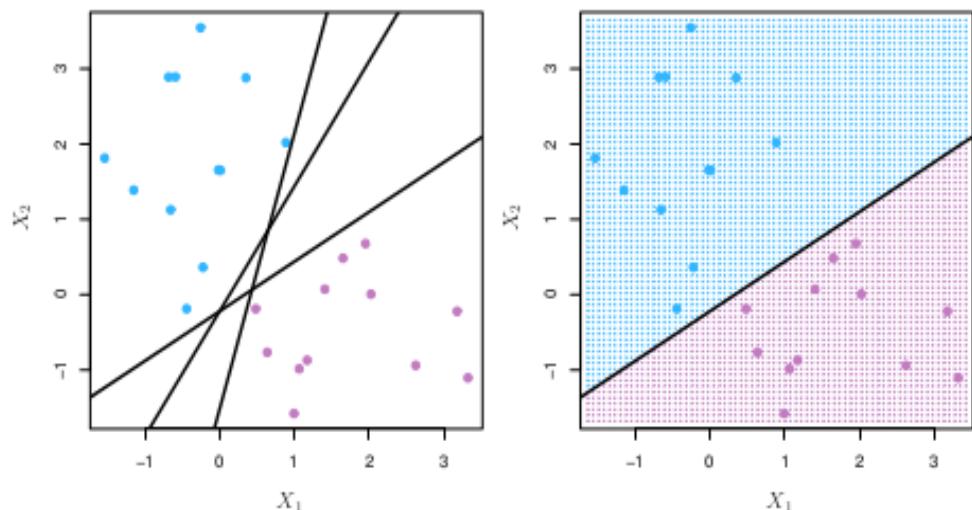
Es decir, de forma matemática:

$$\text{Si } f(X) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \dots + \beta_p X_p \text{ entonces}$$

$f(X) > 0$ establece los puntos en un lado del hiperplano

$f(X) < 0$ en el otro

Sin embargo, lo habitual es encontrarse en la situación en la cual existen múltiples hiperplanos posibles, y por tanto es necesario decidir ¿cuál de ellos establecer?



Gráfica 2. Ejemplo de tres hiperplanos posibles para separar dos conjuntos de clase (izquierda). En el caso de elegir un hiperplano los puntos se clasifican en función de cada una de las regiones sombreadas de rojo y azul (derecha). Fuente: James et.al., 2013.



Accede a los ejercicios de autoevaluación a través del aula virtual

8.4. Maximal-margin classifier, soft margin y kernel tricks



Accede al vídeo «*Maximal-margin classifier. Soft margin. Kernel trick*» a través del aula virtual

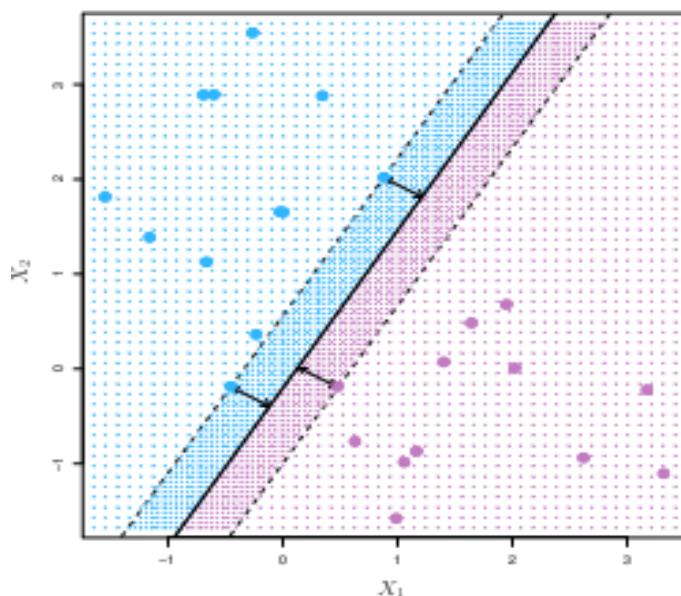
En el caso de un problema de clasificación binaria, de todos los hiperplanos posibles es necesario buscar aquel que nos proporciona la mayor diferencia entre las dos clases, lo cual se traduce en la mayor distancia entre los puntos que pertenecen a una clase y a otra. La hipótesis que hay detrás de esto, es porque suponemos que este hiperplano será el que tendrá una mayor distancia en el conjunto de test y en las predicciones futuras.

Esta situación se puede modelar como un problema de optimización con restricciones donde es necesario maximizar el margen y, matemáticamente, se define:

$$\begin{aligned} & \text{maximize } M \\ & \beta_0, \beta_1, \dots, \beta_p \\ & \text{subject to } \sum_{j=1}^p \beta_j^2 = 1 \\ & y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M \\ & \text{for all } i = 1, \dots, N \end{aligned}$$

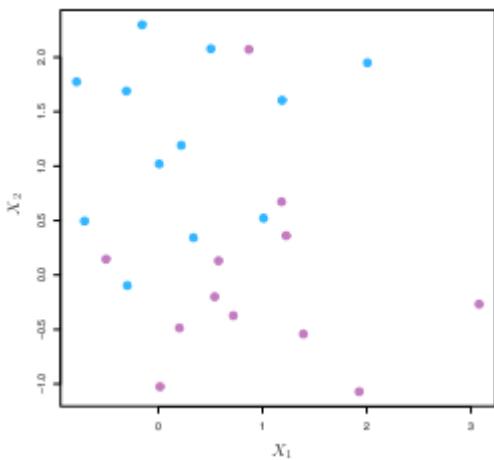
Maximal-margin classifier

En la siguiente imagen se muestra de forma gráfica el concepto donde la línea continua resaltada es el *maximal-margin classifier*, y se muestran dos bandas con líneas discontinuas que contienen la distancia del hiperplano a los primeros puntos de cada una de las clases, el objetivo es maximizar la distancia de estas dos bandas.



Gráfica 3. Ejemplo del hiperplano con una separación óptima entre dos conjuntos de puntos de clases diferentes. Fuente: James et.al., 2013.

El principal problema con esta separación óptima del hiperplano es que **los datos habitualmente no son linealmente separables con una recta** en el caso de un espacio de dos dimensiones, como se observa en la siguiente gráfica, donde es imposible separar los puntos de una clase de los de otra.



Gráfica 4. Ejemplo de un espacio de dos dimensiones donde los puntos no son linealmente separables.

Fuente: James et.al., 2013.

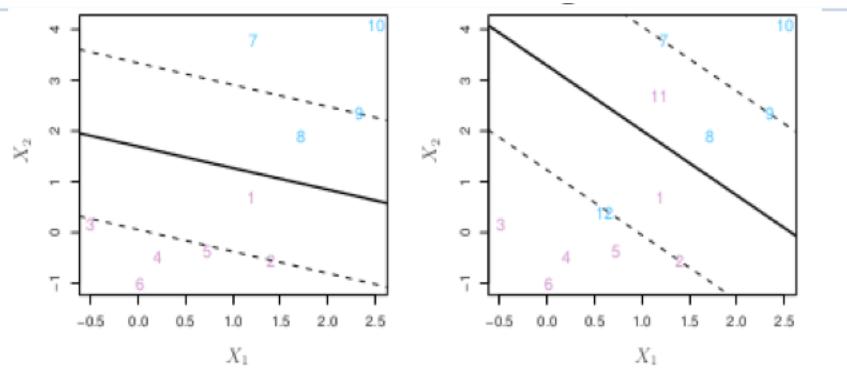
Esta situación produce una solución pobre para el clasificador *maximal-margin*. Por tanto, el clasificador vector de soporte maximiza un *soft margin*.

Soft margin

La solución al problema anterior en el cual los puntos no son linealmente separables es maximizar un *soft margin*. Matemáticamente el problema se define de la siguiente forma:

$$\begin{aligned}
 & \text{maximize } M \\
 & \beta_0, \beta_1, \dots, \beta_p, \epsilon_1, \dots, \epsilon_n \\
 & \text{subject to } \sum_{j=1}^p \beta_j^2 = 1 \\
 & y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M(1 - \epsilon_i) \\
 & \epsilon_i \geq 0, \sum_{i=1}^n \epsilon_i \leq C
 \end{aligned}$$

De forma gráfica, en la siguiente figura se observa:

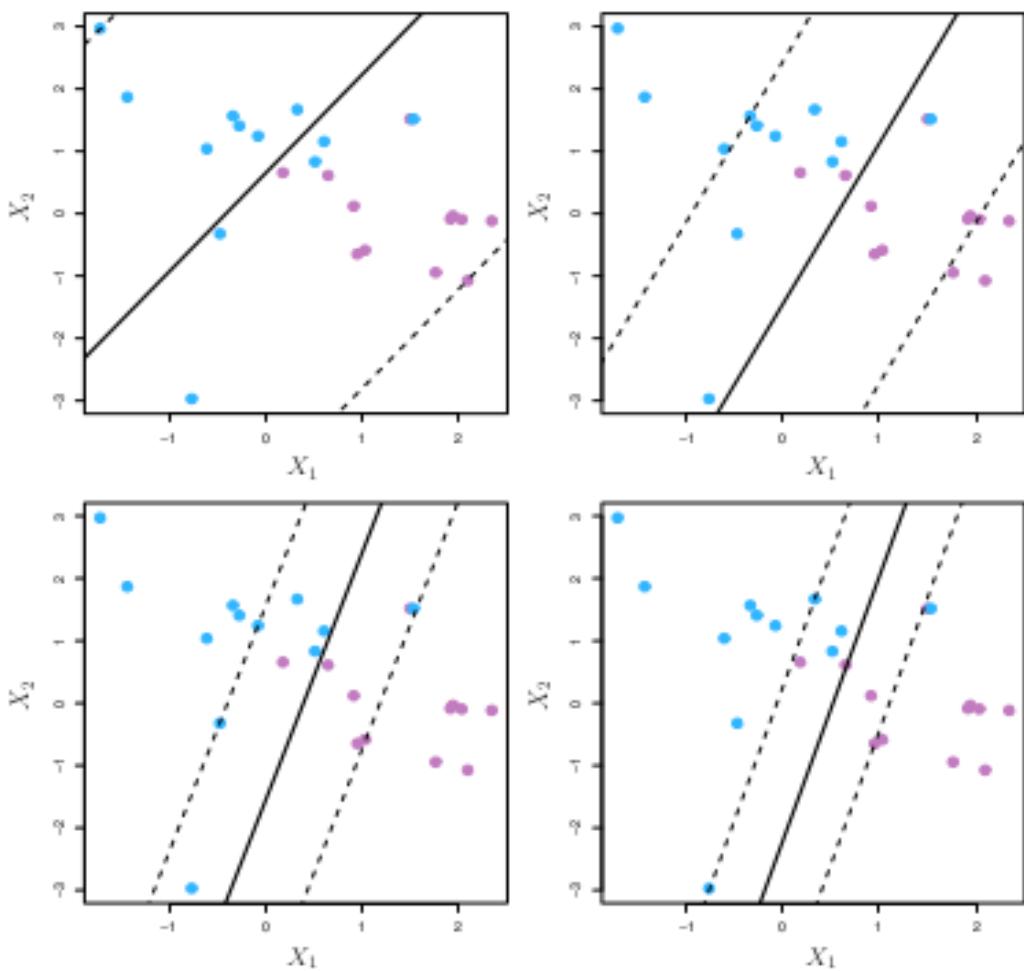


Gráfica 5. Ajuste de un clasificador de vector de soporte a pocos datos. El hiperplano se muestra con una línea sólida y los márgenes con una punteada. La imagen de la derecha muestra el efecto cuando aparecen dos nuevas observaciones (11 y 12) las cuales están en el lado incorrecto del hiperplano y de los márgenes.

Fuente: James et.al., 2013.

Modificando el parámetro C de la ecuación anterior, el cual se conoce como función de coste se puede hacer el margen más grande o pequeño. En concreto un valor de C más grande hace el margen más pequeño y a la inversa un valor C más pequeño hace el margen más grande.

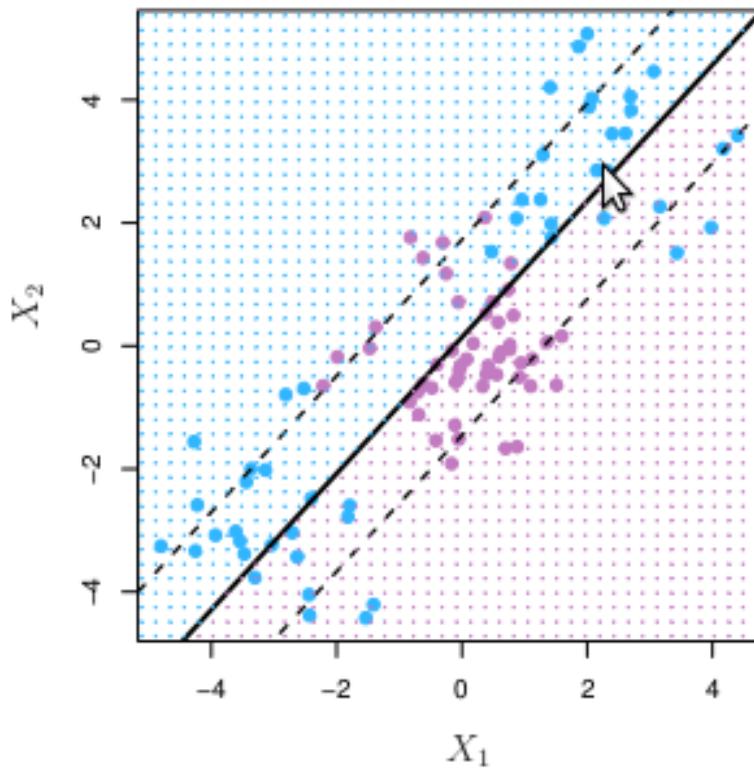
En la siguiente figura se muestra un hiperplano y los márgenes correspondientes en función de diferente valor de C .



Gráfica 6. Diferentes fronteras de decisión en los márgenes de una *soft-margin* en función de diferentes valores de C y utilizando los mismos puntos y el mismo hiperplano. El margen más grande es el de la gráfica superior izquierda y el más pequeño el de la inferior derecha. Fuente: James et.al., 2013.

Problema con las fronteras lineales

Muchas veces las fronteras lineales siguen sin funcionar, independientemente del valor de coste C que se utilice. Por ejemplo, en la siguiente gráfica se muestra un problema donde la separación necesaria entre los puntos de una clase y la otra es no lineal.



Gráfica 7. Ejemplo donde no es posible una separación lineal entre dos clases. Fuente: James et.al., 2013.

Expansión de variables

Para solucionar este problema donde no es posible realizar una separación entre dos clases, una opción es **incrementar el espacio de las variables por medio de transformaciones**, es decir pasar de un espacio **p-dimensional** a un espacio de D dimensiones $D > p$. Una vez realizada esta transformación, se debería ajustar un clasificador de vector de soporte en este nuevo espacio. El **objetivo es obtener fronteras de decisión no lineales sobre el espacio original**.

Por ejemplo, si tenemos datos de entrada en dos dimensiones (X_1, X_2) es posible utilizar el siguiente espacio de 6 dimensiones: $(X_1, X_2, X_1^2, X_2^2, X_1X_2)$ siendo por tanto la frontera de decisión:

$$\beta_0 + \beta_1X_1 + \beta_2X_2 + \beta_3X_1^2 + \beta_4X_2^2 + \beta_5X_1X_2 = 0$$

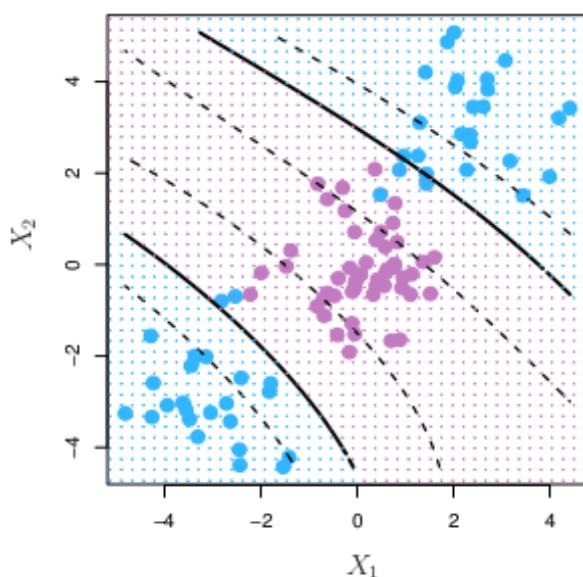
Esta transformación conlleva fronteras de decisión no-lineales en el espacio original.

Polinomios cúbicos

Utilizando una transformación en forma de polinomios cúbicos se puede pasar de 2 a 9 variables. De esta forma el clasificador en el nuevo espacio soluciona el problema de buscar los patrones en el espacio con menor dimensiones.

En la siguiente gráfica se muestra el resultado de las fronteras de separación obtenidas por medio de una transformación en polinomios cúbicos., resultado del cálculo de la siguiente ecuación:

$$B_0 + B_1X_1 + B_2X_2 + B_3X_1^2 + B_4X_2^2 + B_5X_1X_2 + B_6X_1^2 + B_7X_2^3 + B_8X_1X_2^2 + B_9X_1^2X_2 = 0$$



Gráfica 8. Ejemplo de las fronteras de decisión no lineales obtenidas por medio de una transformación obtenida con polinomios cúbicos. Fuente: James et.al., 2013.

Kernel trick

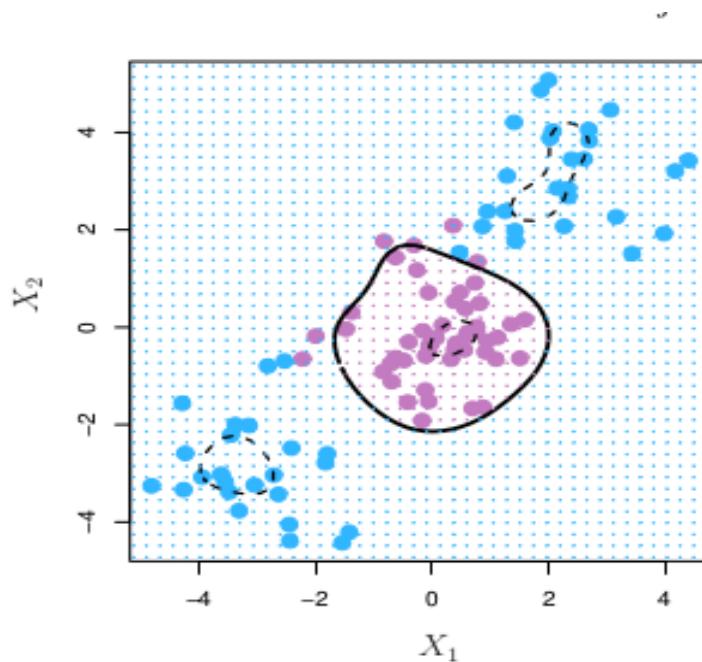
Las expansiones de las variables con polinomios, especialmente aquellos con grandes dimensiones, son computacionalmente costosos. Existe una solución más elegante y controlada de introducir no-linealidad que es utilizada en las máquinas vector de soporte por medio del uso de *kernels*. El *kernel* de un operador A denotado por *ker* es el conjunto de todos los vectores cuya imagen sea el vector nulo:

$$\ker A = \{ \underset{v}{\rightarrow} \in V : A \underset{v}{\rightarrow} = 0 \}$$

Los *kernels* se apoyan en concepto del producto vectorial de los vectores de soporte. Se trata de funciones que reciben dos vectores como parámetros. Uno de los *kernels* más utilizados es el de base radial que asume que el *feature space* es de altas dimensiones y se define con la siguiente función matemática:

$$K(x_i, x'_i) = \exp(-\gamma \sum_{j=1}^p (x_{ij} - x'_{ij})^2)$$
$$y_i(\beta_0 + \beta_1 x_{i1} + \dots + \beta_p x_{ip}) \geq M$$

El uso de *kernels* permite obtener fronteras de decisión no lineales por medio de transformaciones matemáticas sin necesidad de tener que realizar transformaciones con polinomios. En la siguiente figura se muestran las fronteras de decisión no lineales obtenidas por medio del uso de un *kernel* de base radial.



Gráfica. Ejemplo de las fronteras de decisión no lineales obtenidas por medio del uso de un *kernel* de base radial. Fuente: James et.al., 2013.



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Repaso del tema» a través del aula virtual

8.5. Referencias bibliográficas

Hastie, T., Tibshirani R., Friedman, J. (2011). *The Elements of Statistical Learning*. Second edition. Springer.

James, G., Witten, D., Hastie, T and Tibshirani, R. (2013). *An Introduction to Statistical Learning with Applications in R*. Springer.

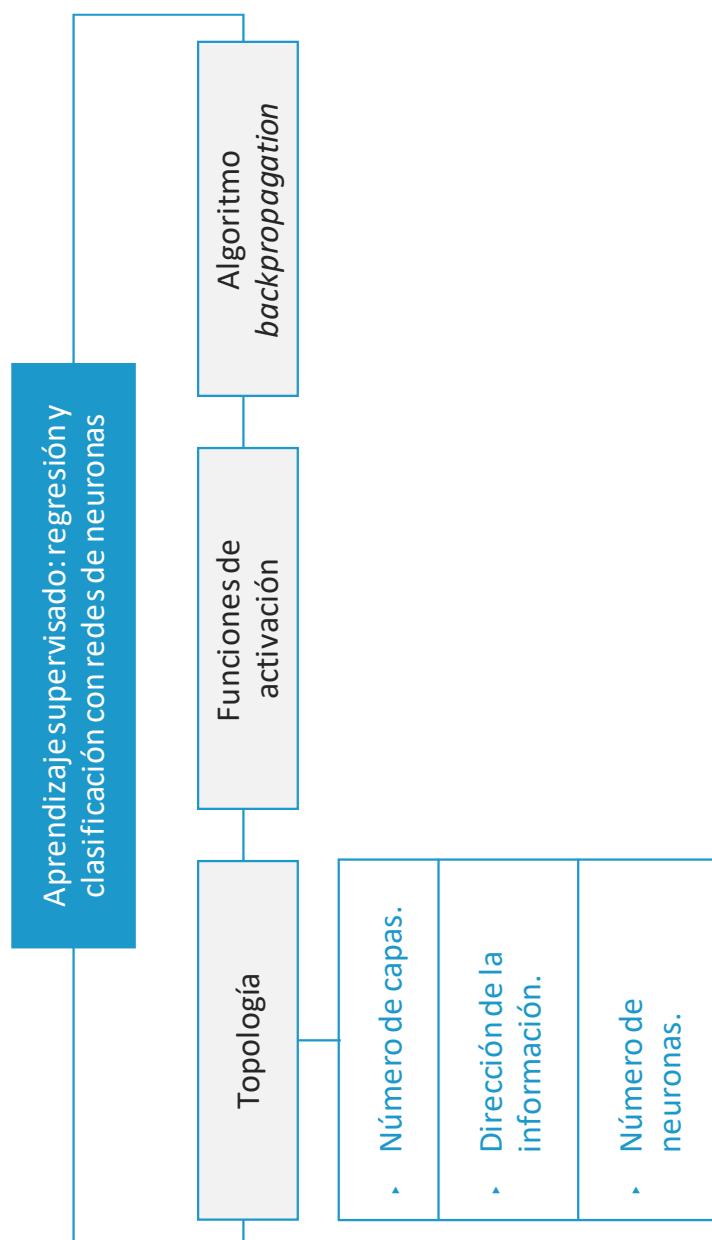
Aprendizaje Automático

Aprendizaje supervisado: regresión y clasificación con redes de neuronas

Índice

Esquema	3
Ideas clave	4
9.1. Introducción y objetivos	4
9.2. Neuronas artificiales	4
9.3. Arquitectura de una red de neuronas: capas, funciones de activación	7
9.4. Algoritmo de entrenamiento: <i>backpropagation</i>	13
9.5. Referencias bibliográficas	14

Esquema



9.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos: clasificador redes neuronales» a través del aula virtual

E

n este tema nos vamos a iniciar con los modelos basados en redes de neuronas artificiales. En primer lugar, vamos a ver en qué consisten las redes de neuronas artificiales.

A continuación, vamos a definir que son las funciones de activación, la topología de la red y el algoritmo de entrenamiento.

9.2. Neuronas artificiales



Accede al vídeo «Neuronas artificiales» a través del aula virtual

L

as redes de neuronas artificiales modelan la relación entre un conjunto de señales de **entrada** y una señal de **salida** utilizando un modelo que «simula» la forma en que el cerebro responde a estímulos. Para conseguir su propósito, las redes de neuronas se apoyan en neuronas artificiales o nodos los cuales están interconectados entre sí.

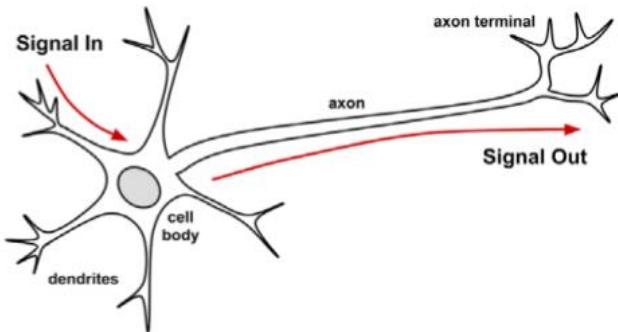


Figura 1. Ilustración de una neurona artificial. Fuente: Brett, 2013

Las **redes de neuronas** son modelos versátiles que se pueden aplicar a diferentes tareas de aprendizaje automático, como la clasificación y la predicción numérica. Se ha demostrado que una red de neuronas con al menos una capa oculta y las neuronas suficientes es una función aproximadora universal. En esencia, esto conlleva que esa red se puede utilizar para **aproximar cualquier función continua con una precisión arbitraria**. Esta teoría está basada en el teorema de aproximación universal de **George Cybenko (1989)**.

Una red de neuronas es un **aproximador universal** capaz de computar cualquier función matemática. No obstante, su aplicación es más apropiada en los siguientes escenarios: cuando los procesos que relacionan la entrada y la salida son muy ruidosos y es muy difícil encontrar el patrón, puesto que el proceso que relaciona la entrada con la salida es bastante complejo.

En una **neurona artificial** la señal de cada dendrita es ponderada (con los pesos w) de acuerdo a su importancia. En cada una de las neuronas de la red, las señales de entrada se suman en la neurona y la señal es enviada utilizando una **función de activación** sobre la aplicación de los pesos a las señales de entrada.

A continuación, se describe de forma matemática este proceso:

$$y(x) = f \left(\sum_{i=1}^n w_i X_i \right)$$

Donde la salida $y(x)$ de cada neurona es obtenida después de aplicar la función $f(\cdot)$.

Este mismo proceso se describe en el siguiente diagrama:

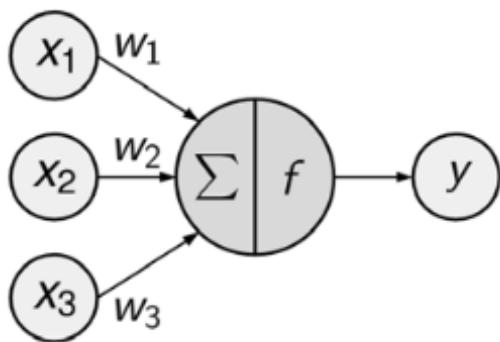


Figura 2. Diagrama descriptivo de una neurona artificial y las operaciones matemáticas que se llevan a cabo.

Fuente: Brett, 2013.

Es importante destacar que este proceso se lleva a cabo en cada una de las neuronas de la red de neuronas artificiales. Por tanto, las redes de neuronas utilizan neuronas artificiales para construir complejos modelos de datos. Estos modelos de datos pueden tener diferentes características dependiendo de la **función de activación, la topología de la red y el algoritmo de entrenamiento**.

Las redes de neuronas artificiales modelan la relación entre un conjunto de señales de entrada y una señal de salida utilizando un modelo que «simula» la forma en que el cerebro responde a los estímulos. Los modelos de redes de neuronas se diferencian dependiendo de la **función de activación, la topología de la red y el algoritmo de entrenamiento**.



Accede a los ejercicios de autoevaluación a través del aula virtual

9.3. Arquitectura de una red de neuronas: capas, funciones de activación



Accede al vídeo «Arquitectura de una red de neuronas: capas, funciones de activación» a través del aula virtual

E

xisten numerosas variantes y modificaciones de redes de neuronas en función de:

- ▶ El concepto de cultura. la **topología de la red (arquitectura)**, que describe el número de neuronas en el modelo, así como el número de capas y la forma en que están conectadas.
- ▶ La **función de activación** de cada una de las neuronas artificiales. Esta función transforma las entradas de una neurona en la señal que se propaga por la red.
- ▶ El **algoritmo de entrenamiento**, que especifica cómo se establece la conexión de los pesos para inhibir y/o excitar neuronas en función de la señal de entrada.

Topología de la red

La capacidad de una red de neuronas de aprender se debe a su **topología**, es decir, a los **patrones y estructuras de las neuronas conectadas**. Existen diversas topologías de red, pero todas ellas se pueden definir en función de:

- ▶ El número de capas.
- ▶ Si se permite que los datos de la red viajen hacia atrás, lo que se conoce como dirección de la información.
- ▶ El número de neuronas (nodos) en cada capa de la red.

La topología determina la **complejidad de las tareas que pueden ser aprendidas por la red**. Generalmente, redes más grandes y complejas son capaces de identificar patrones más sutiles y fronteras de decisión más complejas.

Sin embargo, la potencia de la red no es solo una función de su tamaño sino también de la forma en que las neuronas se conectan.

Número de capas

Las neuronas de entrada reciben las señales sin procesar directamente de los datos de entrada. Por regla general cada una de las neuronas de entrada es capaz de procesar una variable (*feature*) del conjunto de datos. El valor de esta variable se transforma por medio de la función de activación del nodo. A su vez las señales de los nodos de entrada se reciben en el nodo de salida, el cual utiliza su propia función de activación (que puede ser diferente de las anteriores) para realizar la estimación.

Los nodos de entrada y salida **se agrupan en capas**. Por ejemplo, en la siguiente figura se muestra una red de neuronas con 0 capas ocultas, pues la entrada está conectada directamente a la neurona de la capa de salida.

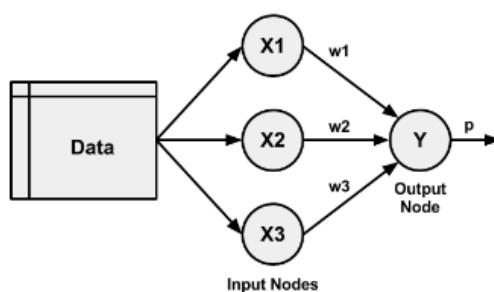


Figura 3. Ejemplo de una red de neuronas sin ninguna capa oculta, con 3 neuronas en la capa de entrada y una neurona en la capa de salida. Fuente: Brett, 2013.

Una red de neuronas multicapa añade una o más capas ocultas que procesan las señales de entrada antes de alcanzar el nodo de salida.

La mayoría de las redes multicapa son *fully connected* lo cual implica que cada nodo en una capa se conecta a todos los nodos en la capa siguiente como se muestra en la siguiente figura.

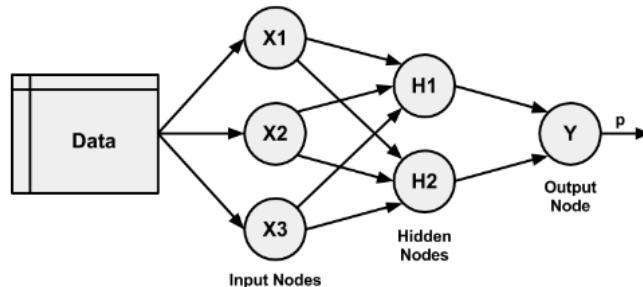


Figura 4. Ilustración de una red de neuronas multi-capa (1 capa oculta) *fully connected*.

Fuente: Brett, 2013.

Dirección de información

Las redes en las que los datos viajan de una conexión a otra hasta alcanzar las capas de salida se conocen con el nombre de *feedforward networks*. A pesar de la limitación del flujo de información las redes *feedforward* proporcionan mucha flexibilidad.

Por otro lado, las redes de neuronas recurrentes, *recurrent network*, proporcionan un *feedback* que permiten que las señales viajen en los dos sentidos por medio de bucles. Esta propiedad les permite aprender patrones más complejos.

La adición de una memoria a corto plazo (con un *delay*) permite la capacidad de entender secuencias de eventos en el tiempo. En la siguiente figura se muestra un ejemplo de una red recurrente con memoria.

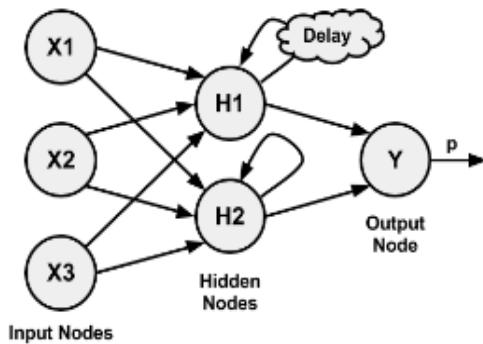


Figura 5. Ejemplo de una arquitectura de red recurrente con memoria.

Fuente: Brett, 2013.

Número de neuronas (nodos)

El número de neuronas de la capa de **entrada** viene determinado por el número de variables de los datos de entrada. El número de neuronas de la capa de **salida** viene determinado por el número de salidas a modelar o el número de niveles de la clase.

No hay una regla para determinar el número de neuronas de cada capa. El número apropiado depende del número de variable de entrada, la cantidad de datos de entrenamiento, la cantidad de datos con ruido y la complejidad de la tarea de aprendizaje.

En general, redes más complejas con un mayor número de conexiones permiten aprender problemas más complejos.

La mejor práctica es empezar con pocas neuronas e ir incrementando el número de forma gradual.

Funciones de activación

La función de activación es el mecanismo por el cual las neuronas artificiales procesan la información y esta se propaga por la red.

La señal de entrada se agrega y si supera un umbral determinado la señal atraviesa la neurona. Este umbral se conoce con el nombre de *threshold activation function*. Por ejemplo, la función de activación unitaria se activa cuando la suma de la señal de entrada es > 0 .

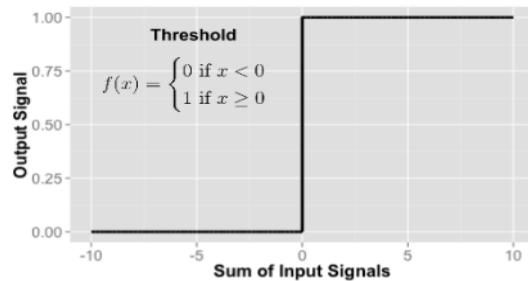


Figura 3. Gráfico ilustrativo de la función de activación unitaria.

Fuente: Brett, 2013.

Una de las funciones de activación más utilizada es la función *sigmoide*. Esto se debe a que la función es **diferenciable** lo que implica que se puede calcular la derivada a lo largo de todo el rango de entrada y simplifica el proceso de entrenar la red.

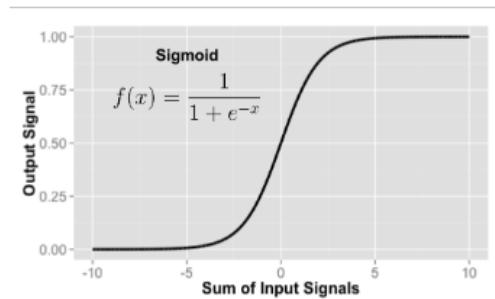


Figura 4. Gráfico ilustrativo de la función de activación *sigmoide*.

Fuente: Brett, 2013.

Existen muchas otras funciones de activaciones disponibles como pueden ser la función de activación lineal, lineal saturada, tangente hiperbólica y Gausiana.

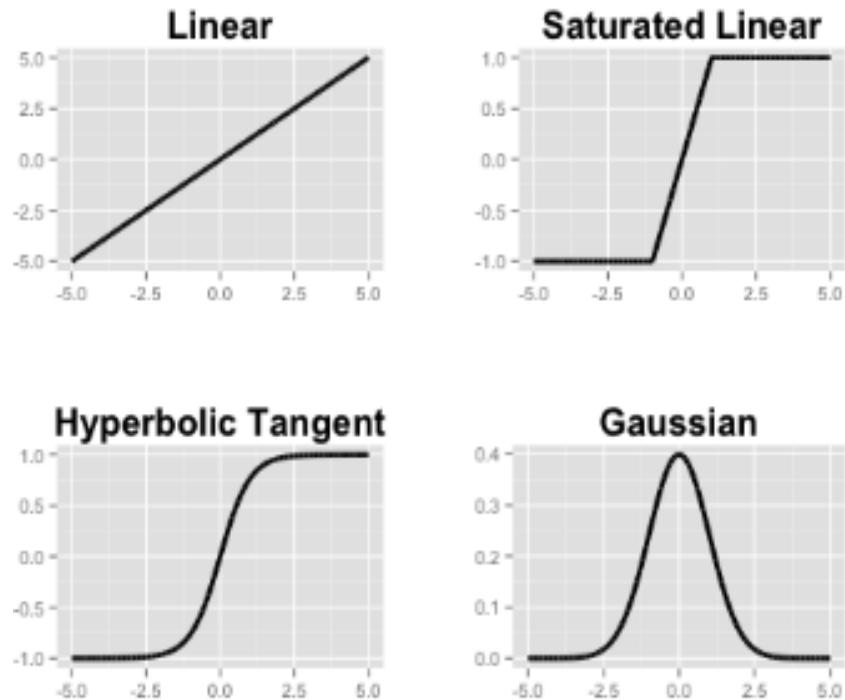


Figura 5. Gráfico ilustrativo de otra serie de funciones de activación.

Fuente: Brett, 2013.

La principal diferencia entre las funciones de activación anteriores es el rango de la señal de salida, el cual puede variar entre $(0, 1)$, $(-1, +1)$ y $(-\infty, +\infty)$. La elección de la **función de activación** sesga la red de neuronas haciendo que trate mejor ciertos tipos de datos.

Por ejemplo, una función de activación lineal da lugar a una red de neuronas similar a un modelo de regresión lineal, mientras que una **función de activación gaussiana da lugar a un modelo de red de base radial (Radial basis function)**.

Para muchas funciones de activación el rango de los valores de entrada es muy estrecho. Por ejemplo, en el caso de la *sigmoide* la señal de salida siempre es $0/1$ para valores por debajo o encima de $-5/+5$. Por este motivo es necesario normalizar los datos de entrada antes de entrenar y a la hora de predecir con una red de neuronas.



Accede a los ejercicios de autoevaluación a través del aula virtual

9.4. Algoritmo de entrenamiento: *backpropagation*



Accede al vídeo «Algoritmo de entrenamiento: *backpropagation*» a través del aula virtual

El proceso de entrenamiento de una red de neuronas consiste en encontrar el valor óptimo de los pesos. Se trata de una tarea costosa computacionalmente. El algoritmo utilizado para el entrenamiento de la red de neuronas se conoce con el nombre de ***backpropagation***. Este algoritmo proporciona una estrategia eficiente para entrenar las redes. El algoritmo itera en ciclos, llamados *epochs*, utilizando dos fases en cada ciclo:

1. *Forward*: las neuronas se activan en secuencia desde la capa de entrada a la capa de salida aplicando los pesos de cada neurona y la función de activación.
2. *Backward*: la señal de salida de la red se compara con el valor real. El error se propaga hacia atrás en la red para modificar los pesos entre las neuronas y reducir errores futuros.

Para determinar en cuanto se deben de modificar los pesos de una red se utiliza una técnica llamada descenso del gradiente (***gradient descent***). Los pesos se modifican siguiendo la dirección que produce una mayor reducción del error, utilizando para ello la derivada de la activación de cada neurona para identificar el gradiente de la dirección de los pesos futuros.

El algoritmo intenta modificar aquellos pesos que proporcionan una mayor reducción del error utilizando un parámetro conocido como ***learning rate***. Cuanto mayor es el ***learning rate*** más rápido el algoritmo desciende por los gradientes.



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Resumen del tema» a través del aula virtual

9.5. Referencias bibliográficas

Brett, L. (2013). *Machine Learning with R*. Packt.

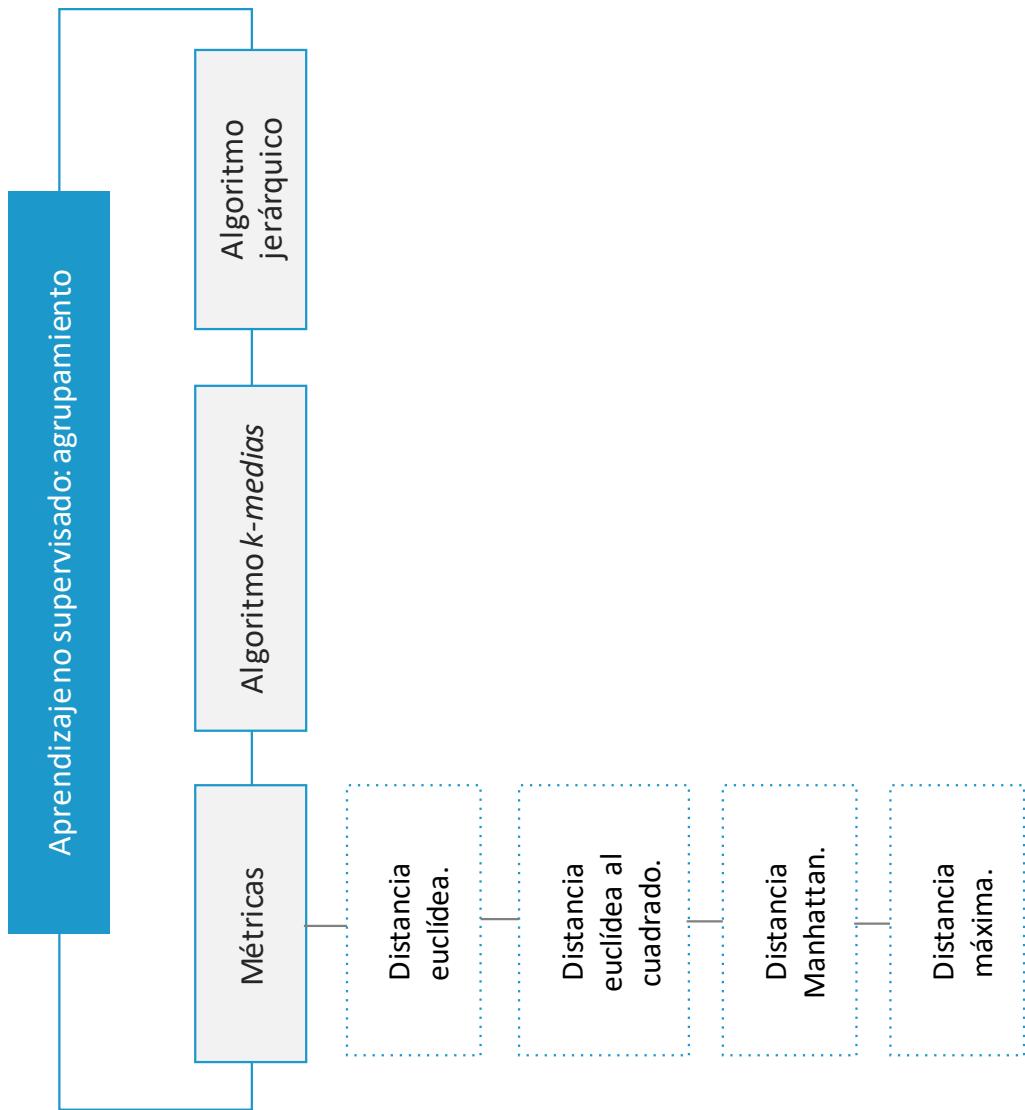
Aprendizaje Automático

Técnicas de aprendizaje no supervisado: agrupamiento

Índice

Esquema	3
Ideas clave	4
10.1. Introducción y objetivos	4
10.2. Introducción al aprendizaje no supervisado	4
10.3. Algoritmo de <i>k-medias</i>	6
10.4. Agrupamiento jerárquico	10
10.5. Referencias bibliográficas	12

Esquema



10.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos» a través del aula virtual

En este tema se hace énfasis en los algoritmos de aprendizaje no supervisado.

- ▶ En primer lugar, se realiza una introducción a estos algoritmos y se describen las dos grandes familias.
- ▶ A continuación, se detalla el algoritmo de *k-medias* y se describe su funcionamiento.
- ▶ Posteriormente, se presentan los algoritmos basados en agrupamiento jerárquico.

10.2. Introducción al aprendizaje no supervisado



Accede al vídeo «Aprendizaje no supervisado» a través del aula virtual

En el aprendizaje supervisado dadas unas series de etiquetas se «aprendía» un patrón para obtenerlas. En el caso del aprendizaje no supervisado el problema consiste en probar y determinar la estructura existente en los datos, pero **sin utilizar una etiqueta previa**. Estos algoritmos, también se conocen con el nombre de **algoritmos de agrupamiento** o *clustering* puesto que agrupan instancias de los datos en función de las variables de los conjuntos de datos. Es decir, este tipo de algoritmos buscan patrones en los datos con el objetivo de encontrar agrupaciones en los datos.

Se utilizan cuando se **desconoce la estructura de los datos** puesto que no se tiene la variable objetivo de los datos. Por ejemplo, cuando se desconoce cuántos grupos de usuarios similares existen.

Estas técnicas dividen los datos en *clusters* o grupos similares. Pero esta división se lleva a cabo sin la necesidad de indicar las características de cada uno de estos grupos. Para este objetivo, las instancias de un grupo deben de ser muy similares entre sí, pero muy distintas entre los grupos. Por tanto, es necesario definir una medida de similitud entre los elementos.

Entre las medidas de similitud más comunes tenemos:

- ▶ La distancia euclídea:

$$d = \sqrt{\sum_i (a_i - b_i)^2}$$

- ▶ La distancia euclídea al cuadrado:

$$d = (a_i - b_i)^2$$

- ▶ La distancia manhattan:

$$d = \sum_i |a_i - b_i|^2$$

- ▶ La distancia máxima:

$$d = \max_i |a_i - b_i|$$

La clave de estos algoritmos consiste en **buscar buenas variables** capaces de distinguir entre las diferentes instancias o registros.

Los algoritmos de agrupamiento se pueden utilizar para diversos objetivos, tales como:

- ▶ Segmentación de mercado: agrupar clientes en diferentes segmentos del mercado.
- ▶ Análisis de redes sociales: similitud de usuarios.
- ▶ Organizar centros de datos en función de la carga de la red y la localización.
- ▶ Segmentar clientes entre grupos con patrones de compra similares.
- ▶ Detectar comportamiento anómalo, identificando grupos que caen fuera de los *clusters* habituales.
- ▶ Simplificar o resumir conjuntos de datos muy grandes.

Los **algoritmos de aprendizaje no supervisado** se pueden dividir en dos grandes grupos:

1. Agrupación: tienen definidos de antemano un número de grupos. Son algoritmos iterativos que comienzan con una asignación inicial y se van modificando siguiendo un criterio de optimización.
2. Jerárquicos: En cada iteración solo un objeto cambia de grupo y los grupos están anidados en los de los pasos anteriores. Si un objeto ha sido asignado a un grupo ya no vuelve a cambiar.



Accede a los ejercicios de autoevaluación a través del aula virtual

10.3. Algoritmo de *k-medias*



Accede al vídeo «Algoritmo de *k-medias*» a través del aula virtual

El algoritmo *k-medias* es el método de agrupamiento más utilizado. Entendiendo su funcionamiento podemos entender casi cualquier algoritmo de agrupamiento utilizado hoy en día.

El algoritmo **consiste en asignar cada uno de los n ejemplos a uno de los k clusters**, donde k es un número definido previamente. El objetivo es minimizar las diferencias entre los grupos de cada *cluster* y maximizar las diferencias entre *clusters*.

A menos que k y n sean extremadamente pequeños no es factible calcular los grupos óptimos entre todas las combinaciones posibles de ejemplos. En su lugar, el algoritmo utiliza un **proceso heurístico** para calcular la solución óptima.

El algoritmo comprende dos fases:

1. Asigna ejemplos a un conjunto inicial de k clusters.
2. Después actualiza las asignaciones ajustando los límites de los grupos de acuerdo con los ejemplos de cada *cluster*.

Este proceso de asignación y actualización **ocurre varias veces** hasta que los cambios no proporcionan mejoras en los *clusters*.

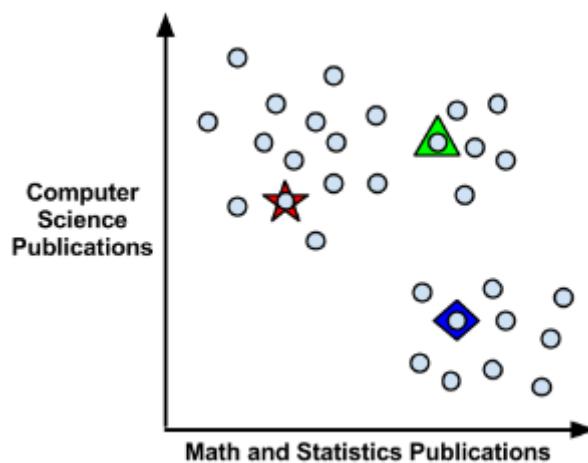
Debido a la naturaleza heurística del algoritmo los resultados pueden ser distintos en función de la inicialización del algoritmo. Sin embargo, si los resultados difieren mucho los unos de los otros puede indicar un problema. Por ejemplo, puede ocurrir que los datos no se puedan agrupar bien en k clusters.

El algoritmo *k-medias* considera que los valores de las variables son coordenadas en un espacio multi-dimensional.

Ejemplo de ejecución

A continuación, veamos un ejemplo de ejecución:

El algoritmo empieza eligiendo los k puntos iniciales.

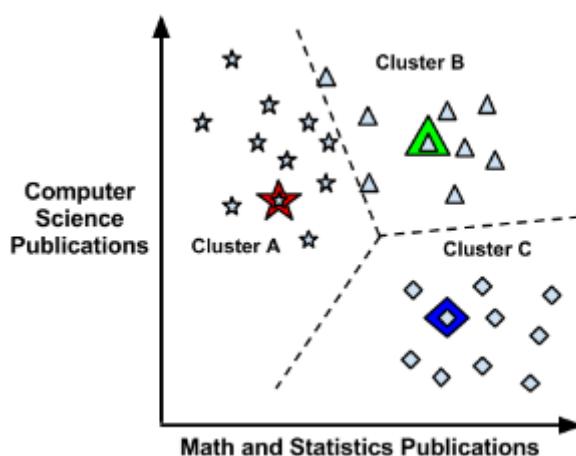


Gráfica 1. Elección inicial de tres puntos como centroides.

Fuente: Lantz, 2013.

Los puntos iniciales se suelen elegir al azar, en este caso se eligen tres ejemplos al azar. Otras opciones son elegir puntos que pueden ocurrir en cualquier intervalo del espacio de las variables de entrada. Otra opción es asignar inicialmente de forma aleatoria cada punto a un *cluster*.

Después de elegir los puntos iniciales, los otros ejemplos se asignan al centroide del *cluster* más cercano de acuerdo a la función de distancia, esta función suele ser la distancia euclídea.

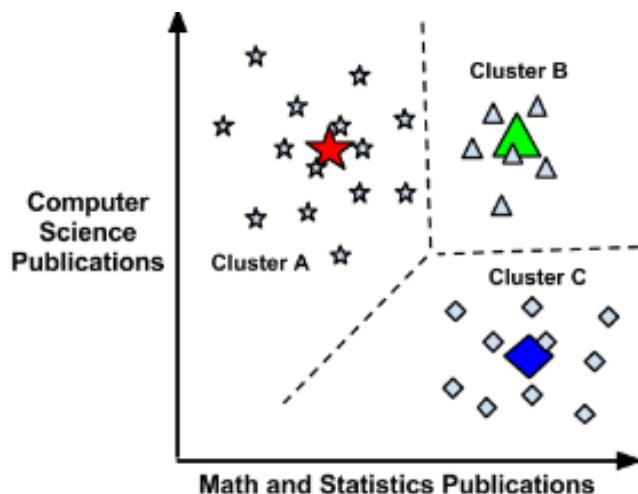


Gráfica 2. Asignación del resto de puntos a los centroides.

Fuente: Lantz, 2013.

Hay que tener en cuenta que debido a que estamos usando distancias, todos los datos deben de ser numéricos y normalizados antes de utilizarlos.

El primer paso de la actualización conlleva reubicar los centroides iniciales a una nueva posición calculada como la media de los puntos asignados al *cluster*. Como los límites de los centroides se han modificado es muy posible que haya que reasignar instancias a otros *clusters*.



Gr 3. Reajuste de los centroides en función de la asignación del resto de puntos.

Fuente: Lantz, 2013.

Elegir el valor de *K*

Elegir un buen valor de *K* requiere de cierto balance. Un número muy grande mejora la homogeneidad, pero a la vez sobre-ajusta los datos.

Idealmente existe un conocimiento *a priori* sobre el **número de grupos apropiado**. Por ejemplo, si estamos agrupando películas se puede elegir el valor de *k* que concuerde con los géneros existentes. Otras veces el número de *clusters* viene dada por los requisitos de negocio.

Sin conocimiento *a priori* se suele elegir un valor de $K = \sqrt{n}/2$. También se puede usar métricas para medir la homogeneidad *versus* la heterogeneidad.



Accede a los ejercicios de autoevaluación a través del aula virtual

10.4. Agrupamiento jerárquico



Accede al vídeo «Agrupamiento jerárquico» a través del aula virtual

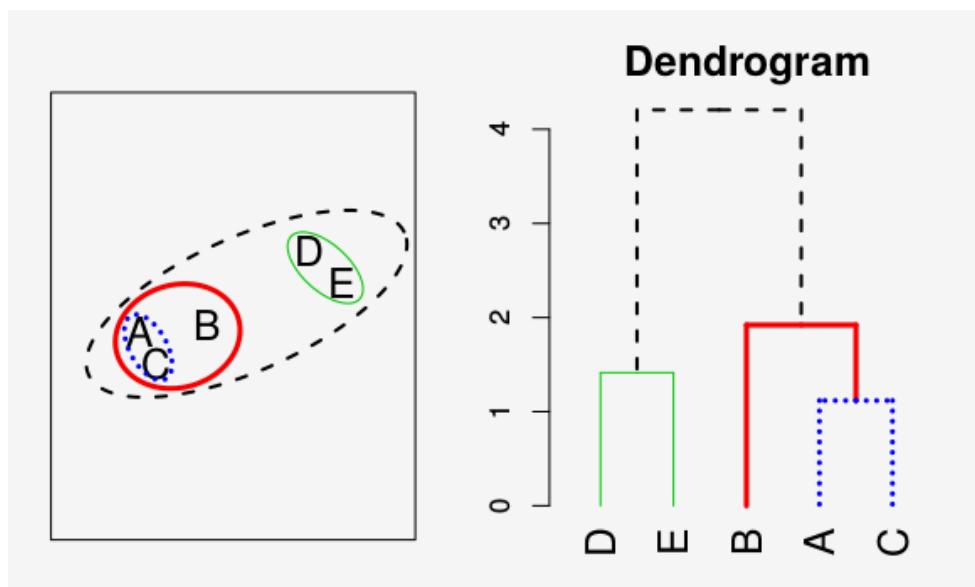
Los algoritmos de agrupamiento jerárquicos pueden ser de dos tipos:

1. Aglomerativos: se trata de métodos *bottom-up*, donde cada observación empieza en un *cluster* y los pares de *clusters* se combinan cuando se avanza hacia arriba en la jerarquía.
2. Divisivos: es un método *top-down* en donde todas las observaciones empiezan en un *cluster* y se van haciendo divisiones hacia abajo.

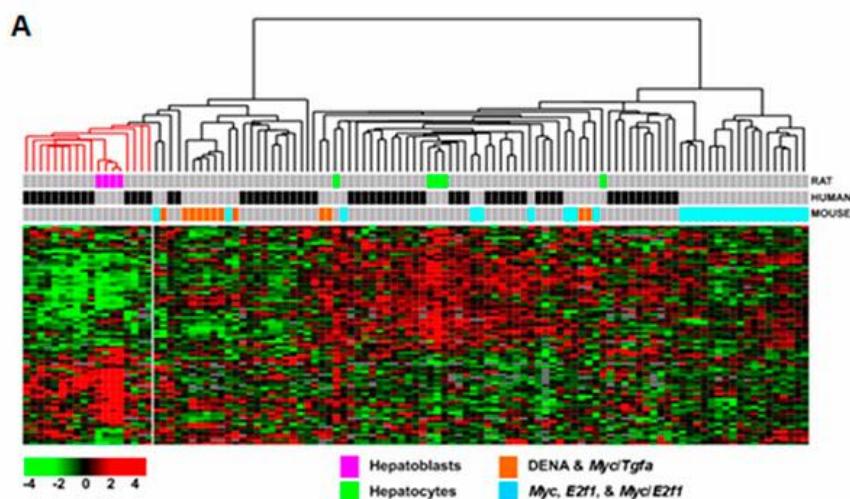
En el *clustering* jerárquico no conocemos de antemano cuántos *clusters* queremos. Para ello, obtenemos una **representación basada en árboles** llamada *dendrograma* que nos permite visualizar los grupos obtenidos para cada posible número de *clusters* de 1 a n .

El algoritmo de *clustering* jerárquico aglomerativo se puede resumir de la siguiente forma:

- ▶ Empezar con cada punto en su propio cluster.
 - Identificar los dos *clusters* más cercanos y combinarlos.
- ▶ Repetir.
- ▶ Terminar cuando todos los puntos estén en un *cluster*.



Gráfica 4. Ejemplo de un dendrograma y su correspondiente agrupamiento.



Gráfica 5. Ejemplo de un dendrograma y su mapa de calor de un *clustering* jerárquico. Fuente:

<https://ccrod.cancer.gov/confluence/display/COEICBG/Highlighted+Article+1>



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Resumen del tema» a través del aula virtual

10.5. Referencias bibliográficas

Lantz, B. (2013). *Machine Learning wiht R*. Packt Publishing.

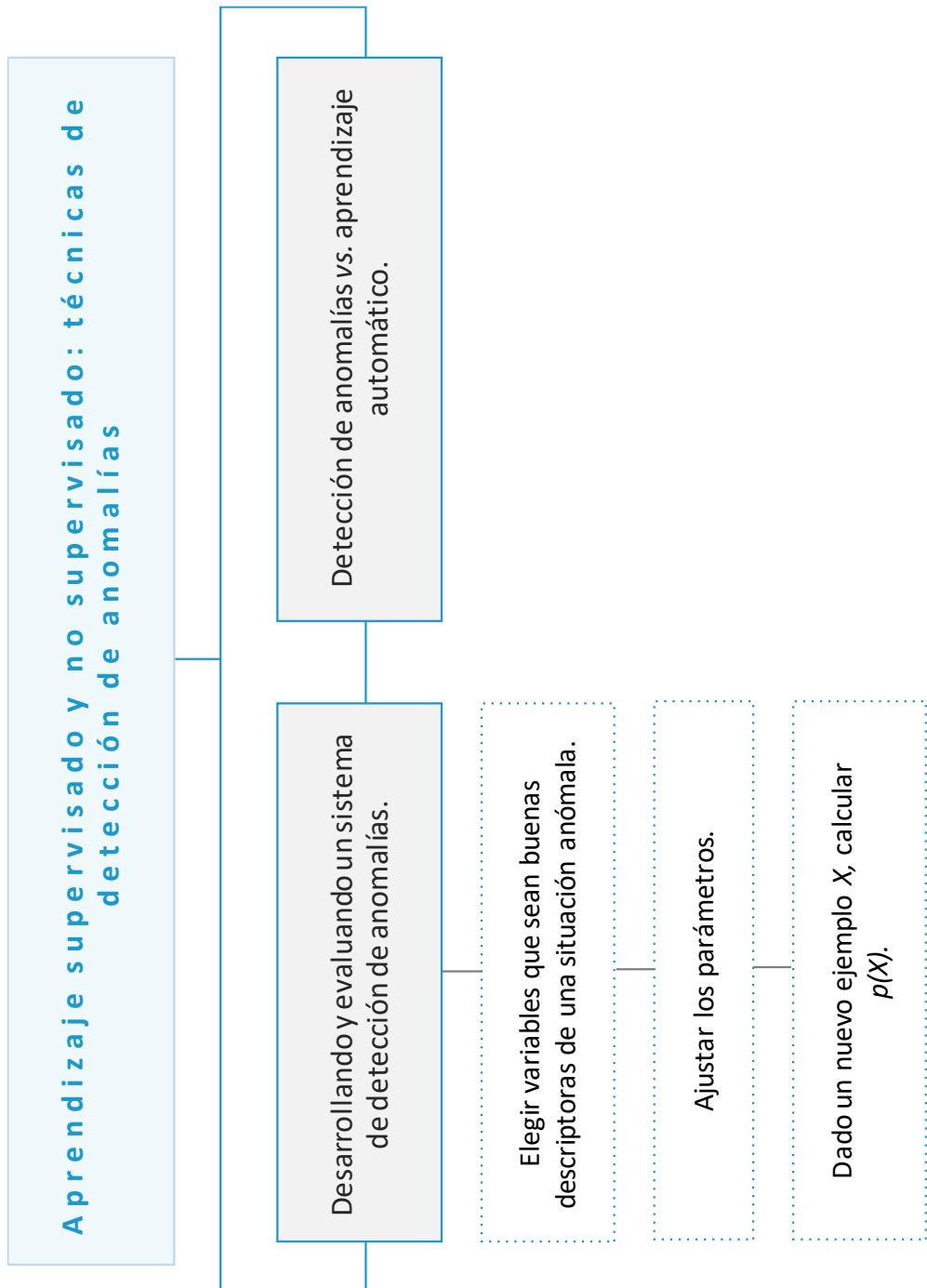
Aprendizaje Automático

Técnicas de detección de anomalías

Índice

Esquema	3
Ideas clave	4
1.1. Introducción y objetivos	4
11.2. Introducción a la detección de anomalías	4
11.3. Aplicación del aprendizaje automático a la detección de anomalías	8
11.4. Desarrollando y evaluando un sistema de detección de anomalías	12
11.5. Detección de anomalías <i>vs.</i> aprendizaje supervisado	14

Esquema



1.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos de detección de anomalías» a través del aula virtual

En este tema se desarrollan los conceptos de las **técnicas de detección de anomalías**. Estas técnicas también se conocen en la literatura con el nombre de detección de *outliers*. En esencia un *outlier* es un valor poco habitual y, por tanto, puede ser considerado una anomalía.

- ▶ En primer lugar, se introducen los métodos de detección de anomalías y su principal aplicación.
- ▶ A continuación, se describe cómo se pueden utilizar los métodos de aprendizaje supervisado para utilizarse en el ámbito de la detección de anomalías.
- ▶ Posteriormente, se cubren las pautas necesarias para desarrollar y evaluar un sistema de detección de anomalías.
- ▶ Finalmente, se describen las diferencias existentes entre el aprendizaje supervisado y los métodos de detección de anomalías.

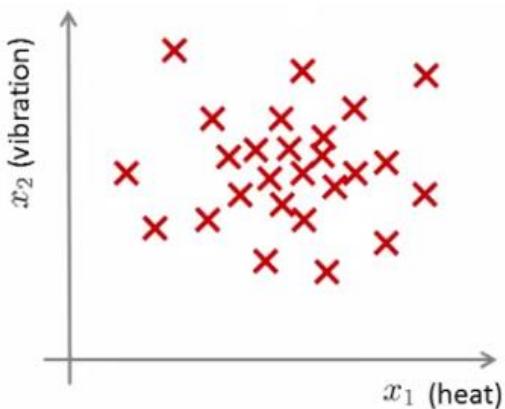
11.2. Introducción a la detección de anomalías



Accede al vídeo «Introducción a la detección de anomalías» a través del aula virtual

Los problemas de detección de anomalías son una **aplicación común del aprendizaje automático**. Se pueden ver como una posible solución a un problema de aprendizaje no supervisado, pero tienen también aspectos de aprendizaje supervisado.

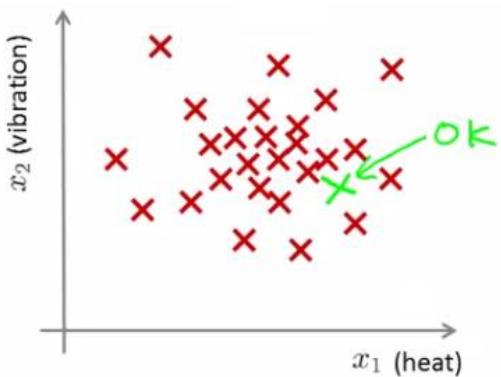
Pero, ¿qué es la detección de anomalías? Imagine que trabaja en una empresa que fabrica motores de aviones. A medida que los motores salen de la cadena de montaje, se realiza una fase de aseguramiento de la calidad en la cual se miden algunas características de los motores (ejemplo: calor generado y vibración). Supongamos que tenemos un conjunto de datos con m motores que han sido evaluados positivamente y han dado como resultado lo dibujado en la siguiente gráfica.



Gráfica 1. Distribución de valores de vibración y calor generado por motores. Fuente:

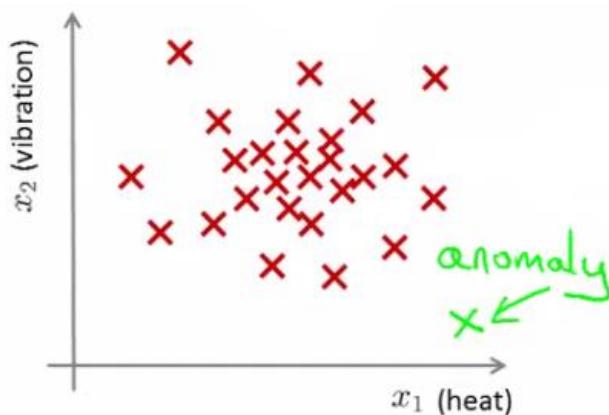
<https://es.coursera.org/learn/machine-learning>

A partir de ahora, al día siguiente se fabrica un nuevo motor y se utiliza un método de detección de anomalías para comprobar su correcto funcionamiento, comparando el de este motor nuevo con respecto a los motores previos. Si obtenemos una gráfica como esta:



Gráfica 2. Distribución de valores de vibración y calor generado por motores. Ejemplo de una instancia correcta. Fuente: <https://es.coursera.org/learn/machine-learning>

Lo más probable es que el motor funcione correctamente pues su comportamiento es muy similar al de motores previos. Sin embargo, si la gráfica fuera:



Gráfica 3. Distribución de valores de vibración y calor generado por motores. Ejemplo de una instancia anómala. Fuente: <https://es.coursera.org/learn/machine-learning>

Lo más probable es que el motor presente algún tipo de anomalía.

En este tipo de problemas partimos de un conjunto de datos que contiene registros normales, o bien la gran mayoría de ellos lo son. El objetivo es utilizar este conjunto como referencia y observar si existen nuevos ejemplos que son anómalos.

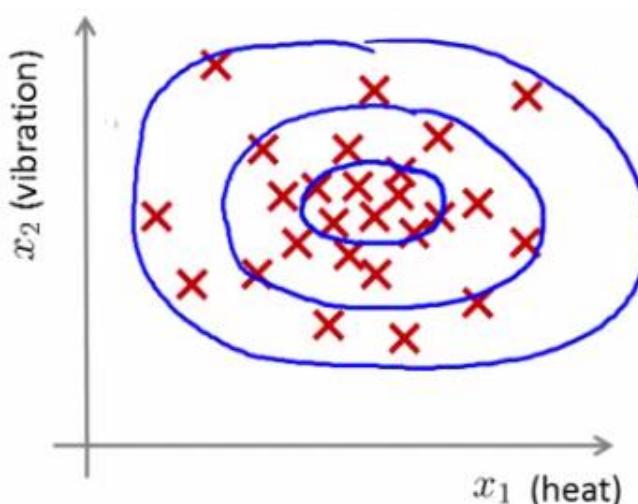
¿Cómo se realiza esta comprobación? En primer lugar, se utiliza el conjunto de entrenamiento para entrenar un modelo.

Este modelo responde a la pregunta: ¿cuál es la probabilidad de que el ejemplo x sea normal?

Una vez que se ha construido el modelo:

- ▶ Si $p(X_{test}) < \varepsilon$ se trata de una anomalía.
- ▶ Si $p(X_{test}) \geq \varepsilon$ se trata de un ejemplo normal.

Donde ε es un umbral de probabilidad definido en función de que nivel de certeza queramos tener. En el caso de un modelo en dos dimensiones, lo que estamos definiendo se puede representar gráficamente de la siguiente forma:



Gráfica 4. Distribución de valores de vibración y calor generado por motores. Fuente:
<https://es.coursera.org/learn/machine-learning>

Donde a medida que nos alejamos del centro, la probabilidad de que aparezcan ejemplos similares disminuye.

Aplicaciones

Las aplicaciones de los problemas de detección de anomalías son muy variadas. A continuación, se presentan dos ejemplos:

- ▶ Detección de fraude: se puede modelar a los usuarios en función de ciertos valores de su actividad como: localización del *login*, duración de tiempo *online*, frecuencia de gasto, etc. Utilizando este conjunto de datos se puede construir un modelo para generar el patrón de actividad habitual de los usuarios. Con este modelo se puede obtener la probabilidad de comportamiento «normal» para cada usuario y por tanto identificar usuarios anormales. Esto puede desencadenar acciones como bloquear el tráfico a determinados usuarios o automáticamente bloquear transacciones.
- ▶ Monitorización *data-center*: si tenemos un *data center* con muchos ordenadores, se puede construir un conjunto de datos con información sobre cada ordenador (uso de memoria, accesos al disco, carga de CPU, etc.). En el caso de que se observe un comportamiento anómalo de un ordenador posiblemente sea porque vaya a fallar.

La detección de anomalías combina las técnicas de aprendizaje supervisado para generar un modelo de valores normales y, posteriormente, se utiliza este modelo con nuevos registros para detectar valores anómalos o inusuales.



Accede a los ejercicios de autoevaluación a través del aula virtual

11.3. Aplicación del aprendizaje automático a la detección de anomalías



Accede al vídeo «Aplicación del aprendizaje automático a la detección de anomalías» a través del aula virtual

Para realizar la detección de anomalías se puede utilizar el siguiente algoritmo de aprendizaje automático. Dado un conjunto de m ejemplos de entrenamiento sin etiquetar:

$$Datos = \{x^1, x^2, \dots, x^m\}$$

Donde cada ejemplo es un vector de n dimensiones y, por tanto, tenemos n variables. Vamos a obtener la probabilidad de aparición de cada elemento x , que denotamos por $P(x)$. Nos interesa conocer cuáles son las variables con alta y baja probabilidad de aparición, si x es un vector, el modelo $P(x)$ se define como:

$$P(x) = p(x_1; \mu_1, \sigma_1^2) * p(x_2; \mu_2, \sigma_2^2) * \dots * p(x_n; \mu_n, \sigma_n^2)$$

Por tanto, multiplicamos la probabilidad de cada una de las variables y asumimos que cada una de ellas se distribuye de acuerdo a una distribución gaussiana. Es decir, obtenemos la distribución de probabilidad de la variable x_i dado μ_i y σ_i^2 utilizando una distribución gaussiana.

Por tanto, este modelo asume **independencia condicional de las variables**, aunque el algoritmo funciona si las variables son independientes o no. La fórmula anterior se puede escribir de forma compacta como:

$$P(x) = \prod_{j=1}^n p(x_j; \mu_j; \sigma_j^2)$$

El problema de estimar esta función se conoce también con el nombre de **estimación de densidad**.

Algoritmo

1. Elegir variables x_i que consideres son buenos indicadores del comportamiento anómalo.
2. Ajustar los parámetros $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

$$\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

3. Dado un nuevo ejemplo x , calcular $p(x)$:

$$P(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Tenemos una **anomalía** si $p(x) < \epsilon$

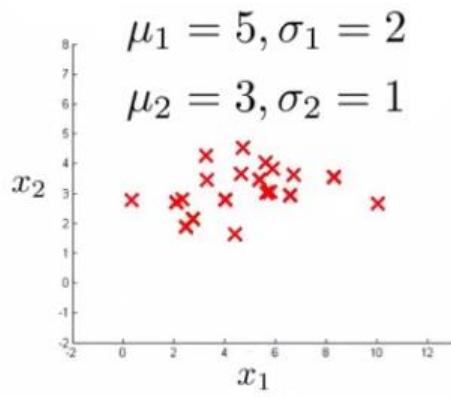
La **primera fase de elegir variables** consiste en obtener información que pueda identificar el comportamiento anómalo futuro de un cliente. Este comportamiento anómalo normalmente es inusualmente largo o pequeño.

La **segunda fase de ajuste de parámetros** determina los valores para cada uno de los ejemplos y parámetros μ_i y σ_i^2

La **tercera fase** se calcula los valores teniendo en cuenta la formula anterior (fórmula para la probabilidad gaussiana). Si este número es muy bajo, tenemos una probabilidad muy baja de que sea un registro normal.

Ejemplo de detección de anomalías

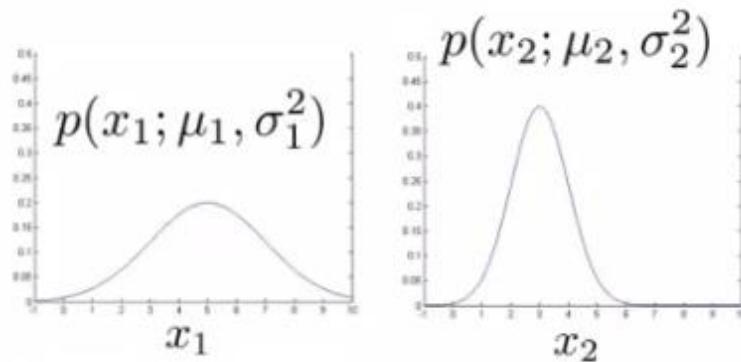
Supongamos que tenemos un modelo de dos dimensiones (X_1 y X_2). En el caso de la variable X_1 la media es 5 y la desviación estándar es 2. En el caso de la variable X_2 la media es 3 y la desviación estándar es 1. Tenemos ejemplos de datos que siguen esta distribución:



Gráfica 5. Ejemplo de distribución de datos en función de dos variables. Fuente:

<https://es.coursera.org/learn/machine-learning>

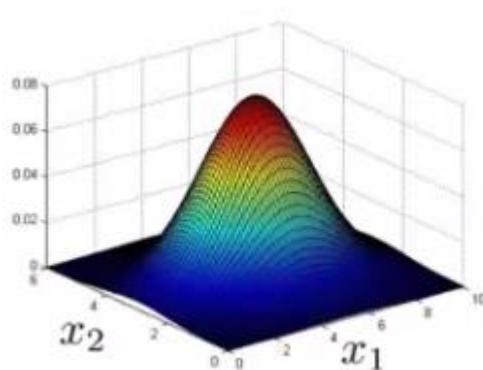
Si pintamos la distribución de la variable $X1$ y $X2$, tenemos algo como:



Gráfica 6. Distribución de cada una de las variables anteriores de forma independiente. Fuente:

<https://es.coursera.org/learn/machine-learning>

Y si pintamos el producto de ambas, obtenemos:



Gráfica 7. Distribución las variables $x1$ y $x2$ de forma conjunta.

Fuente: <https://es.coursera.org/learn/machine-learning>

En este gráfico de superficie, la altura de la superficie es la probabilidad $p(x)$. No siempre es posible hacer este tipo de gráficos puesto que habitualmente se utilizan espacios de más de dos dimensiones para crear los sistemas de detección de anomalías. Para comprobar si un valor es anómalo se establece el parámetro **épsilon** a un determinado valor. Supongamos que ahora tenemos dos puntos nuevos: X_3 y X_4 . $P(x_3) = 0.436$, lo que nos da una probabilidad de un 43 % de que el dato sea normal y por otro lado $P(x_4) = 0.0021$, lo que nos da una probabilidad de un 0,2 % de que el dato sea normal.

En este caso, el segundo ejemplo X_4 se detectaría como una anomalía.



Accede a los ejercicios de autoevaluación a través del aula virtual

11.4. Desarrollando y evaluando un sistema de detección de anomalías



Accede al vídeo «Desarrollando y evaluando un sistema de detección de anomalías» a través del aula virtual

Un aspecto importante cuando se desarrolla un sistema de detección de anomalías es **cómo se realiza la evaluación utilizando una métrica objetiva**. Esta fase de evaluación es muy relevante porque suele ser habitual y necesario el tener que tomar ciertas decisiones. Estas decisiones son más sencillas de tomar si el resultado del algoritmo es un único número que demuestra que los cambios realizados mejoran o empeoran el rendimiento del sistema de detección de anomalías.

Además, para desarrollar un sistema de detección de anomalías de forma rápida es útil disponer de una métrica de evaluación. Supongamos que disponemos de datos etiquetados sobre ejemplos anómalos y no. La evaluación puede considerar estos ejemplos para obtener una métrica de calidad del modelo.

Retomando el ejemplo de los motores. Tenemos datos etiquetados de los motores donde las muestras normales las etiquetamos con 0 y las muestras anómalas con 1. El conjunto de entrenamiento es el conjunto de ejemplos. A continuación, es necesario definir el conjunto de validación cruzada para entrenamiento, el conjunto de test y, para ambos, incluimos algunos ejemplos anómalos.

Por ejemplo, podemos tener 10 000 motores buenos y 20 motores anómalos, lo que nos proporciona un ratio de 1-500. Una posible división sería un conjunto de entrenamiento con 6000 ejemplos de tipo 0 (normales), un conjunto de validación cruzada con 2000 ejemplos de tipo 0 y 10 ejemplos de tipo 1 y un conjunto de test con 2000 ejemplos de tipo 0 y los otros 10 ejemplos de tipo 1.

El modelo $p(x)$ se entrena sobre los datos de entrenamiento y se prueba con los ejemplos en el conjunto de validación cruzada y de tipo test:

$$y = 1 \text{ if } p(x) < \text{epsilon} \text{ (anomalo)}$$
$$y = 0 \text{ if } p(x) > \text{epsilon} \text{ (normal)}$$

Se trata de **encontrar el valor de épsilon que detecte todos los ejemplos de tipo anómalo**. Es decir, puede verse como un problema de clasificación binaria, pero en este caso como las clases están muy desbalanceadas, una buena métrica a utilizar es el cálculo de *f-measure*.



Accede a los ejercicios de autoevaluación a través del aula virtual

11.5. Detección de anomalías vs. aprendizaje supervisado



Accede al vídeo «Detección de anomalías vs. Aprendizaje supervisado» a través del aula virtual

La principal diferencia del uso de algoritmos de detección de anomalías con respecto de los algoritmos de aprendizaje supervisado viene dada por **el número de ejemplos positivos frente a los negativos**. En el caso del aprendizaje supervisado es posible extraer patrones de los ejemplos positivos y el aprendizaje se realiza con ejemplos de las dos clases. Por otro lado, en el caso de la detección de anomalías al tener muy pocos ejemplos de la clase positiva no hay suficientes datos para «aprender» el patrón de la clase positiva. Por este motivo, los ejemplos positivos se reservan para los conjuntos de validación cruzada y para el test.

Además de tener pocos ejemplos de la clase positiva (anomalía) hay veces en que los tipos de anomalías son muy diferentes entre sí y, por tanto, no es posible extraer un patrón determinado.

Por otro lado, en el caso del aprendizaje supervisado tenemos un número razonable de ejemplos positivos y negativos. O bien, esperamos que todas las anomalías se comporten de una forma similar entre ellas.



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Resumen del tema» a través del aula virtual

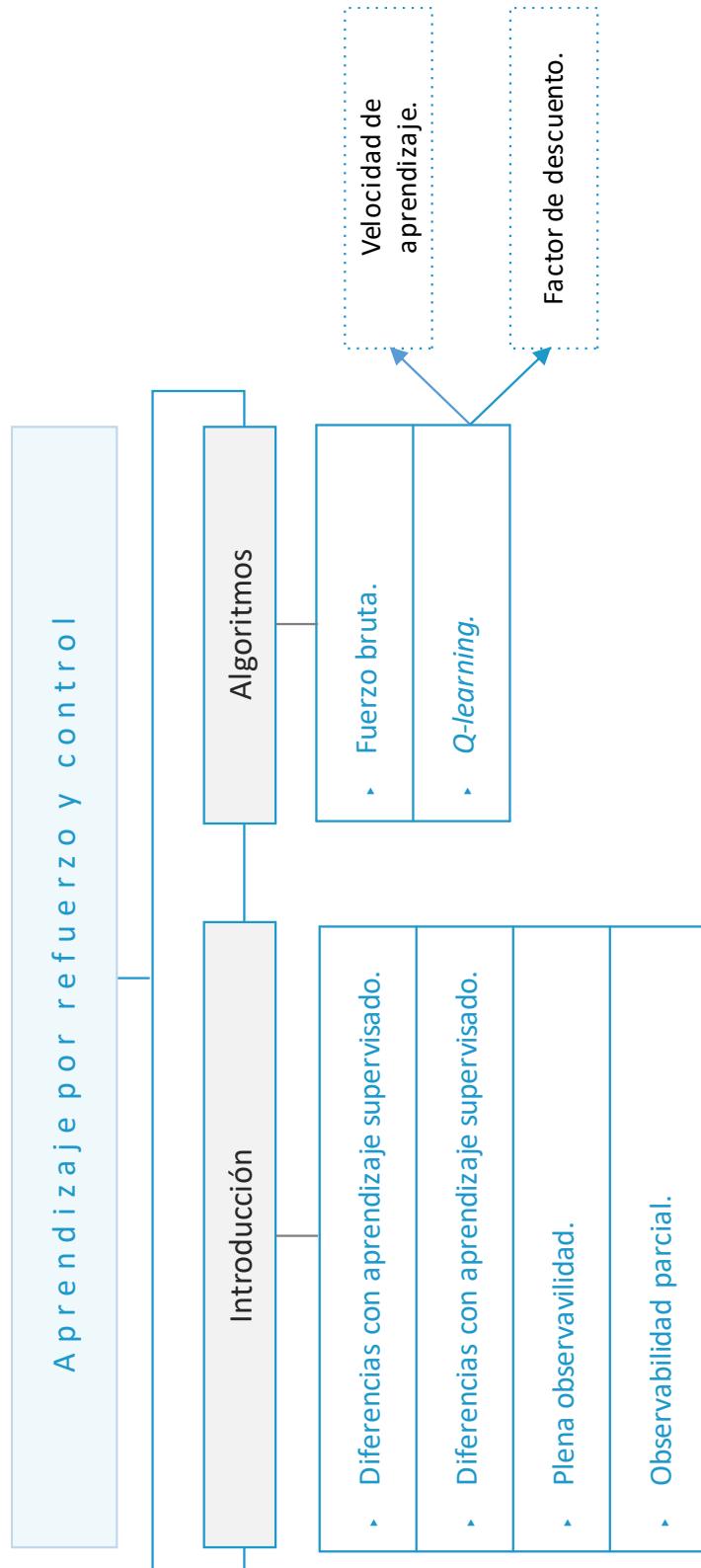
Aprendizaje Automático

Aprendizaje por refuerzo y control

Índice

Esquema	3
Ideas clave	4
12.1. Introducción y objetivos	4
12.2. Introducción al aprendizaje por refuerzo	4
12.3. Algoritmos de aprendizaje por refuerzo fuerza bruta	7
12.4. Algoritmos de aprendizaje por refuerzo Q- Learning	9
12.4. Referencias bibliográficas	12

Esquema



12.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos del aprendizaje por refuerzo» a través del aula virtual

En este tema se presentan los **algoritmos de aprendizaje por refuerzo**. En primer lugar, se realiza una introducción a los mismos y se presentan las diferencias de estos algoritmos con las técnicas de aprendizaje supervisado y aprendizaje no supervisado.

A continuación, se describe formalmente un proceso de decisión de Markov y los mecanismos que se realizan para aprender del entorno.

Posteriormente, se describen los algoritmos de aprendizaje por refuerzo basados en fuerza bruta y el algoritmo *Q-learning*.

12.2. Introducción al aprendizaje por refuerzo



Accede al vídeo «Introducción al aprendizaje por refuerzo» a través del aula virtual

Los seres vivos, y en particular los seres humanos, aprendemos las acciones a realizar en función del *feedback* o resultado que hemos observado por estas acciones previamente. Este aprendizaje se basa en las **técnicas de aprendizaje por refuerzo**, las cuales están inspiradas en la psicología conductista.

Uno de los ejemplos más populares y conocidos del aprendizaje por refuerzo es el **estudio del perro de Iván Pávlov** (1849-1936), donde se condicionaba a la mascota en función de un premio o penalización por sus acciones.

Curiosamente, la aproximación del aprendizaje supervisado existe con una mayor frecuencia en la naturaleza que los algoritmos de aprendizaje supervisado estudiados previamente. El campo del aprendizaje por refuerzo estudia los **algoritmos que son capaces de aprender de su entorno** (Taweh Beysolow II, 2019).

Diferencias con aprendizaje supervisado

La **principal diferencia** de los algoritmos de aprendizaje por refuerzo respecto de los algoritmos supervisados y no supervisados es que **reciben información del entorno acerca de lo que es apropiado**. El aprendizaje por refuerzo se estudia en diversas disciplinas como la teoría de juegos, la teoría de control o la simulación. En estos algoritmos las recompensas vienen con retraso (ganar un juego se premia al final), mientras que en el aprendizaje supervisado se optimiza una acción-efecto concreta, es decir, no se tienen en cuenta la serie de acciones futuras. En el aprendizaje por refuerzo el número de combinaciones que un agente puede llevar a cabo para conseguir el objetivo es muy grande.

Diferencias con aprendizaje no supervisado

En aprendizaje por refuerzo existe una **relación entre la entrada y salida** que no está presente en el aprendizaje no supervisado. En el aprendizaje no supervisado el objetivo es encontrar los patrones ocultos en lugar del mapeo acción-resultado. Por ejemplo, si el caso de uso es sugerir nuevas noticias a una persona: un modelo no supervisado tendrá en cuenta artículos similares a los que ha visto la persona y le serán sugeridos, mientras que un modelo de aprendizaje por refuerzo sugiere continuamente nuevos artículos para construir un «grafo de conocimiento» de los artículos que le gustan a una persona.

Para simular el aprendizaje automático en algoritmos, es necesario realizar algunas suposiciones, las cuales permiten tener un sistema más flexible capaz de una mayor generalización.

En general, lo habitual es suponer que los agentes que aprenden del entorno siguen un **proceso de decisión de Markov** (en inglés, Markov Decision Process, MDP). Básicamente, esta situación se puede definir de la siguiente forma:

- ▶ El agente puede percibir un conjunto finito (S) de estados diferentes en su entorno y dispone de un conjunto finito (A) de acciones para interactuar con el entorno.
- ▶ El tiempo avanza de forma discreta en cada instancia de tiempo t , el agente percibe un estado concreto S_t , selecciona una posible acción, a_t y la ejecuta, lo cual da lugar a un nuevo estado que se define como: $S_{t+1} = a_t(S_t)$.
- ▶ El entorno responde a cada una de las acciones del agente por medio de un castigo o recompensa, que se puede denotar por $r(S_t, a_t)$, y que por medio del uso de un número cuantos mayor es, indica que mayor será el beneficio.

Una cuestión importante de este algoritmo es que cumple la **propiedad de Markov**. Esto quiere decir que tanto la recompensa como el estado siguiente dependen únicamente del estado actual y de la acción tomada.

La finalidad de estos algoritmos es que el agente se adelante a las consecuencias de las acciones tomadas y sea capaz de identificar los estados que le llevan a conseguir una mayor eficacia y mayores recompensas.

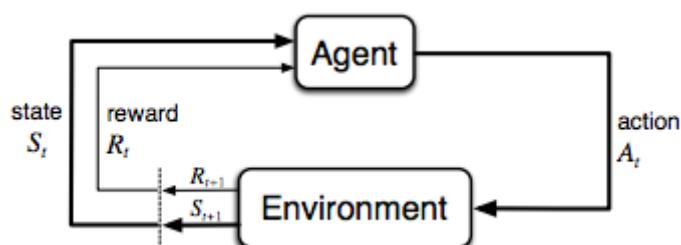


Figura 1. Esquema de funcionamiento de interacción de un agente con su entorno en un proceso de aprendizaje por refuerzo. Fuente: Sutton y Barto, 1998.

El **objetivo del aprendizaje por refuerzo** es establecer aquellas acciones que deben ser elegidas en los diferentes estados con el objetivo de maximizar la recompensa. Es decir, se busca que el agente aprenda una política que consiste en la mejor decisión a llevar a cabo en cada uno de los estados.

Hay situaciones en las cuales el agente puede observar el entorno por completo y son definidos como **plena observabilidad** y en otras se trata de **observabilidad parcial**. También situaciones con restricciones sobre las acciones que puede llevar a cabo el agente.

El aprendizaje automático es un área que ha sido aplicada con éxito a problemas de control de robots, aprendizaje de juego como el *backgammon* y las damas.

El aprendizaje por refuerzo estudia los algoritmos que son capaces de aprender de su entorno. En estas situaciones el agente que interactúa con el entorno puede tener plena observabilidad o bien observabilidad parcial.



Accede a los ejercicios de autoevaluación a través del aula virtual

12.3. Algoritmos de aprendizaje por refuerzo fuerza bruta



Accede al vídeo «Algoritmos de aprendizaje por refuerzo: fuerza bruta» a través del aula virtual

Existen varios algoritmos o formas de implementar los conceptos de aprendizaje por refuerzo. Antes de entrar en detalle en los algoritmos vamos a hacer una definición

formal de un proceso de decisión de Markov, donde tenemos los siguientes elementos:

- ▶ Conjunto de **estados**: S .
- ▶ Conjunto de **acciones**: A .
- ▶ Función de **transición**: $T: S \times A \rightarrow S$.
- ▶ Función de **recompensa**: $R: S \times A \rightarrow \mathbb{R}$

Un proceso de decisión de Markov (**MDP**) se define: $\langle S, A, T(s, a), R(s, a) \rangle$

Donde tenemos una **política**: $\pi: S \rightarrow A$ una función de **valor**:

$$V^\pi(s_t) = r_{t+1} + \gamma r_{t+2} + \gamma r_{t+3} + \dots = \sum_{i=1}^{\infty} \gamma^{i-1} r_{t+i}$$

Donde $V^\pi(s_t)$ es el valor acumulado que se consigue al seguir la política π a partir del estado s_t ; γ es un **factor de descuento** ($0 \leq \gamma \leq 1$)

La función de valor se puede definir de forma recursiva utilizando la **ecuación de Bellman**:

$$V^\pi(s) = R(s, \pi(s)) + \gamma V^\pi(T(s, \pi(s)))$$

$R(s, \pi(s))$ es la recompensa inmediata y $\gamma V^\pi(T(s, \pi(s)))$ el valor del siguiente estado.

El objetivo del agente es aprender la política óptima π^* :

$$\pi^*(s) = \operatorname{argmax} [R(s, a) + \gamma V^*(T(s, a))]$$

Donde se busca la máxima ganancia esperada a partir de s , ejecutando la acción a .

Fuerza bruta

Se trata de los algoritmos conceptualmente más sencillos de implementar. El tipo de algoritmos basados en fuerza bruta conlleva las siguientes fases:

1. Para cada acción posible, muestrear los resultados.
2. Elegir la acción con el mayor retorno esperado.

El **problema de este método** es que el número de políticas suele ser extremadamente grande, o incluso infinito. Además, la varianza de los rendimientos puede ser muy grande, lo cual hace necesario un gran número de muestras para estimar con más precisión el retorno de las acciones.



Accede a los ejercicios de autoevaluación a través del aula virtual

12.4. Algoritmos de aprendizaje por refuerzo *Q-Learning*



Accede al vídeo «Algoritmos de aprendizaje por refuerzo: *Q-Learning*» a través del aula virtual

Se trata de un algoritmo de aprendizaje por refuerzo clásico inventado hace más de 25 años, en el que **el agente aprende a asignar valores de bondad a los pares** (estado, acción). Es uno de los métodos más populares por su efectividad y por las posibilidades que ofrece para combinarlo con otras técnicas, como redes de neuronas o *deep learning*.

Si un agente está en un determinado estado y toma una acción, estamos interesados en conocer el resultado de esa acción, pero también en las recompensas futuras que se pueden obtener por pasar a otros estados. Es decir, deberemos de ser capaces de

evaluar no solamente la recompensa actual sino también la recompensa futura de las posibles acciones posteriores.

En el algoritmo *Q-learning* el valor *Q* contiene la suma de todas las posibles recompensas futuras. El problema es que este valor puede ser infinito en el caso de que no haya un estado terminal que alcanzar.

Además, es necesario establecer diferentes ponderaciones a las recompensas más recientes frente a las más lejanas. Para este último propósito se utiliza lo que se conoce como refuerzo acumulado con descuento, donde las recompensas futuras están ponderadas por un valor entre 0 y 1.

El **reto es en las primeras interacciones del agente con el entorno**, momento en el cual no se tiene la información necesaria para calcular el valor *Q*. Por tanto, se utiliza:

- ▶ Si una acción en un estado determinado es la causante de un resultado no deseado, se utiliza esta situación para no utilizar esta acción en ese estado en el futuro. De forma contraria, si una acción causa un resultado deseado, hay que aprender a aplicar esa acción en ese estado.
- ▶ Si todas las acciones que se pueden realizar desde un determinado estado dan un resultado negativo, se aprende este patrón para no tomar acciones desde otros estados que lleven a este. Por otro lado, si cualquier acción en un estado determinado proporciona un resultado positivo, es necesario aprender que se debe buscar ese estado. De esta forma se propaga la recompensa de un par (estado, acción) a los pares de los estados adyacentes.

Algoritmo

Iniciarizar $Q(s,a)$ al azar.

Repetir (para cada episodio)

- ▶ Iniciarizar s .
- ▶ Repetir (para cada paso del episodio):

- Elegir a en s según una política basada en Q .
- Ejecutar la acción a , observar r, s' .
- $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
- $s \leftarrow s'$

Hasta que s sea terminal.

Definimos $\pi(s) = \text{argmax}_a [Q(s, a)]$.

Los **episodios** incluyen un estado, la acción realizada y la recompensa recibida. El algoritmo *Q-learning* va iterando para ir llenando todos los efectos posibles de las acciones en el concepto de experiencia.

Todo lo que necesita este algoritmo para poder ser entrenado en memoria es una **tabla para almacenar las recompensas para los estados y las acciones**. La tabla contiene la mejor estimación de cada recompensa, al principio será una estimación muy mala, pero a medida que el algoritmo aprende se irá volviendo más y más precisa.

El algoritmo necesita dos parámetros que debemos ajustar en función del problema que estamos resolviendo:

- ▶ **Velocidad de aprendizaje (*learning rate*)**: es un valor entre 0 y 1 que indica cuánto se puede aprender en cada episodio. En el caso de cero indica que no se aprende nada de ese episodio y en el caso de uno establece que se borra lo que se sabía y se confía en el nuevo episodio.
- ▶ **Factor de descuento (*discount rate*)**: también es un valor entre 0 y 1 que indica cómo de importante es el largo plazo respecto del corto. Un valor de 0 significa que solo son importantes los refuerzos inmediatos, mientras que un valor de 1 implica que solo son importantes los refuerzos a largo plazo.

En ambos parámetros es interesante **moverse fuera de los extremos**, pues en este caso proporcionan poca utilidad. La velocidad de aprendizaje se puede ir ajustando en función de la incertidumbre respecto de los estados siguientes. Por otro lado, el factor de descuento establece el balance entre el refuerzo inmediato y a largo plazo.



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Resumen del tema» a través del aula virtual

12.5. Referencias bibliográficas

Sutton, R. S. y Barto, A. (1998). *Reinforcement Learning: An Introduction*. Cambridge: MIT Press.

Taweh Beysolow II, S. P. (2019). *Applied Reinforcement Learning with Python*. San Francisco, CA: Apress.

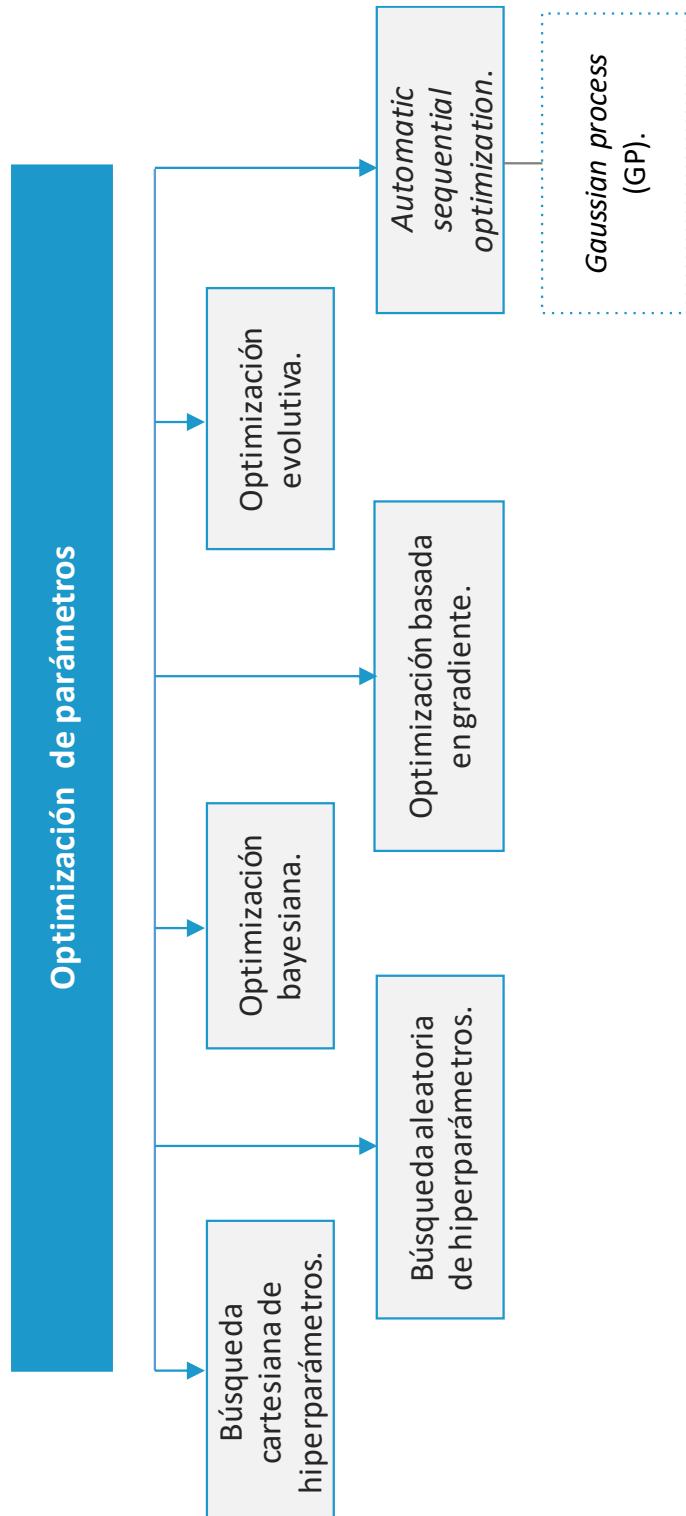
Aprendizaje Automático

Parametrización automática y optimización de algoritmos

Índice

Esquema	3
Ideas clave	4
13.1. Introducción y objetivos	4
13.2. Concepto de hiperparámetros	5
13.3. Búsqueda aleatoria de hiperparámetros	9
13.4. Mejora de la búsqueda de hiperparámetros	11
13.5. Referencias bibliográficas	13

Esquema



13.1. Introducción y objetivos



Accede al vídeo «Introducción y objetivos: parametrización-optimización de algoritmos» a través del aula virtual

E

n este tema nos vamos a centrar en el **proceso de optimización de los hiperparámetros** de un modelo de aprendizaje supervisado.

- ▶ En primer lugar, se va a realizar una definición del concepto de hiperparámetros y de su importancia.
- ▶ A continuación, se van a describir los métodos existentes para realizar la optimización de los hiperparámetros. Brevemente se va a comentar la optimización bayesiana, la optimización por gradiente y la optimización evolutiva.
- ▶ Posteriormente, con un mayor nivel de detalle se va a describir la optimización cartesiana basada en rejilla (*grid search*). Seguidamente, se detalla el proceso de optimización basada en la búsqueda aleatoria.
- ▶ Finalmente, se describen algunas técnicas o métodos para mejorar las técnicas descritas, así como los escenarios en las cuales tienen utilidad.

13.2. Concepto de hiperparámetros



Accede al vídeo «Concepto de hiperparámetros y búsqueda cartesiana» a través del aula virtual

Prácticamente todos los algoritmos de aprendizaje automático tienen en mayor o menor medida **una serie de parámetros**. Estos parámetros que son específicos de cada modelo y que sirven para modificar diversos aspectos de cómo los algoritmos «aprenden» son conocidos con el nombre de hiperparámetros o hiperparámetros (*hyperparameters* en inglés).

Una de las **principales ventajas de los hiperparámetros** en los modelos de aprendizaje automático es que permiten reducir o controlar tanto el *overfitting* como el *underfitting*. En la fase de entrenamiento de los modelos es común ir iterando sobre los datos de entrenamiento e ir comparando los resultados de la aplicación de los modelos sobre el conjunto de test. Cuando estos resultados no son satisfactorios, una forma de mejorarlo es por medio de la modificación de los parámetros del modelo, conocidos como hiperparámetros

Por ejemplo, se puede incrementar el número de iteraciones sobre los datos de entrenamiento, o bien modificar el parámetro de la tasa de aprendizaje (*learning rate*) que indica cuánto se puede ir modificando los valores en cada iteración.

Esta **optimización de hiperparámetros** es una parte importante de la construcción de modelos de aprendizaje automático y muchas veces es más un arte que una ciencia. Hay algunas personas que también la definen como una **fase de ensayo/error**.

Muchas veces, el problema radica en que existen muchos parámetros diferentes y es difícil encontrar el valor óptimo de todos ellos. El problema es precisamente elegir el

conjunto óptimo de hiperparámetros para ese algoritmo de aprendizaje dado un conjunto de datos determinado.

Una cuestión importante a tener en cuenta en la fase de optimización de hiperparámetros es **controlar el sobreajuste** u *overfitting*. El sobreajuste en los datos de entrenamiento implica que el algoritmo ha memorizado todos los detalles del conjunto de datos en lugar de obtener los patrones de los datos que también aplican a datos futuros sobre los cuales se desea realizar las predicciones.

En esta fase de optimización de parámetros y selección de modelos es **necesario evitar el sobreajuste sobre los datos**. En caso contrario, los valores de los hiperparámetros serán los óptimos para los datos de entrenamiento, pero no serán capaces de generalizar sobre nuevos datos.

El método tradicional para seleccionar los valores de los hiperparámetros ha consistido en **entrenar de forma individual diferentes modelos**, cada uno con unos valores de los parámetros y elegir el mejor modelo. A medida que el número de parámetros y los valores que se quieren probar crece, esto se vuelve poco práctico y muy difícil de gestionar.

Existen **diferentes enfoques o métodos** para realizar esta optimización como: la búsqueda cartesiana o por rejilla, la búsqueda aleatoria, la optimización bayesiana, la optimización basada en gradiente o la optimización evolutiva. A continuación, veremos la búsqueda en rejilla y la búsqueda aleatoria con más detalle. En cuanto a las otras técnicas vamos a proporcionar una breve introducción:

- ▶ Optimización bayesiana: es una metodología para optimizar funciones de caja negra ruidosas. Consiste en desarrollar un modelo estadístico de la función de los valores de los hiperparámetros a la función objetivo del conjunto de validación. El método asume que existe una función suave pero ruidosa que mapea los hiperparámetros al objetivo. Se basa en la suposición de que existe una probabilidad *a priori* de las funciones con determinados valores de parámetros y

sus salidas dando lugar a una distribución sobre las funciones. El proceso iterativamente va estableciendo parámetros para observar su resultado.

- ▶ Optimización basada en gradiente: en algunos algoritmos de aprendizaje automático es posible calcular el gradiente respecto de los hiperparámetros y por tanto optimizar los valores de los hiperparámetros utilizando la técnica de descenso del gradiente.
- ▶ Optimización evolutiva: consiste en utilizar algoritmos evolutivos para buscar el espacio de los parámetros de un algoritmo determinado. Este proceso está inspirado en el concepto biológico de la evolución.

En resumen, prácticamente todos los algoritmos de aprendizaje supervisado tienen algunos parámetros que afectan al proceso de aprendizaje del algoritmo sobre los datos, como por ejemplo el número de árboles de un *random forests*. El conjunto de las **combinaciones de todos estos valores** es lo que se conoce como hiperparámetros. En la mayoría de los algoritmos existen una serie de hiperparámetros por defecto, pero para obtener un mejor resultado es necesario optimizarlos.

Los hiperparámetros son una serie de parámetros que afectan al proceso de aprendizaje de los algoritmos supervisados sobre los datos. No se trata de parámetros que los algoritmos aprendan de los datos, sino de parámetros que es necesario establecer *a priori*. Un problema común consiste en encontrar los mejores valores de estos conjuntos de parámetros. A este proceso se le conoce con el nombre de **optimización de hiperparámetros**.

Búsqueda cartesiana de hiperparámetros

Este método de búsqueda también es conocido con el nombre de **búsqueda de rejilla cartesiana** o bien como *cartesian grid search* o *grid search* en inglés.

Se trata de un **tipo de optimización de hiperparámetros** de los modelos donde los usuarios especifican una serie de valores para cada uno de los parámetros del modelo que desean optimizar. A continuación, es necesario entrenar un modelo para cada una de las combinaciones de los valores de los hiperparámetros. Por ejemplo, en el caso de que tengamos tres hiperparámetros y se hayan especificado 5, 10 y 2 valores distintos para cada uno de ellos, será necesario crear $5 \times 10 \times 2 = 100$ modelos diferentes con cada una de las combinaciones de los parámetros.

Se trata de una búsqueda exhaustiva de un subconjunto de hiperparámetros de un algoritmo de aprendizaje automático que han sido manualmente definidos. Este algoritmo de búsqueda se debe guiar por alguna métrica de rendimiento normalmente con técnicas de validación cruzada sobre los datos de entrenamiento o conjunto independiente.

Como es posible que algunos parámetros admitan valores reales o sin límite, es **necesario establecer rangos** antes de poder aplicar este método de optimización.

Por ejemplo, en una máquina de vector de soporte con un *kernel* de base radial, es necesario especificar el valor de la constante de regularización C y el parámetro del *kernel* γ . Como ambos parámetros son continuos es necesario establecer una serie de valores para probar como, por ejemplo:

$$C \in \{10, 100, 1000\}$$

$$\gamma \in \{0.1, 0.2, 0.6\}$$

El método cartesiano entrena una SVM con cada par de valores (C, γ) y evalúa su rendimiento en un conjunto de test o bien en validación cruzada. Finalmente, el algoritmo muestra la combinación que ha aportado un mejor resultado.

El método cartesiano sufre el problema de *curse of dimensionality* o **maldición de la dimensión**. Pero, por otro lado, se trata de un método que se puede paralelizar muy bien pues la evaluación de cada uno de los conjuntos de parámetros es independiente.



Accede a los ejercicios de autoevaluación a través del aula virtual

13.3. Búsqueda aleatoria de hiperparámetros



Accede al vídeo «Búsqueda aleatoria de hiperparámetros» a través del aula virtual

Debido a que el método de búsqueda cartesiana es exhaustivo y costoso computacionalmente han surgido alternativas como la búsqueda aleatoria.

En el caso de la **búsqueda aleatoria de hiperparámetros** también es necesario establecer un rango para aquellos parámetros que se quieran optimizar. La diferencia radica en que en lugar de buscar todas las posibles combinaciones el proceso realiza un muestreo uniforme sobre todas las posibles combinaciones de parámetros.

Además, este proceso suele tener también un **criterio de parada** donde el desarrollador establece con qué umbral se puede detener el proceso. Por ejemplo, el desarrollador puede especificar un número máximo de modelos o un número máximo de tiempo permitido para la búsqueda. O bien, se puede especificar un criterio de parada basado en una métrica de rendimiento (el AUC por ejemplo) que haría que el proceso termine si se mejora el valor en un porcentaje determinado.

Dado un conjunto de tiempo finito, el hacer elecciones aleatorias dentro de unos valores generalmente proporciona mejores resultados que una búsqueda cartesiana exhaustiva.

Este proceso **muestra los parámetros un número de veces determinado** y se ha demostrado que es más efectivo en escenarios con altas dimensiones que la búsqueda cartesiana. Este motivo, se debe a que la mayoría de las veces los parámetros no afectan demasiado a la función de perdida. Por tanto, valores dispersados aleatoriamente proporcionan una mejor composición que buscar de forma exhaustiva todos los parámetros. Bergstra y Bengio en el artículo «Random Search for HyperParameter Optimization» escribieron:

«Compared with neural networks configured by a pure grid search, we find that random search over the same domain is able to find models that are as good or better within a fraction of the computation time».

La cuestión es que para la mayoría de los conjuntos de datos **solo algunos pocos parámetros son relevantes**. Este hecho hace que la búsqueda cartesiana sea una opción poco eficiente para la optimización de algoritmos.

Con la **búsqueda aleatoria** hacemos una pequeña reducción de eficiencia en aquellos algoritmos con pocos parámetros y una gran mejora de eficiencia en aquellos con un gran número de parámetros.

Una vez que se ha hecho una búsqueda aleatoria se puede hacer zoom sobre aquellas regiones del espacio de hiperparámetros que parezcan prometedoras. Esto se puede llevar a cabo haciendo una nueva búsqueda aleatoria, con el método cartesiano o bien de forma manual.

Por ejemplo, si se ha realizado una primera búsqueda aleatoria con los valores de 0,0, 0,25, 0,5, 0,75 y 1,0 y los valores del medio parecen prometedores, se puede hacer una segunda búsqueda más fina sobre 0,3, 0,4, 0,5, 0,6, 0,7.

Criterios de parada

Es habitual establecer algún criterio de parada a la hora de buscar los parámetros óptimos. Por lo general, el criterio de *early-stopping* de una métrica determinada combinada con el **tiempo máximo** de ejecución es el mejor criterio. El número de modelos que son necesarios para converger a un óptimo global puede variar bastante y el criterio de parada basado en la métrica de evaluación tiene en consideración este aspecto de forma que se detiene cuando la gráfica de error se estabiliza.

Número de modelos

El número de modelos que se requiere para converger depende de varios factores, pero principalmente en la «forma» de la función de error del espacio de hiperparámetros. Mientras que la mayoría de los algoritmos se comportan razonablemente bien en una gran región del espacio de hiperparámetros, en la mayoría de los conjuntos de datos algunas combinaciones de conjuntos de datos y algoritmos son muy sensibles: tienen unas funciones de error con muchos picos.

En un experimento de Bergstra y Bengio donde estaban optimizando redes de neuronas con un gran número de hiperparámetros para diferentes conjuntos de datos encontraron convergencia utilizando entre 2 y 64 modelos. Por lo general, los modelos basados en árboles requieren muchas menos pruebas.



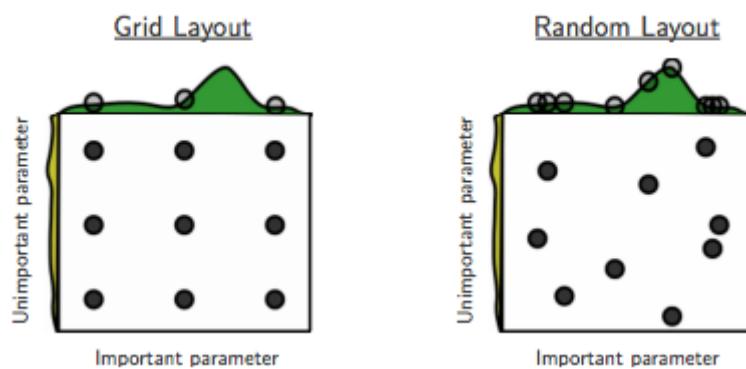
Accede a los ejercicios de autoevaluación a través del aula virtual

13.4. Mejora de la búsqueda de hiperparámetros



Accede al vídeo «Mejora de la búsqueda de hiperparámetros» a través del aula virtual

Una forma de mejorar la búsqueda aleatoria de parámetros es elegir **conjuntos de parámetros que cubran el espacio de forma más eficiente** que si se eligen de forma aleatoria. Bergstra y Bengio realizaron esta prueba y encontraron una mejora potencial pero únicamente cuando se hacían búsquedas de entre 100 y 500 modelos. Esto se debe principalmente a que el número de parámetros que suelen ser importantes para un conjunto de datos particular suele ser pequeño (1-4) y el proceso de búsqueda aleatoria lo cubre bastante bien, lo cual se representa gráficamente en la siguiente imagen.



Gráfica 1. Ejemplo de una búsqueda de hiperparámetros con el método cartesiano y el método aleatorio utilizando 9 ejecuciones. El método aleatorio es capaz de encontrar el punto óptimo mientras que el método por rejilla no lo consigue. Fuente: Bergstra y Bengio, 2012.

Sin embargo, para algunos algoritmos complejos como las *deep belief networks* la búsqueda aleatoria de hiperparámetros puede ser insuficiente. Para estos algoritmos se utiliza una técnica denominada *automatic sequential optimization* que consiste en **construir un modelo del espacio de hiperparámetros** y utilizarlo para guiar el proceso de búsqueda. La técnica más conocida de este tipo de métodos son los **modelos de procesos gaussianos (gaussian process models)**.



Accede a los ejercicios de autoevaluación a través del aula virtual



Accede al vídeo «Resumen del tema» a través del aula virtual

13.5. Referencias bibliográficas

Bergstra, J., Bengio, Y., Bardenet, R. y Kegel, B. (2011). *Algorithms for Hyper-parameter Optimization*. Recuperado de <https://papers.nips.cc/paper/4443-algorithms-for-hyper-parameter-optimization.pdf>

Bergstra, J. y Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 13, 281-305.

Hastie, T, Tibshirani, R. y Friedam, J. H. (2008). *The elements of statistical learning*. Springer.