

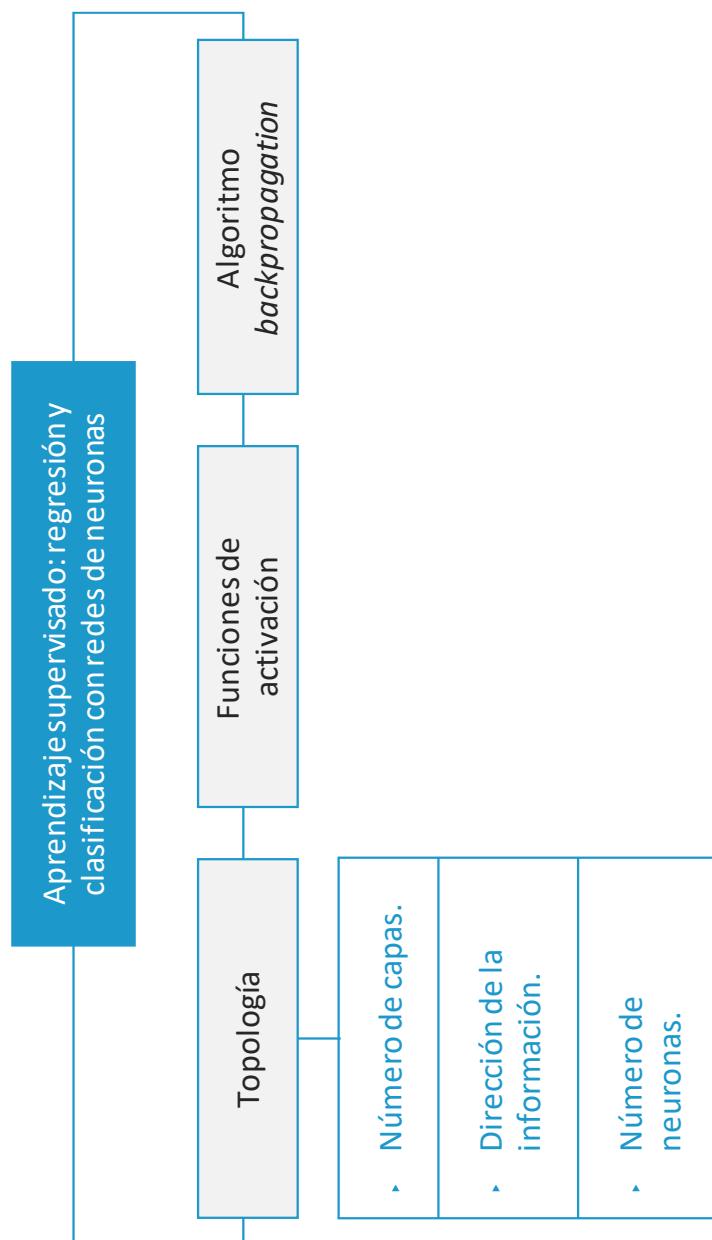
Aprendizaje Automático

Aprendizaje supervisado: regresión y clasificación con redes de neuronas

Índice

Esquema	3
Ideas clave	4
9.1. ¿Cómo estudiar este tema?	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	15
Lo + recomendado	16
+ Información	18
Test	19

Esquema



Ideas clave

9.1. ¿Cómo estudiar este tema?

Para estudiar este tema lee las **Ideas clave** disponibles a continuación.

En 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

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 estímulos. Para conseguir su propósito, las redes de neuronas se apoyan en neuronas artificiales o nodos los cuales están interconectados entre sí.

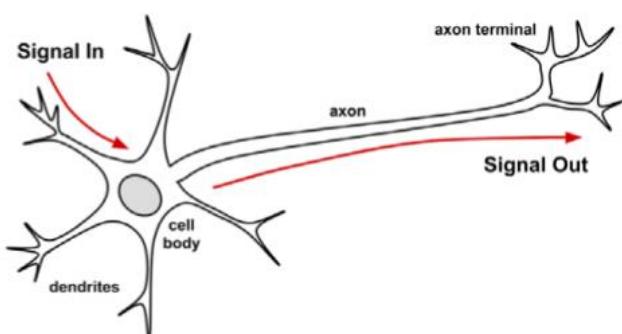


Figura 1. Ilustración de una neurona artificial. Fuente: Brett, 2013

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 relaciona 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 $f()$.

Este mismo proceso se describe en el siguiente diagrama:

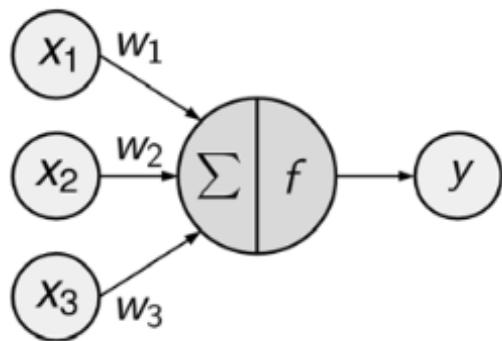


Figura 2. Diagrama descriptivo de una neurona artificial y las operaciones matemáticas que se llevan a cabo.

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.**

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

Existen 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.

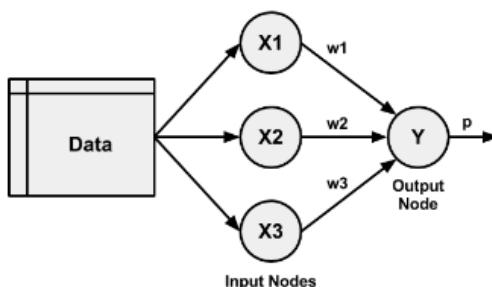


Figura 3. Ejemplo de una red de neuronas sin ninguna capa oculta, con 3 neuronas en la capa de entrada y una neurona en la capa de salida. Fuente: Brett, 2013.

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.

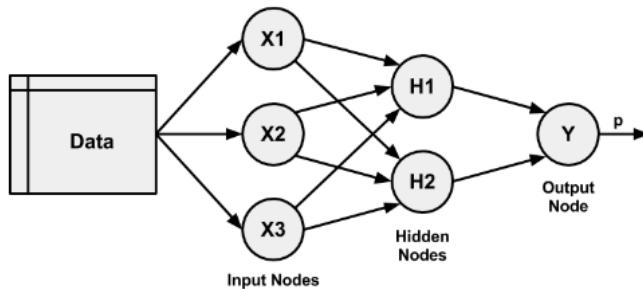


Figura 4. Ilustración de una red de neuronas multi-capa (1 capa oculta) fully connected.

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.

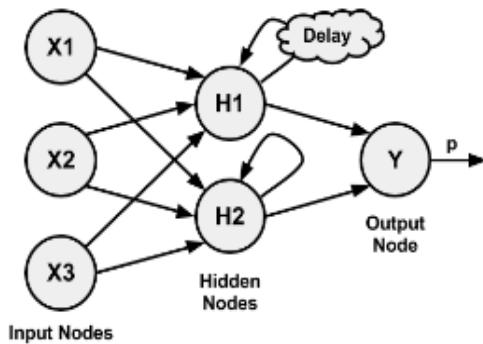


Figura 5. Ejemplo de una arquitectura de red recurrente con memoria.

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 .

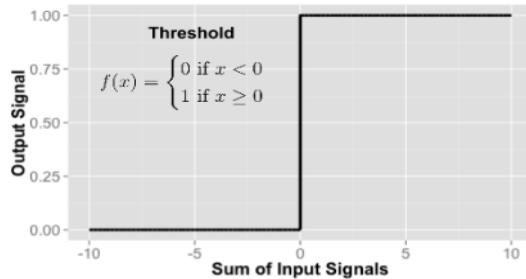


Figura 3. Gráfico ilustrativo de la función de activación unitaria.

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.

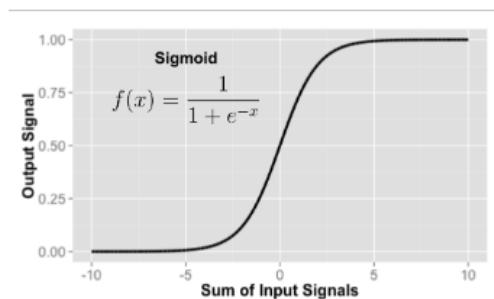


Figura 4. Gráfico ilustrativo de la función de activación *sigmoide*.

Fuente: Brett, 2013.

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

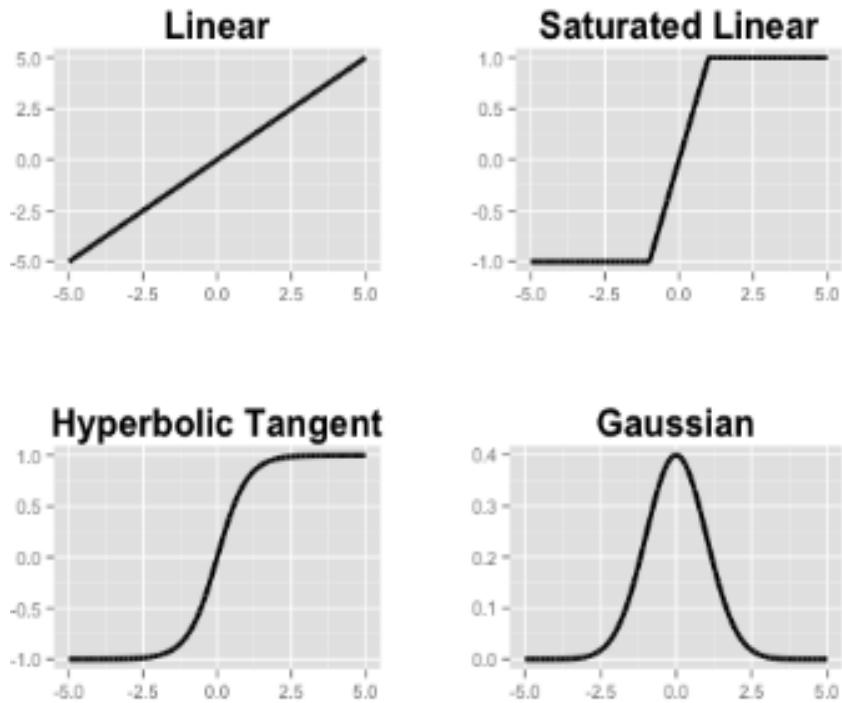


Figura 5. Gráfico ilustrativo de otra serie de funciones de activación.

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.

9.4. Algoritmo de entrenamiento:

backpropagation

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.

En el siguiente vídeo se van a repasar los conceptos de las redes de neuronas y cómo se han popularizado hoy en día siendo la base de los algoritmos de *deep learning*. También se van a discutir las ventajas e inconvenientes de su aplicación.



Ventajas e inconvenientes de las redes de neuronas.

Accede al vídeo a través del aula virtual

En este vídeo tutorial vamos a ver cómo se puede implementar y entrenar una red de neuronas.

A screenshot of a code editor window titled 'rn_neuralnet.R'. The code is an R script for implementing and training a neural network. It includes comments explaining the steps: installing packages, setting a seed, and specifying a dataset. The code also includes sections for model creation and data preparation.

Tutorial Redes de neuronas.

Accede al vídeo a través del aula

9.5. Referencias bibliográficas

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

Lo + recomendado

No dejes de leer

Redes de neuronas con la librería Scikit Learn de Python

Scikit learn. (s.f.). Neural network models.

Descripción de entrenamiento y aplicación de una red de neuronas utilizando la librería scikit-learn en Python.

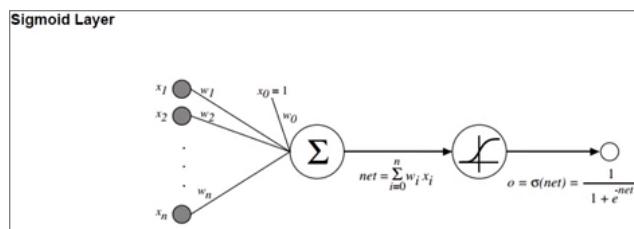
Accede al artículo a través del aula virtual o en la siguiente dirección web:

http://scikit-learn.org/stable/modules/neural_networks_supervised.html

No dejes de ver

Redes de neuronas y *backpropagation* con Scikit learn

Vídeo donde se describen las redes de neuronas y el algoritmo de *backpropagation* utilizando la librería Scikit learn y el conjunto de datos de MNIST.



Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=X8SP0875mQY>

How Deep Neural Networks Work

Vídeo con una explicación clara y detallada sobre el funcionamiento de las redes de neuronas profundas.



Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=ILsA4nyG7I0>

Webgrafía

Radacad

Página web con conceptos sobre redes de neuronas.



Accede a la página web a través del aula virtual o desde la siguiente dirección web:

<http://radacad.com/neural-network-concepts-part-1>

Bibliografía

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

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

Schalkoff, R. J. (1997). *Artificial Neural Networks*. Nueva York: McGraw-Hill.

Test

1. Las redes de neuronas artificiales:

- A. Se pueden aplicar únicamente a problemas de regresión.
- B. Se pueden aplicar únicamente a problemas de clasificación.
- C. Se pueden aplicar tanto a problemas de clasificación como de regresión.

2. Cuáles de las siguientes afirmaciones sobre el número de neuronas son ciertas:

- A. El número de neuronas de la capa de entrada viene determinado por el número de variables de entrada.
- B. El número de neuronas de salida viene determinado por el número de salidas a modelar.
- C. No hay una regla para determinar el número de cada capa.

3. Según el teorema de aproximación universal de Cybenko:

- A. La aproximación de una función matemática depende en gran medida de la semilla utilizada para generar el número aleatorio.
- B. Existe una pequeña probabilidad de no poder aproximar una función matemática con una red de neuronas.
- C. Ninguna de las anteriores es correcta.

4. Cuáles de las siguientes afirmaciones sobre la dirección de información son ciertas:

- A. Las redes *feed-forward* tienen bucles que permiten *feedback* en los datos.
- B. Las redes recurrentes siempre tienen un *delay* que permite modelar la memoria.
- C. Ninguna de las anteriores es correcta.

- 5.** En cuanto a las capas de una red de neuronas:
- A. Una red de neuronas multi-capa añade una o más capas ocultas con un número de neuronas variable en cada una.
 - B. Una red de neuronas multi-capa añade una o más capas ocultas con el mismo número de neuronas en cada una.
 - C. Están siempre definidas de antemano en todos los problemas.
- 6.** En una neurona artificial:
- A. La salida de cada neurona es obtenida después de aplicar la función de activación.
 - B. La señal de cada dendrita es ponderada con los pesos de acuerdo a su importancia.
 - C. Los valores de entrada y de salida siempre son iguales.
- 7.** La función de activación gaussiana:
- A. Da lugar a un modelo de red de base radial.
 - B. Fue aplicada por Gauss la primera vez.
 - C. Se puede utilizar únicamente en las neuronas de la primera capa.
- 8.** La función de activación lineal:
- A. Genera en la salida la misma información que la entrada.
 - B. Da lugar a un modelo de regresión lineal.
 - C. Se utiliza únicamente en las últimas capas.
- 9.** En la técnica conocida como descenso de gradiente:
- A. Los pesos se modifican siguiendo la dirección que produce una menor reducción del error.
 - B. Los pesos se modifican siguiendo la dirección que produce una mayor reducción del error.
 - C. Se utiliza la derivada de la activación de cada neurona para identificar el gradiente.

10. Cuáles de las siguientes afirmaciones sobre el algoritmo *back-propagation* son ciertas:

- A. Itera en ciclos llamados *epochs* utilizando tres fases en cada ciclo.
- B. Se utiliza para encontrar el valor óptimo de los pesos de una red.
- C. Modifica la estructura de la red para realizar el entrenamiento.



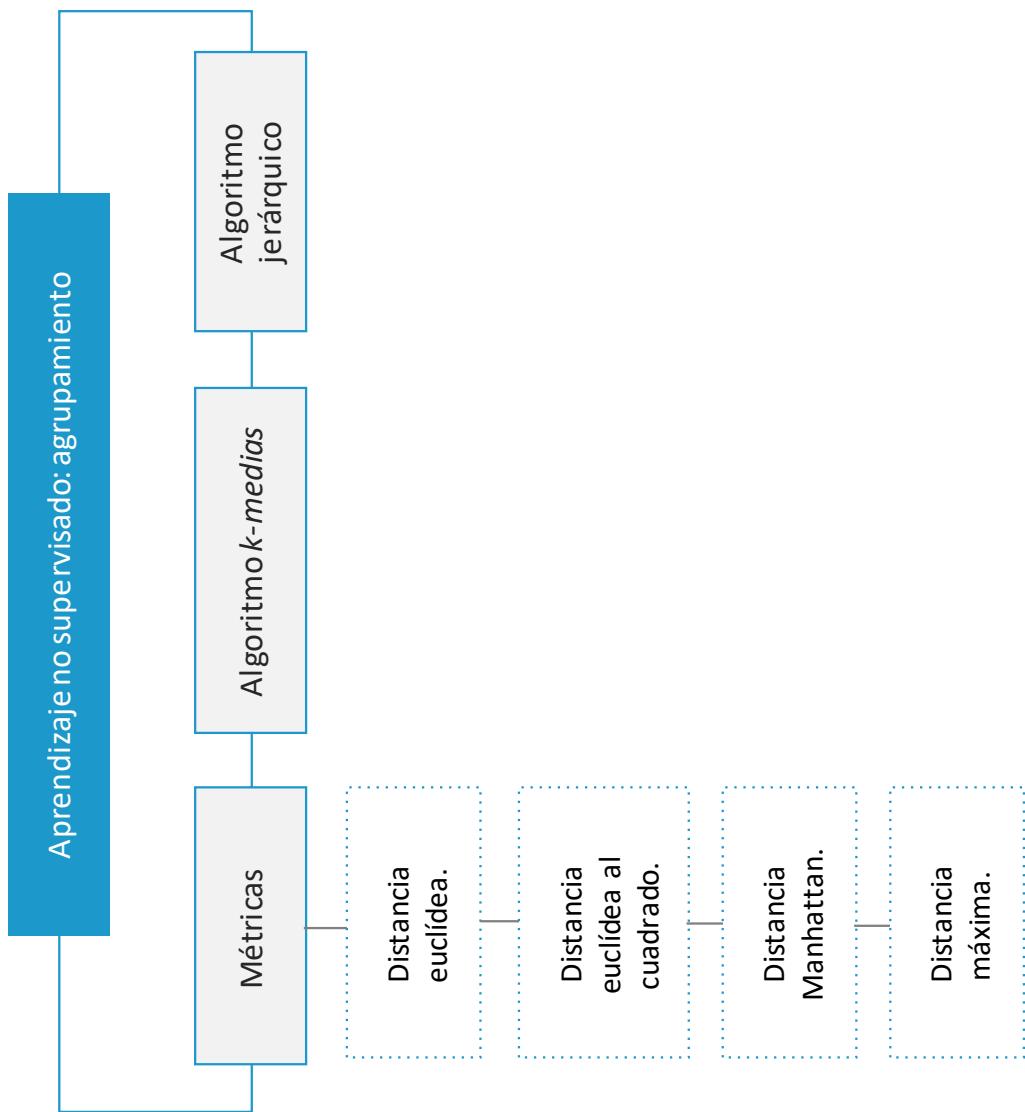
Aprendizaje Automático

Técnicas de aprendizaje no supervisado: agrupamiento

Índice

Esquema	3
Ideas clave	4
10.1. ¿Cómo estudiar este tema?	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	13
Lo + recomendado	14
+ Información	17
Test	18

Esquema



10.1. ¿Cómo estudiar este tema?

Estudia este tema a través de las **Ideas clave** disponibles a continuación.

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

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.

10.3. Algoritmo de *k-medias*

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. Despues 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.

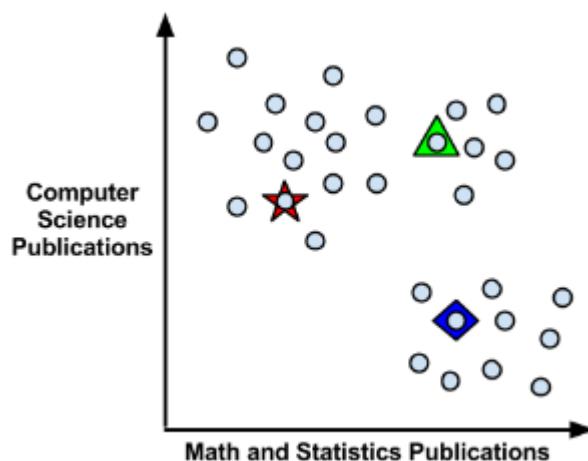
Debido a la naturaleza heurística del algoritmo **los resultados pueden ser distintos en función de la inicialización del algoritmo**. Sin embargo, si los resultados difieren mucho los unos de los otros puede indicar un problema. Por ejemplo, puede ocurrir que los datos no se puedan agrupar bien en k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

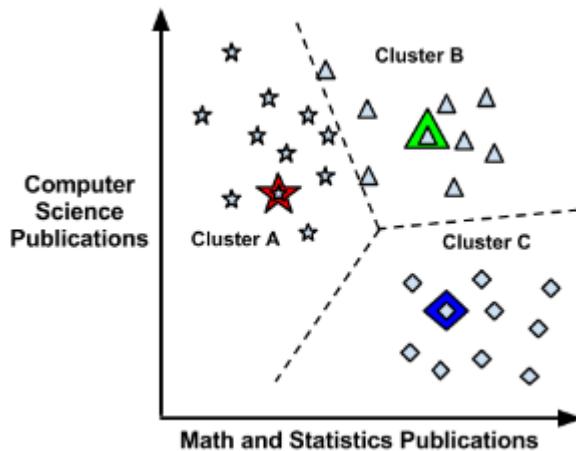


Gráfica 1. Elección inicial de tres puntos como centroides.

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.

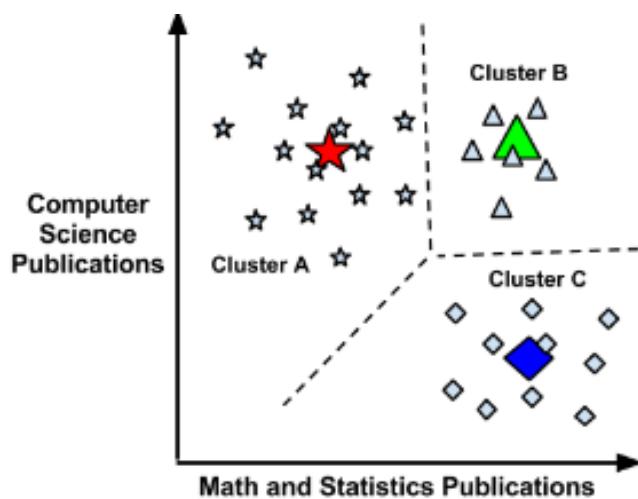


Gráfica 2. Asignación del resto de puntos a los centroides.

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.

10.4. Agrupamiento jerárquico

Los algoritmos de agrupamiento jerárquicos pueden ser de dos tipos:

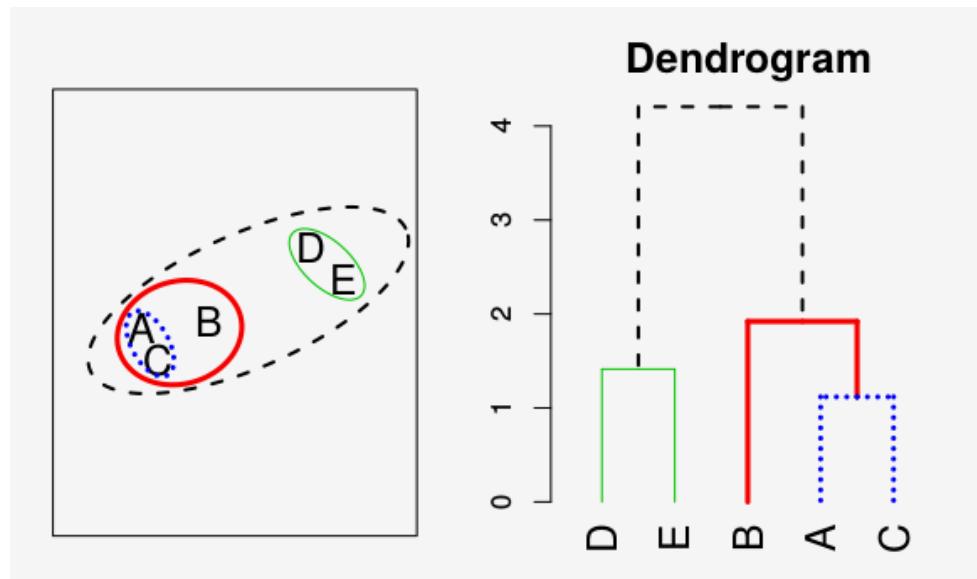
1. **Aglomerativos**: se trata de métodos *bottom-up*, donde cada observación empieza en un *cluster* y los pares de *clusters* se combinan cuando se avanza hacia arriba en la jerarquía.
2. **Divisivos**: es un método *top-down* en donde todas las observaciones empiezan en un *cluster* y se van haciendo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

En el *clustering* jerárquico no conocemos de antemano cuántos *clusters* queremos.

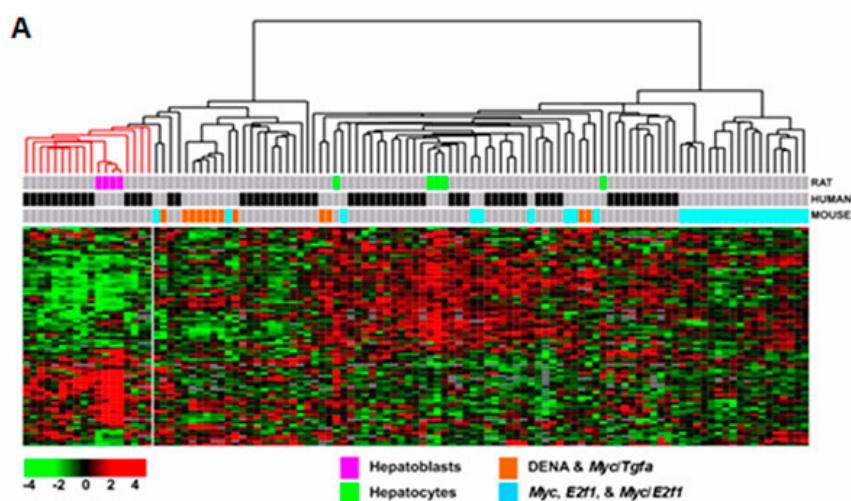
Para ello, obtenemos una **representación basada en árboles** llamada *dendrograma* que nos permite visualizar los grupos obtenidos para cada posible número de *clusters* de 1 a n .

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*.

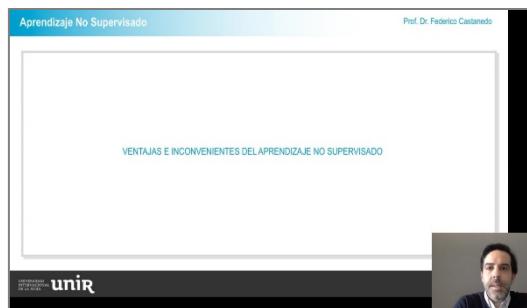


Gráfica 4. Ejemplo de un dendrograma y su correspondiente agrupamiento.



Gráfica 5. Ejemplo de un dendrograma y su mapa de calor de un *clustering* jerárquico. Recuperado de
<https://ccrod.cancer.gov/confluence/display/COEICBG/Highlighted+Article+1>

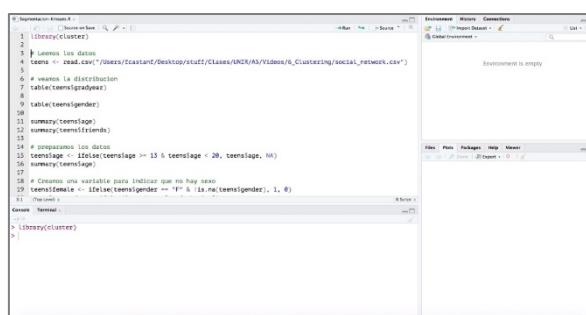
A continuación, en el siguiente vídeo se van a repasar los conceptos del aprendizaje no supervisado y se van a discutir las ventajas e inconvenientes de su aplicación.



Ventajas e inconvenientes del aprendizaje no supervisado.

Accede al vídeo a través del aula virtual

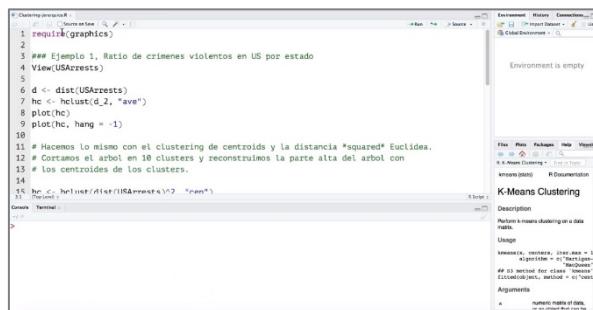
En el vídeo tutorial que te presentamos se utiliza *cluster k-means* para hacer una segmentación de usuarios en una red social.



Tutorial Clustering I.

Accede al vídeo a través del aula virtual

En la segunda parte del tutorial se puede ver cómo se ejecuta el *clustering* jerárquico.



```
R> Clustering jerárquico.R
1 ##> Source file: C:/Users/...
2
3 #### Ejemplo 1. Ratio de crímenes violentos en US por estado
4 View(USArrests)
5
6 d <- dist(USArrests)
7 hc <- hclust(d, "ave")
8 plot(hc)
9 plot(hc, hang = -1)
10
11 # Hacemos lo mismo con el clustering de centroides y la distancia "squared" Euclidea.
12 # Cortamos el árbol en 10 clusters y reconstruimos la parte alta del árbol con
13 # los centroides de los clusters.
14
15 hc <- hc[as.numeric(dist(USArrests))>= "prew"]
16
17
18
```

Tutorial *Clustering* II.

Accede al vídeo a través del aula virtual

10.5. Referencias bibliográficas

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

Lo + recomendado

No dejes de leer

K Means Clustering in R

Kodali, T. (28 de diciembre de 2015). K Means Clustering in R [Mensaje en un blog].

Blog post sobre *k-means* en *R*.

Accede a la entrada a través del aula virtual o desde la siguiente dirección web:

<https://www.r-bloggers.com/k-means-clustering-in-r/>

Hierarchical Clustering

Kodali, T. (22 de enero de 2016). Hierarchical Clustering. [Mensaje en un blog].

Blog post sobre cómo realizar *clustering jerárquico* en *R*.

Accede a la entrada a través del aula virtual o desde la siguiente dirección web:

<https://www.r-bloggers.com/hierarchical-clustering-in-r-2/>

In Depth: k-Means Clustering

VanderPlas, J. (s. f.). In Depth: k-Means Clustering.

Ejemplo sobre la aplicación de *k-means* utilizando Python.

Accede al extracto a través del aula virtual o desde la siguiente dirección web:

<https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html>

SciPy Hierarchical Clustering and Dendrogram Tutorial

Joern. (s. f.). SciPy Hierarchical Clustering and Dendrogram Tutorial [Mensaje en un blog].

Tutorial de *clustering* jerárquico en Python.

Accede a la entrada a través del aula virtual o desde la siguiente dirección web:

<https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/>

Clustering with K-Means in Python

The Data Science Lab. (12 de diciembre de 2013). Clustering with K-Means in Python [Mensaje en un blog].

Ejemplo ilustrativo del algoritmo *k-means*.

Accede a la entrada a través del aula virtual o desde la siguiente dirección web:
<https://datascienceblog.wordpress.com/2013/12/12/clustering-with-k-means-in-python/>

No dejes de ver

Data Science and Machine Learning with scikit-learn

Vídeo sobre aprendizaje no supervisado con Python y la librería Scikit-Learn

Data Science and (Unsupervised) Machine Learning with scikit-learn

By **Nicolas Kruchten at Datacratic**

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=2IpS6gUwiJQ>

A fondo

Data clustering: 50 years beyond K-means

Jain, A. K. (junio de 2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 8, 651-666.

Organizar los datos en grupos sensibles es uno de los modos más fundamentales de comprensión y aprendizaje.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:

<https://www.sciencedirect.com/science/article/pii/S0167865509002323>

Bibliografía

Hastie, T., Tibshirani, R. y Friedman, J. H. (2017). *The Elements os Statistical Learning*. Nueva York: Springer.

Dark, S. (2019). *Aprendizaje automático: la guía definitiva para principiantes para comprender el aprendizaje automático*. Manual autopublicado.

Test

1. En el aprendizaje no supervisado:

- A. Puede existir o no información de la variable objetivo.
- B. El valor de la variable objetivo se desconoce y se desea agrupar los datos.
- C. Se conoce la variable objetivo y se utiliza en estos algoritmos.

2. Los algoritmos de aprendizaje no supervisado:

- A. Buscan patrones en los datos con el objetivo de realizar agrupaciones.
- B. Buscan patrones en los datos con el objetivo de predecir una variable.
- C. Ninguna de las anteriores es correcta.

3. La división entre grupos similares, se lleva a cabo:

- A. Indicando las características de cada grupo.
- B. Utilizando variables de los diferentes grupos, pero sin indicarle las características de los grupos.
- C. Agrupando los grupos en función de una variable objetivo.

4. Los algoritmos de agrupación:

- A. Tienen definidos el número de grupos de antemano.
- B. Comienzan en un punto y van iterando.
- C. Son la base para el aprendizaje supervisado.

5. Los algoritmos jerárquicos:

- A. En cada iteración solo un objeto cambia de grupo.
- B. Los grupos están anidados en pasos anteriores.
- C. Si un objeto ha sido asignado a un grupo, ya no se vuelve a cambiar.

6. El algoritmo *k-medias*:

- A. Se trata de un algoritmo iterativo.
- B. Utiliza K iteraciones.
- C. Se compone de 2 fases, de asignación y re-asignación.

7. El valor de K :

- A. Siempre se elige de forma óptima.
- B. Para seleccionarlo se requiere de un cierto balance.
- C. Se puede utilizar el criterio de negocio.

8. En el *cluster* jerárquico:

- A. No se conoce de antemano cuantos grupos son necesarios.
- B. Se conoce de antemano cuantos grupos se desean.
- C. Se utiliza una representación en árbol llamada dendrograma.

9. Los algoritmos jerárquicos aglomerativos:

- A. Son un método *top-down*.
- B. Son un método *bottom-up*.
- C. Son un método *bottom-up* y *top-down*.

10. Los algoritmos jerárquicos divisivos:

- A. Son un método *top-down*.
- B. Son un método *bottom-up*.
- C. Son un método *bottom-up* y *top-down*.

✓
10

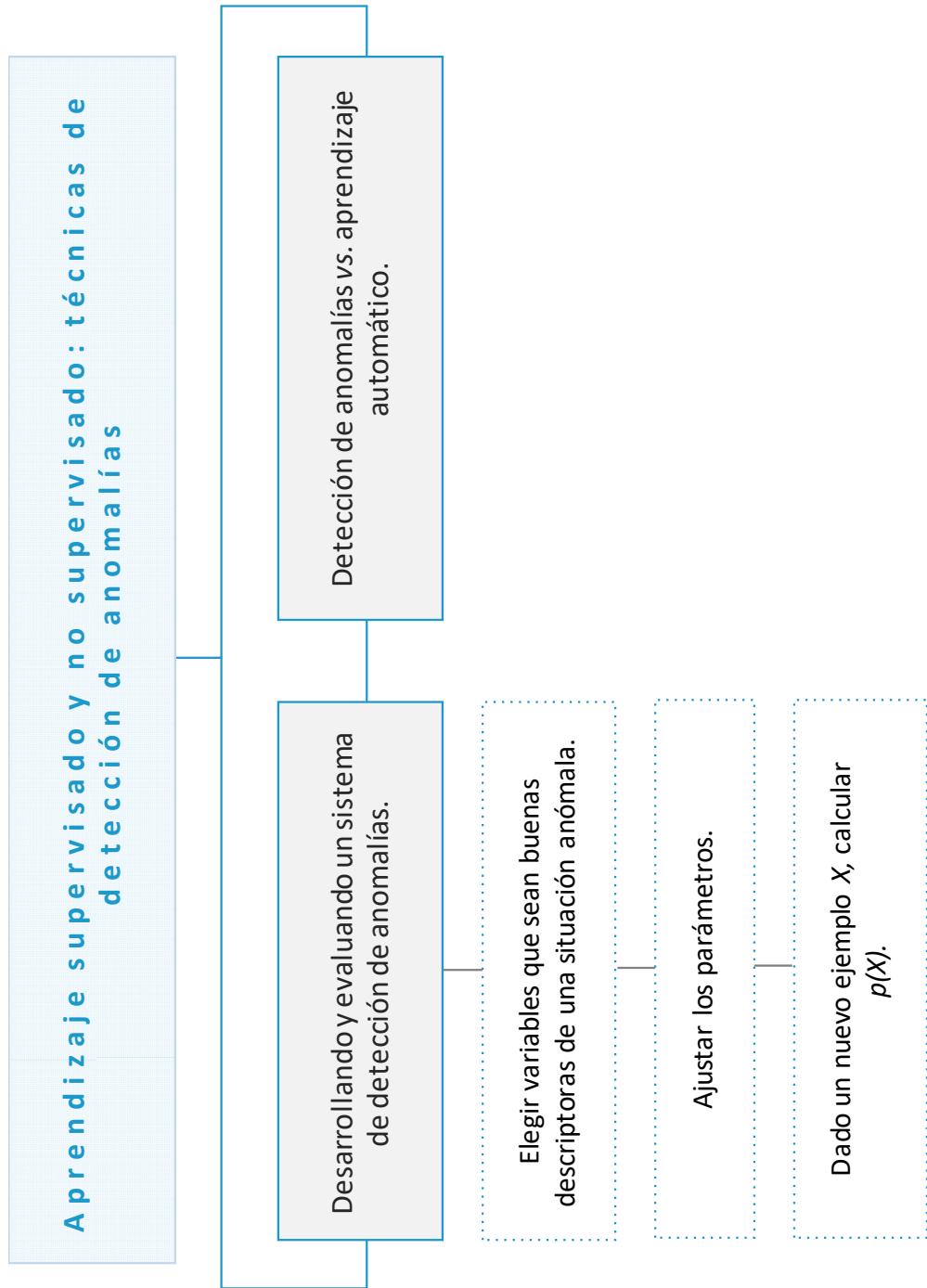
Aprendizaje Automático

Técnicas de detección de anomalías

Índice

Esquema	3
Ideas clave	4
1.1. ¿Cómo estudiar este tema?	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 vs. aprendizaje supervisado	13
Lo + recomendado	15
+ Información	17
Test	18

Esquema



Ideas clave

1.1. ¿Cómo estudiar este tema?

Estudia este tema a través de las **Ideas clave** disponibles a continuación.

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.

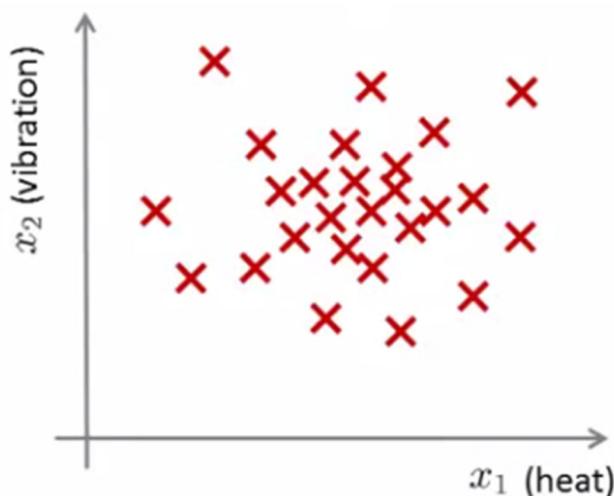
- ▶ En primer lugar, se introducen los métodos de detección de anomalías y su principal aplicación.
- ▶ A continuación, se describe cómo se pueden utilizar los métodos de aprendizaje supervisado para utilizarse en el ámbito de la detección de anomalías.
- ▶ Posteriormente, se cubren las pautas necesarias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

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.

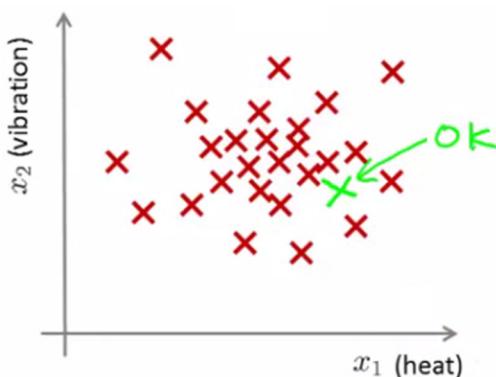
Pero, ¿qué es la detección de anomalías? Imagine que trabaja en una empresa que fabrica motores de aviones. A medida que los motores salen de la cadena de montaje,

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.



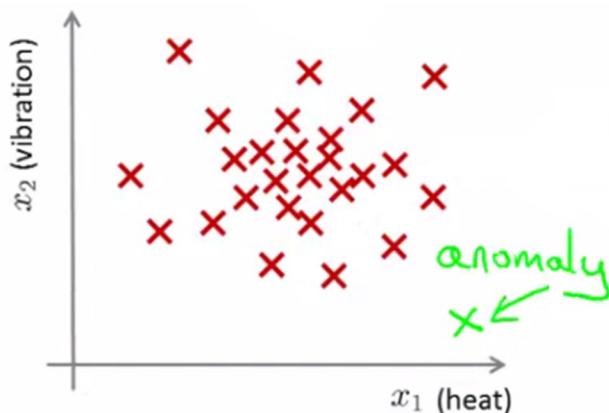
Gráfica 1. Distribución de valores de vibración y calor generado por motores. Fuente:
<https://es.coursera.org/learn/machine-learning>

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.

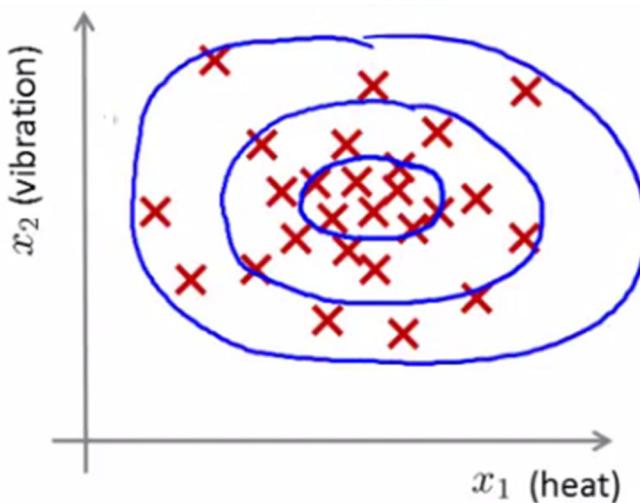
¿Cómo se realiza esta comprobación? En primer lugar, se utiliza el conjunto de entrenamiento para entrenar un modelo.

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 queremos 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.

11.3. Aplicación del aprendizaje automático a la detección de anomalías

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:**

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

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 $p(x) < \epsilon$

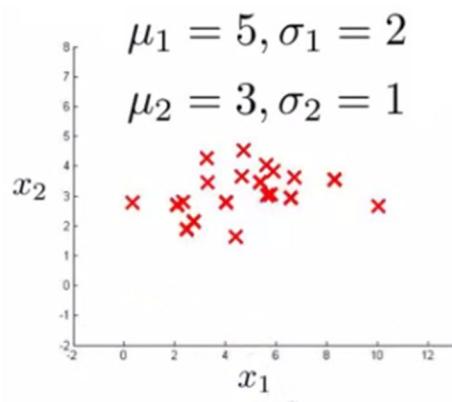
La **primera fase de elegir variables** consiste en obtener información que pueda identificar el comportamiento anómalo futuro de un cliente. Este comportamiento anómalo normalmente es inusualmente largo o pequeño.

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.

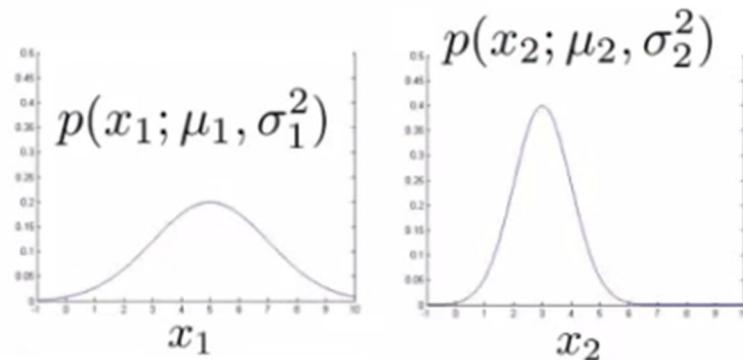
Ejemplo de detección de anomalías

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:



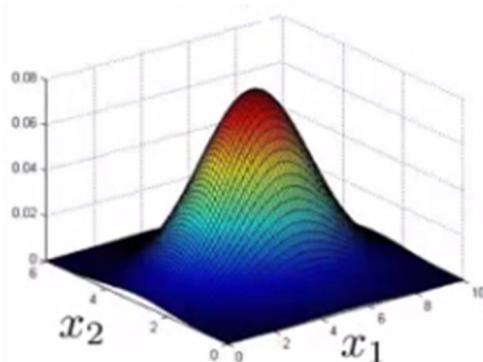
Gráfica 5. Ejemplo de distribución de datos en función de dos variables. Fuente:
<https://es.coursera.org/learn/machine-learning>

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: $X3$ y $X4$. $P(x3) = 0.436$, lo que nos da una probabilidad de un 43 % de que el dato sea normal y por otro lado $P(x4) = 0.0021$, 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.

11.4. Desarrollando y evaluando un sistema de detección de anomalías

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*.

11.5. Detección de anomalías vs. aprendizaje

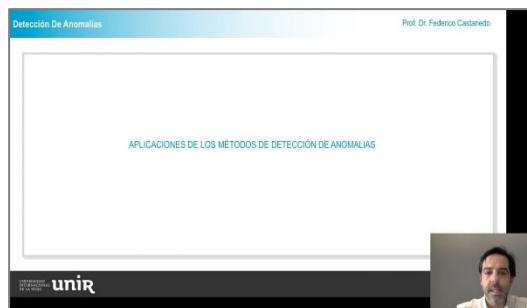
supervisado

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.

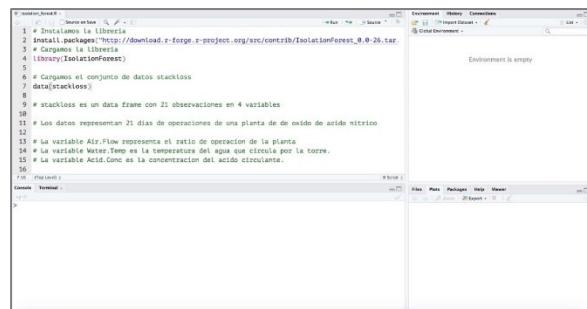
En el siguiente vídeo se van a discutir las principales aplicaciones de los métodos de detección de anomalías.



Aplicaciones de los métodos de detección de anomalías.

Accede al vídeo a través del aula virtual

En este vídeo tutorial se puede ver un ejemplo de detección de anomalías, para ello se utiliza una técnica que se denomina *Isolation Forest*.



Tutorial Detección de anomalías.

Accede al vídeo a través del aula virtual

Lo + recomendado

No dejes de leer

Detección de anomalías en Python

Scikit learn. (s. f.) Novelty and Outlier Detection.

En esta página se describe los métodos implementados en la librería scikit learn de Python para realizar la detección de anomalías.

Accede a la página a través del aula virtual o desde la siguiente dirección web:

http://scikit-learn.org/stable/modules/outlier_detection.html

No dejes de ver

Data-driven Anomaly Detection

Charla de Nikunj Oza en Google, dentro de la serie de charlas técnicas. El vídeo cubre el trabajo del equipo de *data science* de la Nasa en detección de anomalías aplicadas al control de tráfico aéreo.

Nikunj Oza
Data-Driven Methods for Detection
of Anomalous Trajectories

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

https://www.youtube.com/watch?v=5mBiac_dhbs

Anomaly Detection for Payment Processing at Netflix

Charla en Association for Computing Machinery (ACM) sobre la detección de anomalías para el procesamiento de pagos que se ha realizado en la empresa Netflix donde se discuten tanto los retos científicos de trabajar con datos de 50 millones de clientes como los retos técnicos de montar una solución con una arquitectura escalable.

Data Science & Engineering @ Netflix

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=xjHlu9OViVc>

A fondo

Anomaly Detection: A Survey

Chandola, V. (2009). Anomaly Detection: A survey.

Artículo académico con bastante extensión y profundidad sobre los métodos de detección de anomalías existentes.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:

<http://cucis.ece.northwestern.edu/projects/DMS/publications/AnomalyDetection.pdf>

df

1. Los problemas de detección de anomalías:

- A. Utilizan únicamente técnicas de aprendizaje supervisado.
- B. Utilizan únicamente técnicas de aprendizaje no supervisado.
- C. Ninguna de las anteriores es correcta.

2. En los problemas de detección de anomalías:

- A. Partimos de un conjunto de ejemplos distribuido de forma equitativa entre casos normales y anómalos.
- B. Partimos de un conjunto de ejemplos que son en su mayoría normales.
- C. Partimos de un conjunto de ejemplos distribuido principalmente en observaciones anómalas.

3. Para detectar una anomalía se utiliza:

- A. Un umbral de probabilidad definido en función del nivel de certeza que queremos tener.
- B. Un conjunto de test independiente.
- C. Ninguna de las anteriores es correcta.

4. En los problemas de detección de anomalías:

- A. Se multiplica la probabilidad de cada una de las variables y asumimos que cada una de ellas se distribuye siguiendo una distribución de Poisson.
- B. Se multiplica la probabilidad de cada una de las variables y asumimos que cada una de ellas se distribuye siguiendo una distribución gaussiana.
- C. Ninguna de las anteriores es correcta.

5. ¿Qué métrica es apropiada en los sistemas de detección de anomalías?

- A. El área bajo la curva.
- B. La métrica *f-measure*.
- C. Todas las anteriores son correctas.

6. La principal diferencia de la detección de anomalías respecto al aprendizaje supervisado viene dada por:

- A. El número de ejemplos de la clase positiva respecto de la clase negativa.
- B. El desbalanceo de clases.
- C. El tipo de algoritmos que se utilizan.

7. La principal diferencia de la detección de anomalías respecto al aprendizaje no supervisado viene dada por:

- A. El tipo de algoritmos que se utilizan.
- B. El objetivo de cada uno de los algoritmos.
- C. Ninguna de las anteriores es correcta.

8. En el caso de la detección de anomalías:

- A. Hay veces en que las anomalías son muy diferentes entre sí y no se puede encontrar un patrón.
- B. Las anomalías suelen ser siempre iguales.
- C. Ninguna de las anteriores es correcta.

9. En el aprendizaje supervisado:

- A. Tenemos un número razonable de clases positivas y negativas.
- B. Esperamos que todas las anomalías se comporten de forma similar.
- C. Ninguna de las anteriores es correcta.

10. En los métodos de detección de anomalías:

- A. Automáticamente se detecta la anomalía sin necesidad de fijar ningún parámetro.
- B. Es necesario definir un parámetro épsilon de sensibilidad.
- C. El parámetro épsilon es opcional.

70 ✓

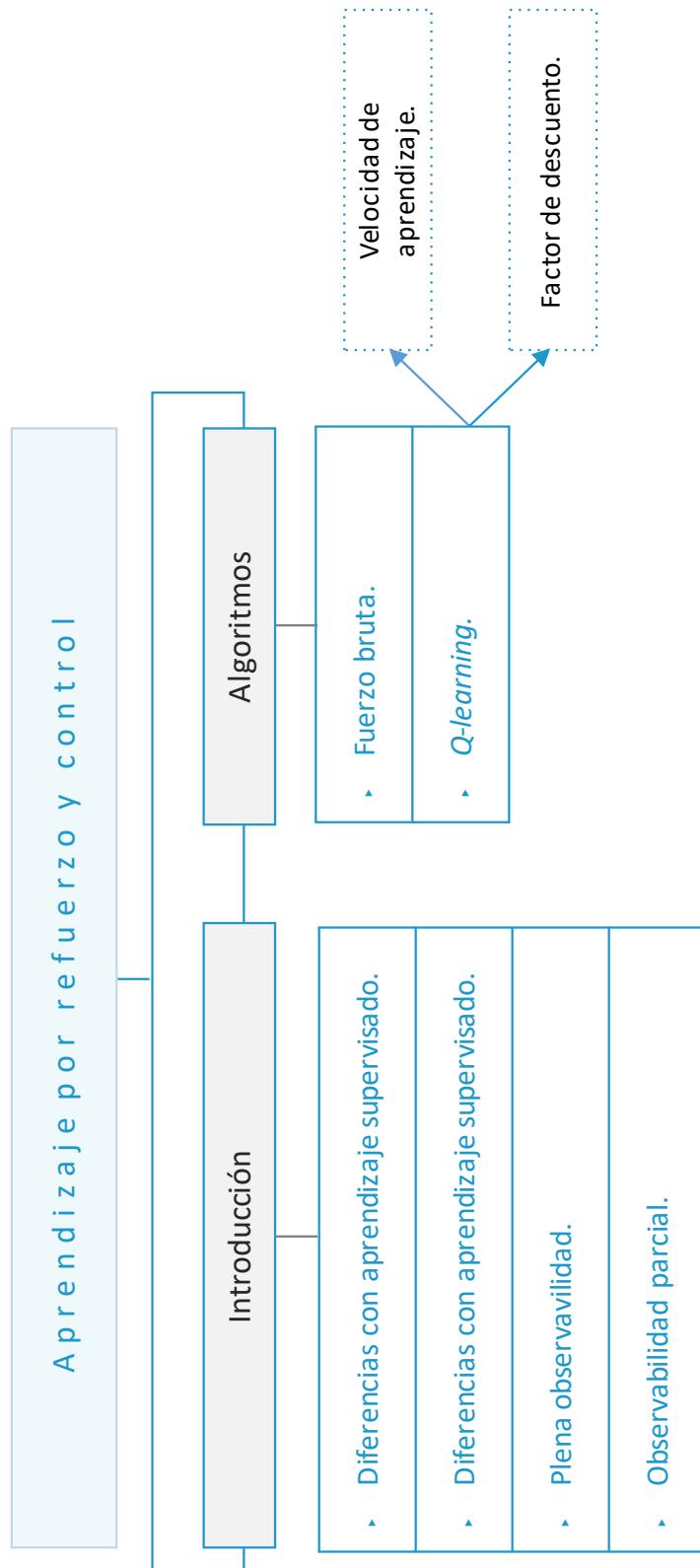
Aprendizaje Automático

Aprendizaje por refuerzo y control

Índice

Esquema	3
Ideas clave	4
12.1. ¿Cómo estudiar este tema?	4
12.2. Introducción al aprendizaje por refuerzo	4
12.3. Algoritmos de aprendizaje por refuerzo	8
12.4. Referencias bibliográficas	12
Lo + recomendado	13
+ Información	16
Test	17

Esquema



12.1. ¿Cómo estudiar este tema?

Estudia este tema a través de las **Ideas clave** disponibles a continuación.

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

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 que cuanto 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.

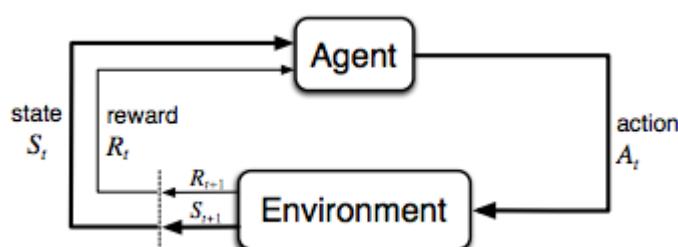


Figura 1. Esquema de funcionamiento de interacción de un agente con su entorno en un proceso de aprendizaje por refuerzo. Fuente: Sutton y Barto, 1998.

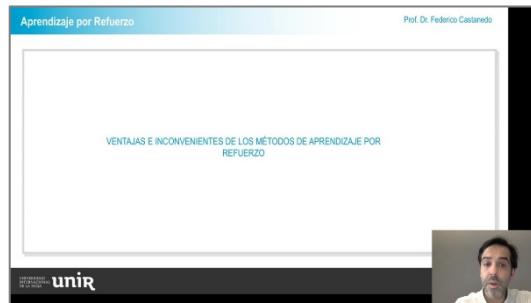
El objetivo del aprendizaje por refuerzo es establecer aquellas acciones que deben ser elegidas en los diferentes estados con el objetivo de maximizar la recompensa. Es decir, se busca que el agente aprenda una política que consiste en la mejor decisión a llevar a cabo en cada uno de los estados.

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.

En el siguiente vídeo se van a discutir las ventajas e inconvenientes de los algoritmos de aprendizaje por refuerzo, así como las situaciones en las cuales se deberían aplicar.



Ventajas e inconvenientes de los métodos de aprendizaje por refuerzo.

Accede al vídeo a través del aula virtual

12.3. Algoritmos de aprendizaje por refuerzo

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.

Q-Learning

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

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

Repetir (para cada episodio)

- ▶ Inicializar 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.

12.4. 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.

Lo + recomendado

No dejes de leer

Introducción al aprendizaje con refuerzo y OpenAI

Francis, J. (13 de julio de 2017). Introduction to reinforcement learning and OpenAI Gym.

Blog post introductorio del aprendizaje por refuerzo y ejemplos de uso de la librería Gym de OpenAI para desarrollar y probar algoritmos de aprendizaje por refuerzo.

Accede al post a través del aula virtual o desde la siguiente dirección web:

<https://www.oreilly.com/learning/introduction-to-reinforcement-learning-and-openai-gym>

Taller de aprendizaje por refuerzo

University of California, Berkeley. (s. f.). Reinforcement Learning.

Ejemplo de un modelo de aprendizaje por refuerzo para implementar la lógica del juego de pacman utilizando Python y estrategias de *Q-learning*.

Accede a la página a través del aula virtual o desde la siguiente dirección web:

<https://inst.eecs.berkeley.edu/~cs188/sp12/projects/reinforcement/reinforcement.html>

No dejes de ver

Aprendizaje por refuerzo

Vídeo demostrativo del uso del aprendizaje por refuerzo (*Q-learning*) con Python para encontrar el camino más corto entre dos puntos.

Reinforcement Learning - A Step Closer to AI with Assisted Q-Learning

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

https://www.youtube.com/watch?v=nSxaG_Kjw_w

Tutorial de *Reinforcement Learning*

Vídeo tutorial de Microsoft Research sobre *deep learning*. Proporciona una descripción de los procesos de decisión de Markov (MDP) incluyendo los métodos de programación dinámica de Monte Carlo. Se centra en la combinación de estos métodos con aproximaciones paramétricas para buscar buenas soluciones a los problemas que de otra forma serían muy largos de ser llevados a cabo.

What is Reinforcement Learning?

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://youtu.be/ggqnxyjaKe4>

Introduction to Reinforcement Learning.

Charla de David Silver sobre aprendizaje por refuerzo con bastantes ejemplos intuitivos y la aplicación en los juegos.



Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=2pWv7GOvufo>

A fondo

Practical Reinforcement Learning

Farrukh, S. M. (2017). *Practical Reinforcement Learning*. Packt.

Este libro te va a ayudar a dominar diferentes técnicas de aprendizaje de refuerzo y su implementación práctica usando OpenAI Gym, Python y Java.

Accede al libro a través del aula virtual o desde la siguiente dirección web:

https://www.amazon.es/Practical-Reinforcement-Learning-Farrukh-Akhtar/dp/1787128725/ref=sr_1_3?ie=UTF8&qid=1519947114&sr=8-3

Bibliografía

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

1. El aprendizaje por refuerzo:

- A. Es un tipo de aprendizaje supervisado.
- B. Es un tipo de aprendizaje no supervisado.
- C. Ninguna de las anteriores es correcta.

2. El aprendizaje por refuerzo:

- A. Va aprendiendo del *feedback* obtenido por cada acción.
- B. Se utiliza en las situaciones en las que un agente puede observar el entorno.
- C. Comprende los algoritmos que son capaces de aprender del entorno.

3. En un proceso de decisión de Markov:

- A. Solo se tienen en cuenta los estados posteriores.
- B. Solo se tienen en cuenta el estado previo.
- C. Se tienen en cuenta el estado previo y los siguientes.

4. En el algoritmo *Q-learning*:

- A. Si una acción en un estado es la causante de un resultado no deseado, esta acción no se usará en el futuro.
- B. Si una acción en un estado es la causante de un resultado deseado, se aplicará esa acción en ese estado.
- C. La mejora del algoritmo *Q-learning* es porque no es necesario utilizar el estado.

5. Los parámetros de *learning rate* y *discount rate* del algoritmo *Q-learning*:

- A. Es mejor que estén cercanos a 1.
- B. Es mejor que estén cercanos a 0.
- C. Idealmente deberían estar alejados de los extremos.

6. La ecuación de Bellman:

- A. Actualmente está en desuso.
- B. Se utiliza como punto inicial del aprendizaje.
- C. Permite definir el valor de forma recursiva.

7. El algoritmo de aprendizaje por refuerzo de fuerza bruta:

- A. Es una forma óptima de solucionar el problema.
 - B. Explora todas las posibles combinaciones.
 - C. Es un método costoso.
8. El algoritmo *Q-learning*:
- A. Únicamente tiene en cuenta las recompensas a largo plazo.
 - B. El valor *Q* contiene la suma de todas las posibles recompensas futuras.
 - C. Tiene en cuenta tanto las recompensas a largo plazo como a corto.

9. La velocidad de aprendizaje del algoritmo *Q-learning*:

- A. Es un valor entre 0 y 1 que indica cuánto se puede aprender en cada episodio.
- B. En el caso de 0 no se aprende nada.
- C. En el caso de 1 se borra lo anterior y se aprende de nuevo.

10. El factor de descuento del algoritmo *Q-learning*:

- A. Es un valor entre 0 y 100 que indica la importancia del largo plazo respecto del corto plazo.
- B. Es un valor entre 0 y 1 que indica la importancia del largo plazo respecto del corto.
- C. Es un valor entre 0 y 1 que indica la importancia de las instancias.

10 ✓

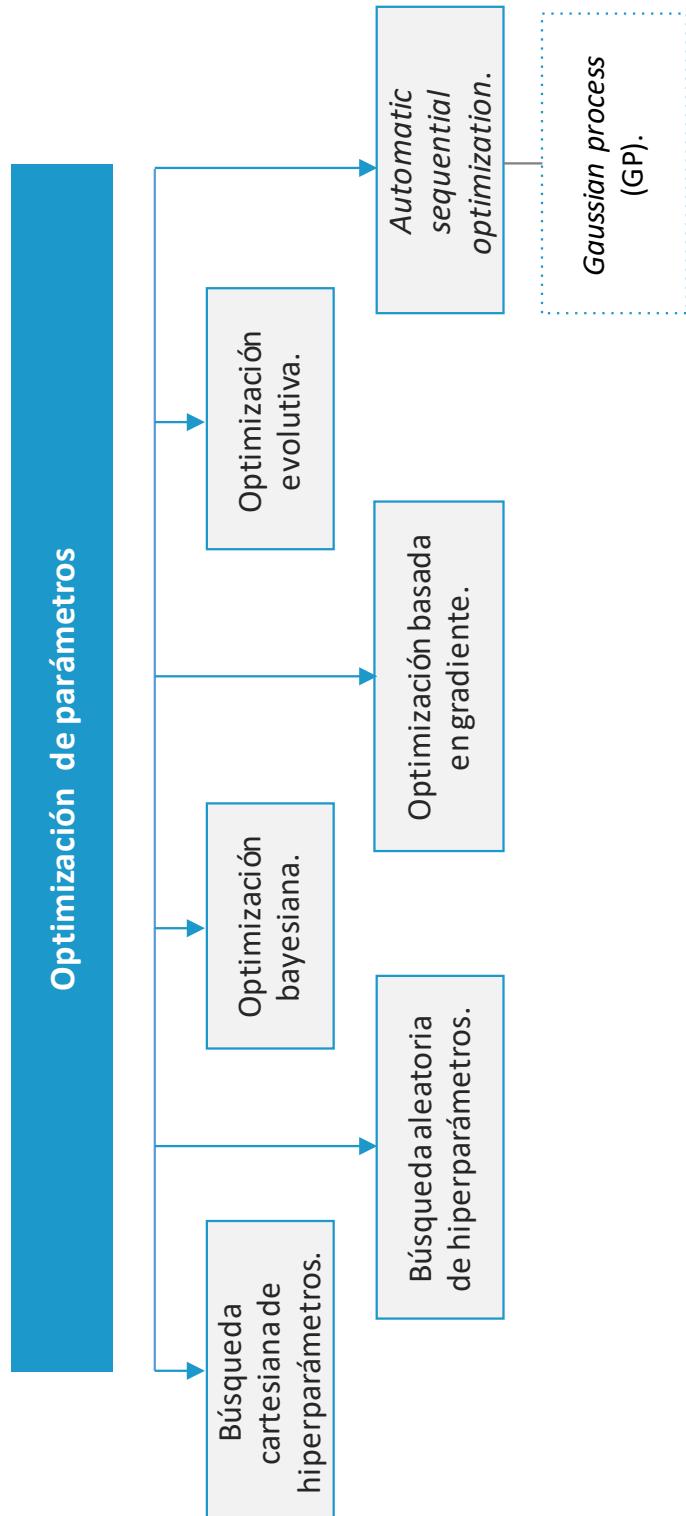
Aprendizaje Automático

Parametrización automática y optimización de algoritmos

Índice

Esquema	3
Ideas clave	4
13.1. ¿Cómo estudiar este tema?	4
13.2. Concepto de hiperparámetros	4
13.3. Búsqueda cartesiana de hiperparámetros	7
13.4. Búsqueda aleatoria de hiperparámetros	8
13.5. Mejora de la búsqueda de hiperparámetros	11
13.6. Referencias bibliográficas	13
Lo + recomendado	14
+ Información	16
Test	17

Esquema



13.1. ¿Cómo estudiar este tema?

Estudia este tema a través de las **Ideas clave** disponibles a continuación.

En 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

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 (*hyperparamters* 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**.

13.3. 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:

$$\begin{aligned} C &\in \{10, 100, 1000\} \\ \gamma &\in \{0.1, 0.2, 0.6\} \end{aligned}$$

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.

13.4. Búsqueda aleatoria de hiperparámetros

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 pérdida. 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.

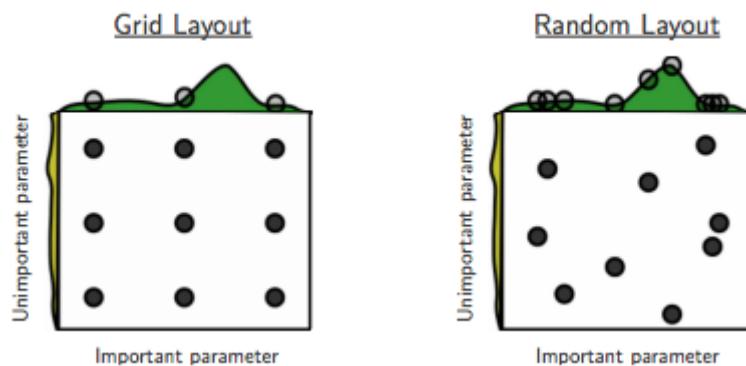
Número de modelos

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.

13.5. Mejora de la búsqueda de hiperparámetros

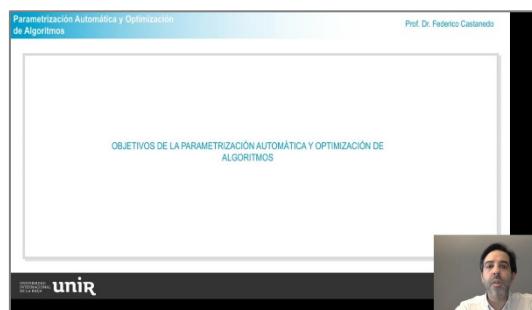
Una forma de mejorar la búsqueda aleatoria de parámetros es elegir conjuntos de parámetros que cubren el espacio de forma más eficiente que si se eligen de forma aleatoria. Bergstra y Bengio realizaron esta prueba y encontraron una mejora potencial pero únicamente cuando se hacían búsquedas de entre 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*).

En el siguiente vídeo se van a discutir los objetivos y necesidades de la búsqueda de hiperparámetros en la fase de construcción de modelos.



Objetivos de la parametrización automática y optimización de algoritmos.

Accede al vídeo a través del aula virtual

13.6. 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.

Lo + recomendado

No dejes de leer

Ajuste distribuido de hiperparámetros con Azure Machine Learning Workbench

Microsoft Azure (20 de septiembre de 2017). Ajuste distribuido de hiperparámetros con Azure Machines Learning Workbench.

Entrada de Microsoft Azure en castellano donde se describe como realizar el ajuste de hiperparámetros de algoritmos que implementan la API de Scikit-learn.

Accede a la entrada a través del aula virtual o desde la siguiente dirección web:

<https://docs.microsoft.com/es-es/azure/machine-learning/preview/scenario-distributed-tuning-of-hyperparameters>

Ramdon Search for Hyper-Parameter Optimization

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

Artículo de Bergstra y Bengio donde demuestran que el método de búsqueda aleatorio es más eficiente que la búsqueda cartesiana o manual.

Accede al artículo a través del aula virtual o desde la siguiente dirección web:

<http://www.jmlr.org/papers/volume13/bergstra12a/bergstra12a.pdf>

No dejes de ver

Cómo buscar los mejores parámetros de un modelo en scikit-learn

Vídeo tutorial que explica como buscar los hiperparámetros de un modelo de aprendizaje automático utilizando los métodos de búsqueda cartesiana y aleatoria de la librería scikit-learn.

Efficiently searching for optimal tuning parameters

From the video series: [Introduction to machine learning with scikit-learn](#)

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

https://www.youtube.com/watch?v=Gol_qOgRqfA

Optimizando parámetros usando *gridsearch* de scikit-learn

Vídeo tutorial donde se muestran los conceptos de *gridsearch* de scikit-learn para optimizar modelos.

Tuning Machine Learning Parameters using scikit-learn Gridsearch

Kevin Goetsch

Accede al vídeo a través del aula virtual o desde la siguiente dirección web:

<https://www.youtube.com/watch?v=wOqRaHSXYw>

Bibliografía

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.

1. Los hiperparámetros:

- A. Son específicos de cada modelo.
- B. Son generales para todos los modelos.
- C. Sirven para mejorar solo el conjunto de test.

2. Los hiperparámetros:

- A. Sirven para optimizar los modelos.
- B. Su optimización puede generar sobreajuste a los datos de entrenamiento.
- C. Su optimización nunca genera sobreajuste pues no modifican los datos de entrenamiento.

3. El método *cartesian grid search*:

- A. Consiste en una búsqueda aleatoria de los parámetros.
- B. Consiste en una búsqueda exhaustiva de los parámetros.
- C. Consiste en una búsqueda estocástica de los parámetros.

4. La optimización de hiperparámetros:

- A. Permite controlar el *overfitting*.
- B. Permite controlar el *underfitting*.
- C. Ninguna de las anteriores es correcta.

5. El método de *cartesian grid search*:

- A. No se puede paralelizar.
- B. Se puede paralelizar.
- C. Se puede calcular de forma distribuida.

6. El método de optimización evolutiva:

- A. Consiste en ejecutar algoritmos evolutivos para encontrar el mejor conjunto de parámetros.
- B. Es una evolución del método cartesiano.
- C. Es una evolución del método bayesiano.

7. Para la mayoría de los conjuntos de datos:

- A. Todos los hiperparámetros son relevantes.
- B. Solo algunos de los hiperparámetros son relevantes.
- C. La relevancia o no de los hiperparámetros depende del volumen de los datos.

8. El número de modelos necesarios para converger a un óptimo global:

- A. Por lo general es siempre el mismo.
- B. Puede variar bastante y el criterio de parada lo debe considerar.
- C. No es algo que debe ser tenido en cuenta.

9. Los modelos de procesos gaussianos:

- A. Son un método de optimización de hiperparámetros basado en búsqueda cartesiana.
- B. Se utilizan para optimizar parámetros de modelos complejos.
- C. Son un tipo de *automatic sequential optimization*.

10. La búsqueda aleatoria de hiperparámetros:

- A. Siempre es suficiente.
- B. En algunos modelos complejos puede ser insuficiente.
- C. Que sea suficiente o insuficiente depende de los datos de entrenamiento.

7