Capítulo 1
The Machine Learning Landscape

¿Qué es el Aprendizaje Automático?

Es la ciencia (y el arte) de programar computadores para que puedan *aprender de los datos*.

O de forma orientada a la ingeniería:

*Se dice de un programa de ordenador que aprende de la experiencia **E** con respecto a alguna tarea **T** y alguna medida de rendimiento **P**, si su rendimiento en **T**, medido por **P**, mejora con la experiencia **E**.*

- Los ejemplos que un sistema usa para aprender se llama **conjunto de entrenamiento**.

- Cada ejemplo de entrenamiento se llama instancia de entrenamiento (o **muestra**).

- La parte de un sistema de aprendizaje automático que aprende y hace predicciones se llama **modelo**.

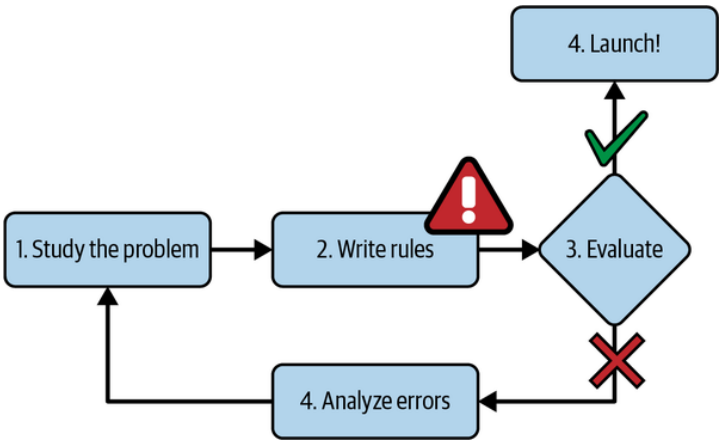
Las redes neuronales y random forest son ejemplos de modelos.

Ejemplo de email:

T = *Marcar el spam* para nuevos correos electrónicos.
E = *Datos de entrenamiento*.
P = *Precisión* o medida de rendimiento que debe definirse (podrían usarse correos correctamente clasificados previamente).

¿Por qué usar el aprendizaje automático?

¿Cómo haría un filtro de spam usando técnicas de programación tradicionales?



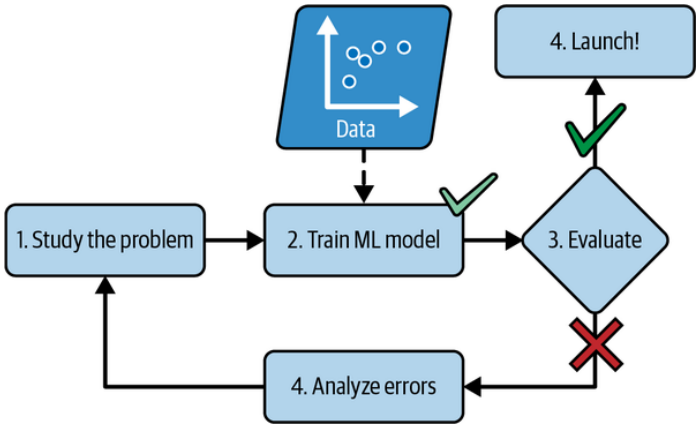
1. Observar cómo se vería normalmente el spam. Se pueden observar patrones en el vocabulario de los emails como la repetición de palabras/frases tipo: "*4U*", *"tarjeta de crédito", "gratis", "increíble"…*

2. Escribo un algoritmo de detección para cada uno de los patrones que observé en los emails haciendo que marcase los correos que llevaran estas palabras.

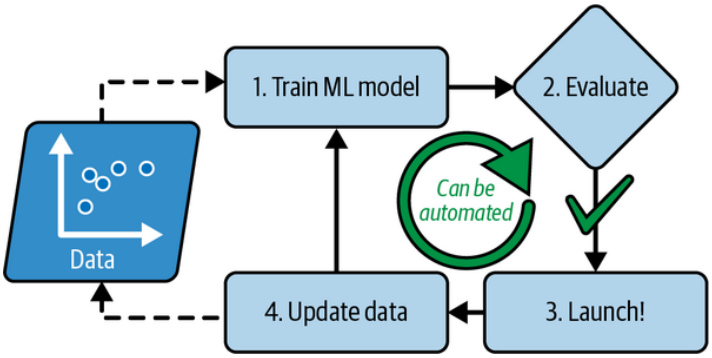
3. Ejecuto el programa y compruebo si es lo suficientemente bueno. Si no lo es, vuelvo al paso 1.

El programa puede convertirse en una larga lista de reglas muy *complejas de mantener* en el tiempo.

Por el contrario, un filtro de spam basado en aprendizaje automático, detecta automáticamente qué palabras y frases son buenos predictores de spam.
Esto hace que el programa sea más corto y fácil de mantener.



En un filtro de spam tradicional tendríamos que estar actualizando manualmente todas las nuevas palabras o expresiones que van apareciendo en los emails mientras que en sistemas basado en aprendizaje automático, esto se hace de forma automática.



Otra área en la que funciona mejor el aprendizaje automático que las técnicas tradicionales de programación es el **reconocimiento de voz**.

Ejemplo: Queremos construir un programa que distinga entre las palabras “uno” y “dos”.

“Dos” comienza con un sonido de todo alto “T”, por lo que se podría programar un algoritmo que mida la intensidad del sonido en tono alto.

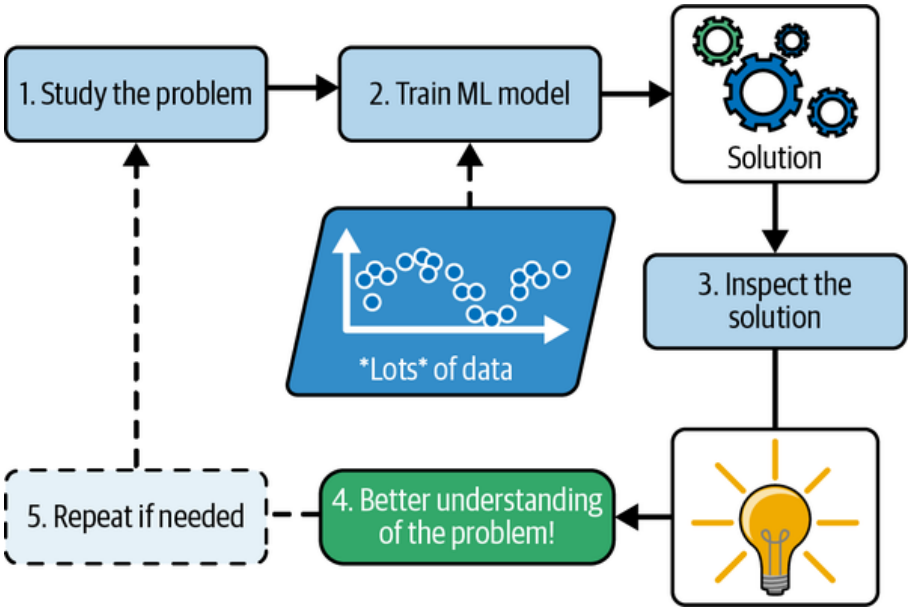
Pero esto no es nada preciso ya que posiblemente se equivocaría con una multitud de palabras y no funcionaría en espacios con ruido de fondo.

Lo más efectivo sería construir un algoritmo que aprende por sí mismo y que sea alimentado con varias grabaciones de voz diciendo esta palabra.

Los modelos de ML se pueden inspeccionar para ayudar a los humanos descifrar qué ha aprendido la máquina.

Por ejemplo, una vez entrenado el filtro de spam se puede inspeccionar para ver qué palabras ha aprendido.
Esto lleva a nuevas correlaciones y por tanto se puede comprender mejor el problema.

La **minería** es la excavación en grandes cantidades de datos para descubrir nuevos *patrones ocultos.*



El aprendizaje automático es ideal para:

- Problemas donde las soluciones requieran ajustes muy finos o largas listas de reglas.
- Problemas complejos en los que el uso de un enfoque tradicional no sirve.
- Entornos fluctuantes.
- Obtener información sobre problemas complejos y grandes cantidades de datos.

Ejemplos de aplicaciones:

- Analizar imágenes de productos en una línea de producción para clasificarlos automáticamente (**CNN**).
- Detección de tumores cerebrales (**CNN, transformadores**).
- Clasificación automática de noticias (**NLP, RNN, CNN, transformadores**).
- Marcar automáticamente los comentarios ofensivos en foros (**PNL**).
- Resumiendo documentos largos (**PNL**).
- Crear un chat bot (**PNL, NLU**).
- Pronosticando ingresos de una empresa (**RNN, CNN, transformadores**).
- Reacción a comandos de voz (**RNN, CNN, transformadores**).
- Detección de fraudes en tarjetas de crédito (**Isolated Forest, Gaussian mix**).
- Segmentar a clientes en función de sus compras (**k-means, DBSCAN, …**).
- Representar un conjunto de datos complejo y de alta dimensión en un diagrama claro y perspicaz.
- Recomendar un producto para el cliente (**NN**).
- Construyendo un bot inteligente para un juego (**RL**).

TIPOS DE SISTEMAS DE APRENDIZAJE AUTOMÁTICO

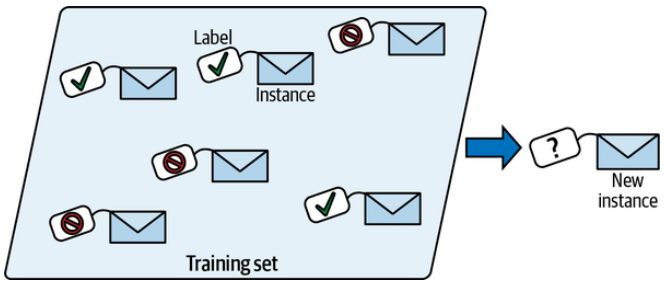
Criterios de clasificación:

- Supervisión:
  - Supervisados
  - No supervisados
  - Semi supervisados
  - Otros
- Aprendizaje incremental sobre la marcha:
  - Online
  - Batch (por lotes)
- Funcionan comparando nuevos datos con datos conocidos.
- Funcionan detectando patrones en los datos de entrenamiento y construyendo un modelo predictivo.

Supervisión de la información

Aprendizaje supervisado

El conjunto de datos de entrenamiento (*dataset*) **incluye las soluciones** (*targets*).



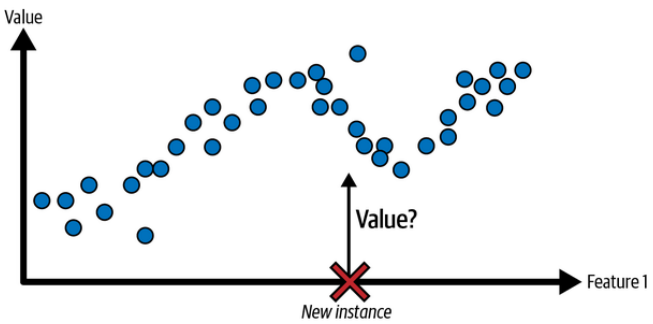
Una tarea típica del aprendizaje supervisado es la *clasificación*.

Ejemplos:

- Clasificador de spam que se entrena con un conjunto de datos de spam.

- Predecir el precio (valor numérico) de un un coche en base a características (**regresión lineal**).

Algunos modelos de regresión también se pueden usar para clasificar (**regresión logística**) y viceversa ya que la regresión logística puede producir un valor que corresponde a la **probabilidad** de pertenecer a una clase determinada.

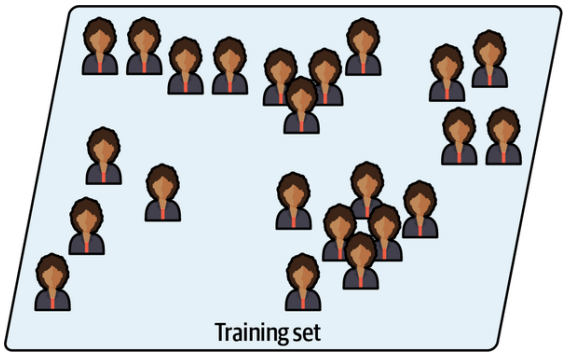


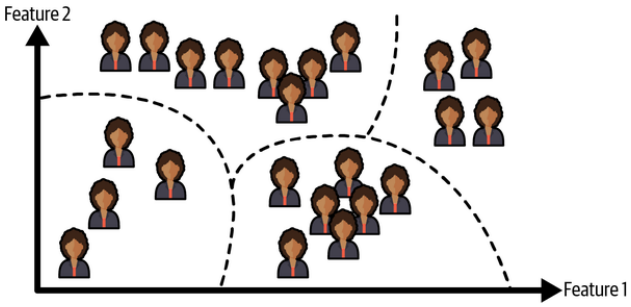
Aprendizaje NO supervisado

Los datos de entrenamiento **no están etiquetados** (no tienen solución), es decir, el sistema intenta aprender sin un profesor.

Ejemplo: Tenemos muchos visitantes a un blog.

Es posible crear un algoritmo para ***agruparlos*** en visitantes similares.
No se le dice al algoritmo a qué grupos pertenecen esos visitantes sino que él trata de encontrar los grupos y clasificarlos por sí mismo.



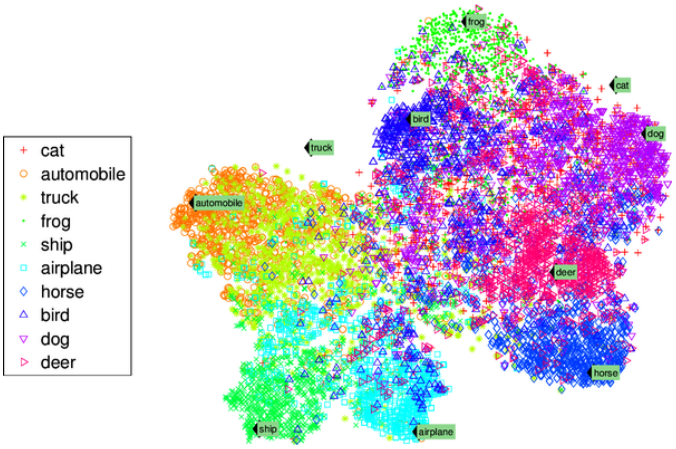


El 40% de los visitantes son adolescentes que aman los comics.
El 20% de los visitantes son adultos que aman la ciencia ficción.
…

Los algoritmos de visualización también son buenos ejemplos aprendizaje no supervisado.
Se les suministra una gran cantidad de datos complejos y sin etiquetar producen una presentación 2D/3D de los datos que se puede trazar fácilmente.

Una tarea relacionada es la ***reducción de la dimensionalidad***, en la que el objetivo es simplificar los datos sin perder mucha información.
Se suele fusionar varias características correlacionadas en una sola.

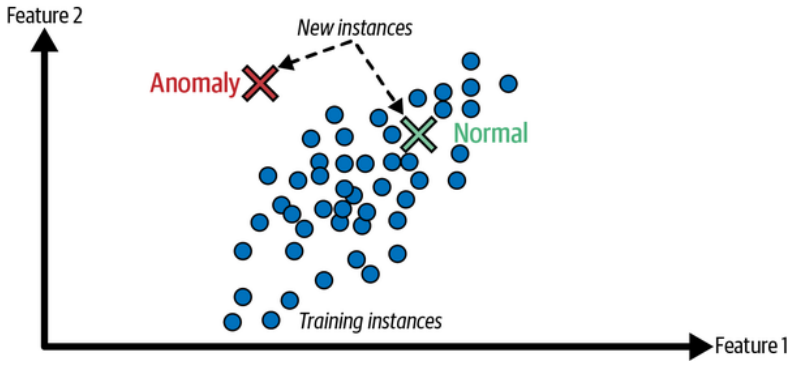
Ej: **Kilometraje de un coche**, que está fuertemente relacionado con su **edad**, se fusionan en una sola característica llamada **desgaste**.



Esta tarea de reducción de la dimensionalidad se recomienda hacer antes de suministrar los datos a un algoritmo de aprendizaje automático supervisado.

Otra tarea importante para la no-supervisada es la ***detección de anomalías***, por ejemplo detectar transacciones inusuales con tarjetas de crédito para evitar fraude o eliminar los valores atípicos de un conjunto antes de alimentar el algoritmo.

Se le entrena con instancias correctas y posteriormente puede predecir si las nuevas tienen anomalías.



Una tarea similar es la ***detección de novedades*** que detecta nuevas instancias que se ven completamente diferentes a las del conjunto de entrenamiento.

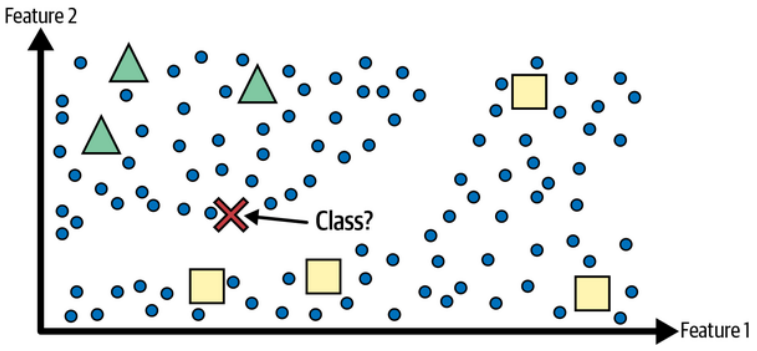
Esto requiere un conjunto de entrenamiento muy limpio y puede considerar a cierto tipo de perros como “raros" o diferentes como por ejemplo un chihuahua sólo por ser un poco diferentes.

Y por último otra tarea común es el ***aprendizaje de reglas de asociación***, donde el objetivo principal es descubrir relaciones interesantes entre atributos.

Aprendizaje semiautomático

El etiquetado de los datos lleva bastante tiempo y es costoso, por lo que a menudo habrá muchas instancias sin etiquetar y pocas instancias etiquetadas.

Es decir, ***datos parcialmente etiquetados***.



Aprendizaje semisupervisado con dos clases (triángulos y cuadrados): los ejemplos sin etiquetar (círculos) ayudan a clasificar una nueva instancia (la cruz) en la clase de triángulo en lugar de la clase cuadrada, a pesar de que está más cerca de los cuadrados etiquetados

Un buen ejemplo de esto es Google Pixel Photos:

- **Parte no-supervisada** (agrupación): Una misma persona A, que no conocemos, sale en las fotos 1, 5 y 11.

- **Parte supervisada**: Solo hay que decir quién es esta persona como etiqueta para que sea útil a la hora de buscar fotos.

La mayoría de estos algoritmos son una mezcla de algoritmos de aprendizaje supervisado con no-supervisado.

Aprendizaje autosupervisado

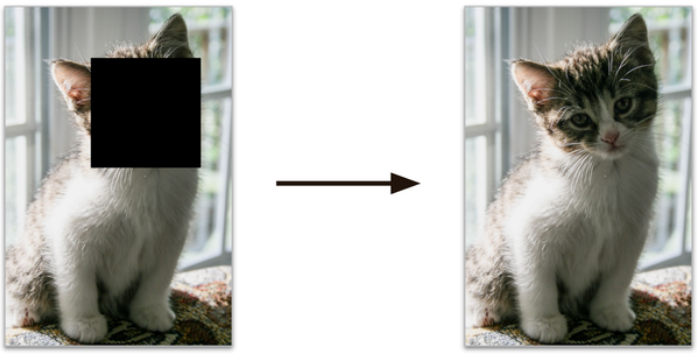
Genera un conjunto de datos completamente etiquetado a partir de uno completamente no etiquetado.

Una vez se haya etiquetado todo el conjunto de datos, se puede usar cualquier algoritmo de aprendizaje supervisado.

Ejemplo:

Si tenemos un gran conjunto de datos de imágenes sin etiquetar, se puede enmascarar al azar una pequeña parte de cada imagen y luego entrenar un modelo para recuperar la imagen original.

Las imágenes enmascaradas son las entradas para entrenamiento y las imágenes originales se usan como etiquetas.



El modelo resultando es útil, por ejemplo, para reparar imágenes dañadas o para borrar objetos no deseados pero la mayoría de las veces el aprendizaje autosupervisado no es el objetivo final.

Ejemplo: Modelo de clasificación de mascotas.

Si le damos una imagen de una mascota, predecirá la especie a la que pertenece.

Si tenemos un gran conjunto de datos sin etiquetar de mascotas, puedo empezar entrenando un modelo de reparación de imágenes usando aprendizaje autosupervisado.

Una vez que funcione bien podrá distinguir diferentes especies.

Se puede ajustar la red neuronal (la mayoría lo permiten) para que, en vez de dibujar la cara de un perro en el hueco negro, prediga la especie a la que pertenece.

Como paso final se ajusta el modelo en un conjunto de datos etiquetados.

La diferencia entere el aprendizaje no supervisado y el autosupervisado es que el autosupervisado utiliza **etiquetas** (generadas) **durante la formación**, por lo que en ese sentido está más cerca del supervisado.

Aprendizaje por refuerzo

Este, es un tipo de algoritmo completamente diferente a los vistos anteriormente.

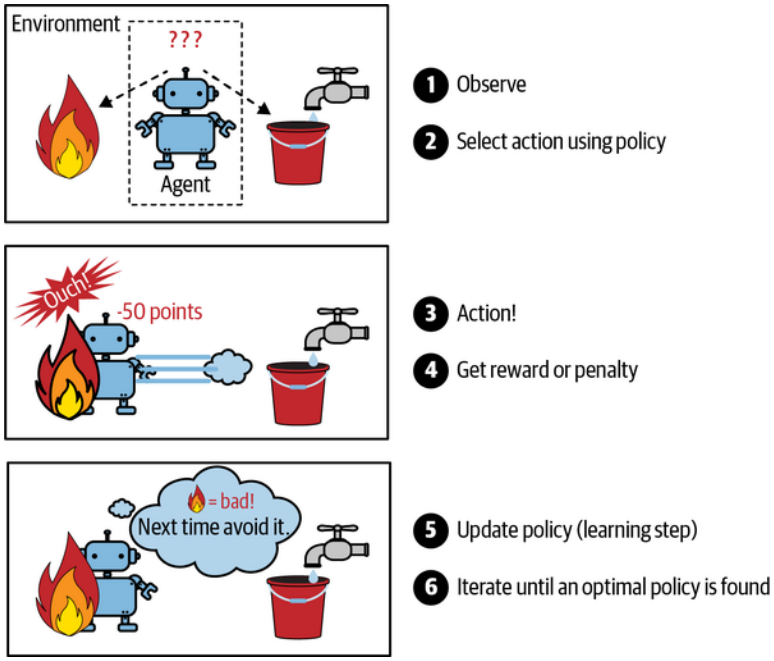
El sistema de aprendizaje se denomina ***agente*** y puede:

- Observar en entorno
- Realizar acciones
- Obtener recompensas a cambio (o sanciones)

Luego debe de aprender por sí mismo cuál es la mejor estrategia, llamada ***política***, para obtener la mayor recompensa en el tiempo.

Política: Define qué acción debe elegir el agente cuando se encuentra en una situación determinada.

Un ejemplo de esto es **AlphaGo** (aprendizaje offline).



Aprendizaje en lotes (batch) vs en línea (online)

El sistema puede puede aprender de forma incremental (o no) de un flujo de datos entrantes?

Aprendizaje por lotes

El sistema es ***incapaz de aprender de forma incremental*** por lo que debe ser entrenado usado todos los datos disponibles.

Esto lleva mucho tiempo y recursos informáticos.

Primero se entrena el sistema y luego se lanza a producción sin aprender más, simplemente aplicando lo que ha aprendido previamente. (Aprendizaje **offline**).

Si se entrena un modelo con esta técnica, su rendimiento va a ir cayendo con el tiempo ya que este permanece sin cambios mientras que el mundo continúa evolucionando.

El remedio a esto es estar entrenando el mimo modelo continuamente pero esto es costoso.

Si se requiere que el sistema de aprendizaje por lotes reconozca nuevos datos (nuevos tipos de spam) se deberá volver a entrenar el modelo desde 0 y sustituir por completo al antiguo.

Llegará a un punto en el que la cantidad de datos sea tan grande que será imposible el manejo de estos.

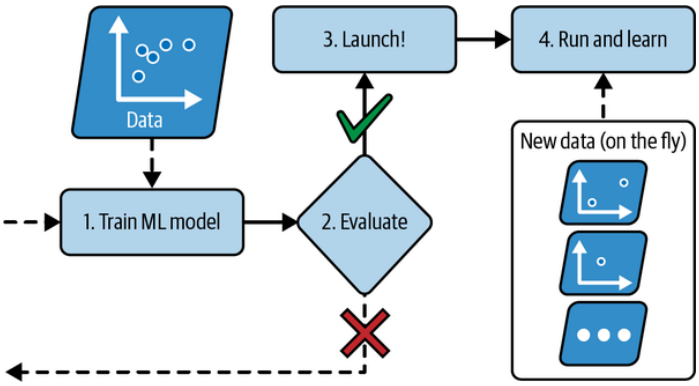
Si necesitamos un sistema que se adapte a los cambios rápidamente, esta no es la mejor técnica (es necesaria una técnica más reactiva).

- **Ventajas**: Fácil de entrenar.
- **Desventajas**: Poca escalabilidad y consumo de recursos muy elevado.

Aprendizaje online

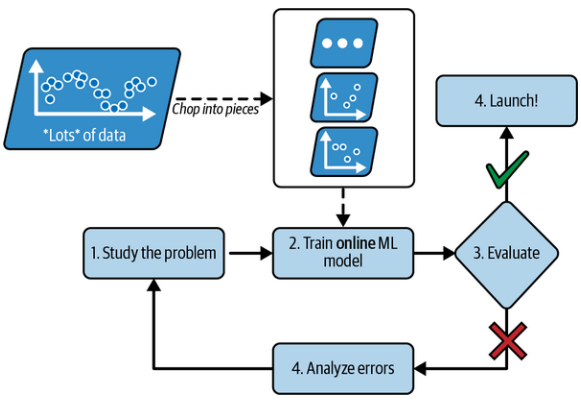
Se ***entrena el sistema de forma incremental*** alimentando con instancias de otros datos de forma secuencial, ya sea individualmente o en pequeños grupos llamados ***mini-batches***.

Cada incremente o paso en el aprendizaje es rápido y barato, por lo que aprende de los datos sobre la marcha (*on the fly*).



Útil para sistemas que se deben adaptar a las circunstancias muy rápido o que tienen recursos informáticos bastante limitados (smartphone).

Ademas, también se puede usar para entrenar con enormes cantidades de datos que no caben en memoria principal de la propia máquina (*entrenamiento descentralizado*).



- **Ventajas**: Rapidez adaptándose a los nuevos cambios (***tasa de aprendizaje***).
    - **Alta tasa de aprendizaje** = Adaptación muy rápida pero se olvidan los datos antiguos.
    - **Baja tasa de aprendizaje** = Adaptación muy lenta pero se mantienen los datos antiguos.
- **Desventajas**: Si se suministran datos defectuosos al sistema, el rendimiento disminuirá muy rápidamente (dependiendo de la calidad de los datos y la tasa de aprendizaje).

Es posible que se desee monitorizar los nuevos datos de entrada para reaccionar ante las anomalías que hacen que decaiga el rendimiento, usando algún *algoritmo de detección de anomalías*.

Aprendizaje basado en instancias VS aprendizaje basado en modelos

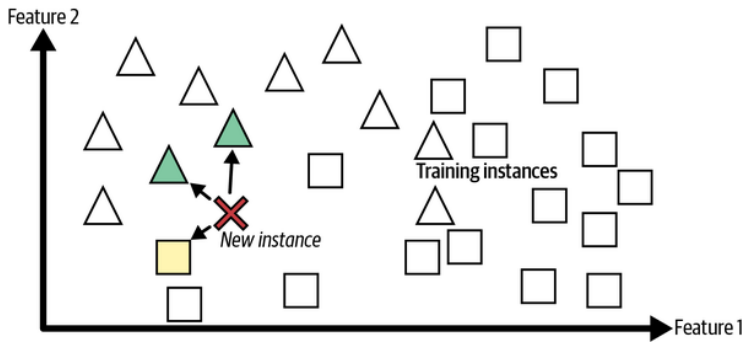
Otra forma de categorizar los sistemas de aprendizaje es por como *generalizan*.

Dada una serie de ejemplos de entrenamiento, el sistema debe ser capaz de hacer una predicción correcta (generalizar).

No basta con predecir correctamente sino predecir bien en todos los casos por muy distintos que sean.

Aprendizaje basado en instancias

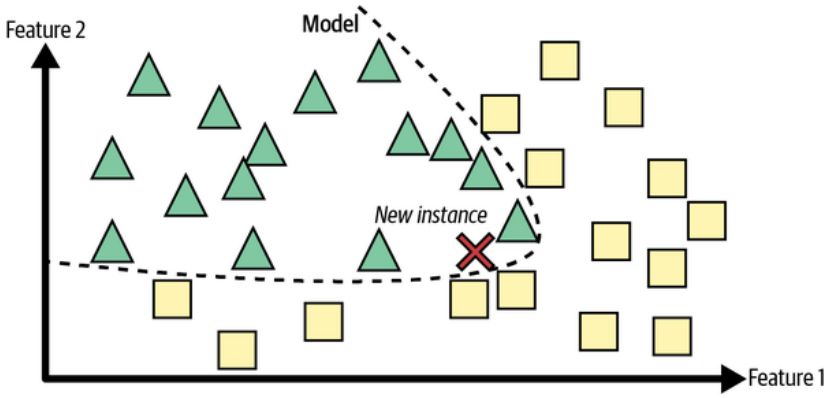
El sistema aprende ejemplos de memoria y luego se generaliza para nuevos casos usando una *medida de similitud* para compararlos con los ejemplos aprendidos.



(La nueva instancia se clasificaría como un triángulo porque la mayoría de instancias similares pertenecen a esa clase).

Aprendizaje basado en modelos y **flujo de trabajo típico de aprendizaje automático**

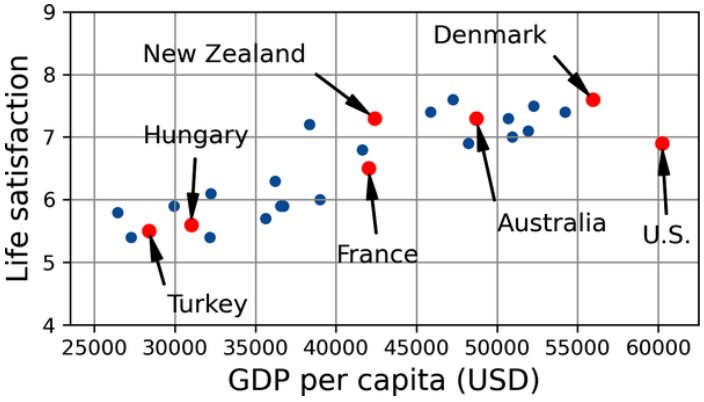
Otra forma de generalizar a partir de un conjunto de ejemplos es construir un modelo de estos ejemplos y luego usar ese modelo para hacer *predicciones*.



Ejemplo: Queremos saber si el dinero hace feliz a la gente.

Tabla 1-1. ¿El dinero hace más feliz a la gente?

país	PIB per cápita (USD)	Satisfacción con la vida
Turquía	28.384	5.5
Hungría	31.008	5.6
Francia	42.026	6,5
Estados Unidos	60.236	6.9
Nueva Zelanda	42.404	7.3
Australia	48.698	7.3
Dinamarca	55.938	7,6



Parece que existe una tendencia y, aunque los datos son ruidosos (en parte aleatorios), la satisfacción con la vida aumenta de forma lineal a medida que aumenta el PIB per cápita del país.

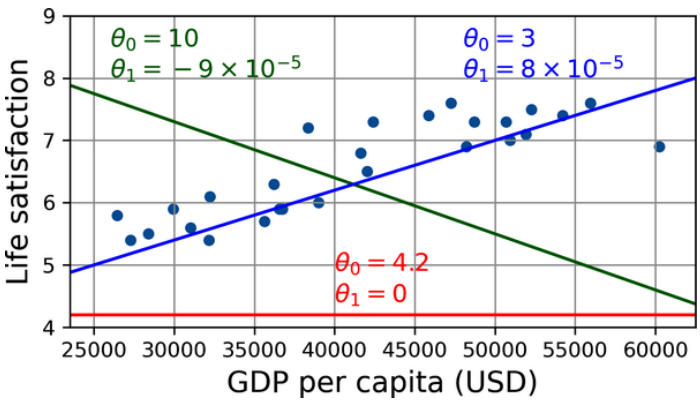
Podemos modelar el gráfico con una línea (**selección del modelo**).

- **Ecuación 1.1: Modelo Lineal Simple**

$$\text{life\_satisfaction} = \theta_0 + \theta_1 \times \text{GDP\_per\_capita}$$

Este modelo tiene dos parámetros, theta **0** y theta **1**.

Al ajustar este modelo se puede representar cualquier función lineal, por ejemplo:



Antes de usar el modelo hay que definir los valores para los dos parámetros theta 0 y theta 1.

Qué valores para los thetas son los mejores? Primero hay que establecer una forma de medir el rendimiento (*función utilidad o* ***función de costo***).

Normalmente en regresión lineal se suelen usar:

- Para **medir** el rendimiento: Función de coste que mide las distancias entre los puntos (MSE).
- Para **minimizar** la distancia: Descenso del gradiente, fórmula original para R.L, …

Entrenar el modelo es pasar al algoritmo de regresión lineal los datos y obtener una línea que minimice la distancia entre los puntos.

Entrenar es encontrar los theta (o pesos) que minimizan el error del modelo.

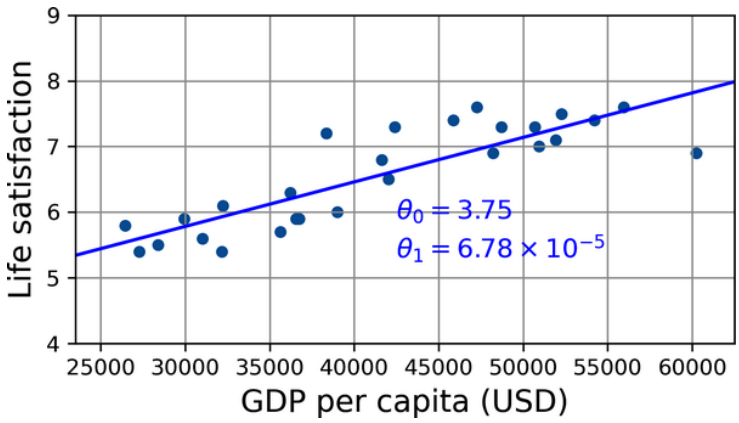
Una vez lo tenemos entrenado obtenemos las theta (o pesos) y el sesgo (b) que forman la recta.

Por ejemplo:

* Theta 0 = 3.75
* Theta 1 = 6.78x10^-5

Por lo tanto, la recta del tipo: **y = O0*x1 + O1*x3 + b**, quedaría como:

y = 3.75 *x1 + 6.78x10^-5*x3 + b



(Esa es la recta que minimiza el error del modelo)

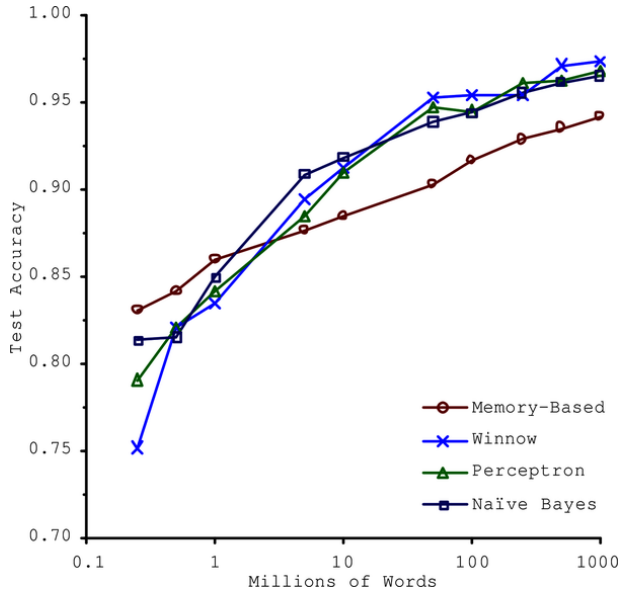
Principales desafíos del aprendizaje automático

Cantidad insuficiente de datos de entrenamiento

Para que un niño pequeño aprenda qué es una manzana es necesario que antes haya visto multitud de manzanas antes de multitud de formas y colores.

Normalmente se necesitan miles o millones de ejemplos incluso para problemas muy simples.

Aunque los conjuntos de datos pequeños y medianos siguen siendo muy comunas ya que no siempre es fácil obtener esos conjuntos de datos adicionales.



Datos de formación no representativa

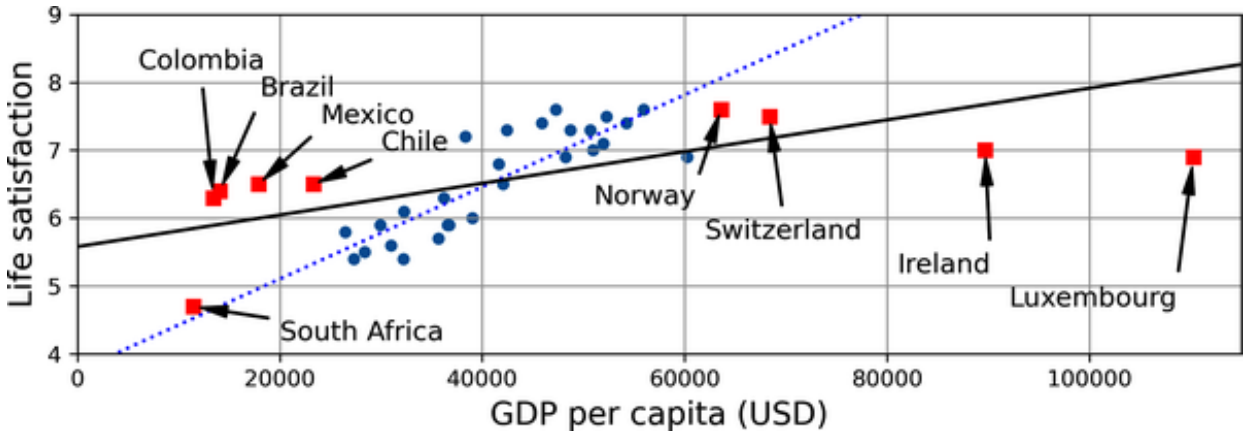
Para generalizar bien es imprescindible que los datos de entrenamiento sean representativos de los nuevos casos a los que desea generalizar.

Esto es cierto si se usa el aprendizaje basado en instancias o en modelos.

En el ejemplo anterior, si no hubiéramos tomado países con un PIB inferior a 23.500$ o superiores a 62.500$, la recta que se genera nunca va a predecir bien ya que se tiene que ajustar a un conjunto de datos menor que el anterior.

Obtenemos unas predicciones en las que los países muy pobres son más felices que los países ricos y eso claramente no es así.

Recuerda, cuanto más datos, mejores predicciones.



Si la muestra es demasiada pequeña tendrá ruido de muestreo pero incluso las muestras grandes pueden ser no representativas si el método de muestreo es defectuoso (***sesgo de muestreo***).

Ejemplo de sesgo de muestreo:

Digamos que queremos construir un sistema para reconocer los vídeos musicales de funk.

Una forma de construir el conjunto de entrenamiento es buscar “música funk” en YouTube y usar los vídeos resultantes.

Posiblemente el resultado que arroje esté sesgado hacia artistas populares, por ejemplo, si vivimos en Brasil obtendremos funk brasileño, que nada tiene que ver con James Brown.

El sesgo de muestreo es el equivalente a no barajar bien las cartas antes de tomar una.

Datos de mala calidad

Si los datos de entrenamiento están llenos de errores, valores atípicos y ruido (como mediciones de mala calidad), hará que sea más difícil para el sistema detectar los patrones subyacentes.

Es muy importante **limpiar** los datos antes de usarlos en el sistema.

Cuándo hay que limpiarlos?:

- Si algunas instancias son atípicas lo mejor es desecharlas.
- Si a algunas instancias les falta esa característica, se puede ignorar esa característica para todas las demás instancias o simplemente hacer la media de las demás y completarla con ella.

Características irrelevantes

El sistema solo conseguirá aprender si tenemos muchas características relevantes y muy pocas poco relevantes.

Es muy importante no pasarse con características ni tampoco quedarnos cortos.

A esto se le llama *ingeniería de características*, que implica con los siguientes pasos:

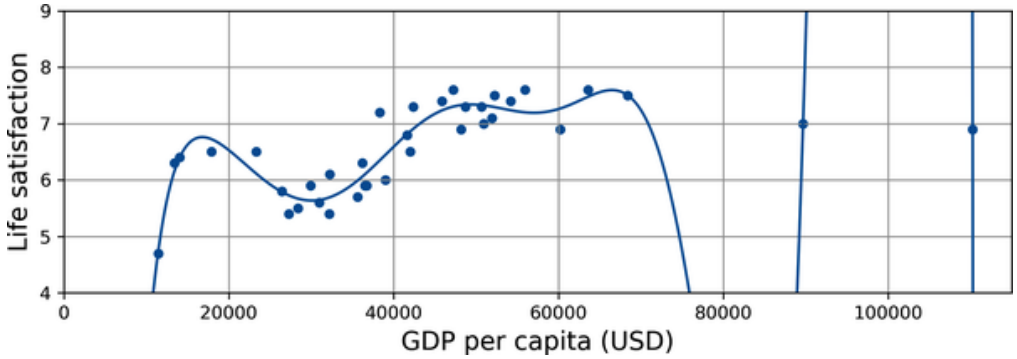
1. **Selección de características**: Usando sólo las más útiles entre las existentes para entrenar.
2. **Extracción de características**: Crear nuevas a partir de otras ya existentes (reducción de dimensionalidad).
3. **Creación de nuevas características** mediante la recopilación de datos nuevos.

Sobreajuste de los datos de entrenamiento

Digamos que vas a un país extranjero y el taxista te estafa. Podrías sentir la tentación de decir que todos los taxistas de ese país son ladrones.

Generalizar es algo en lo que caemos los humanos y las máquinas también pueden hacerlo.

Esto se llama ***overfitting*** y significa que el modelo funciona bien para los datos de entrenamiento pero NO generaliza bien.



Los modelos como las redes neuronales detectan patrones muy sutiles en los datos pero si el conjunto de datos de entrada es demasiado pequeño o es ruidoso, es muy probable que el modelo ajuste a ese ruido (no debe) en vez de generalizar.

El overfitting ocurre cuando el modelo es demasiado complejo en relación con la cantidad y el ruido de los datos de entrenamiento.

El modelo no sabe si ese dato es bueno o es ruido, simplemente intenta ajustarse a ello.

Por eso hay que limpiar y eliminar los datos irrelevantes antes de entrenar.

Posibles soluciones ante el overfitting:

* **Simplificar el modelo**: Usar un modelo con menos parámetros. En vez de usar un modelo polinómico de alto grado uso un modelo de regresión lineal simple, **reduciendo el número de atributos**.

* **Recopilar más datos de entrenamiento**.

* **Reducir el ruido de los datos de entrenamiento**: Corregir errores y eliminar valores atípicos.

La restricción de un modelo para hacerlo más simple y reducir el riesgo de sobreajuste se llama **regularización**.

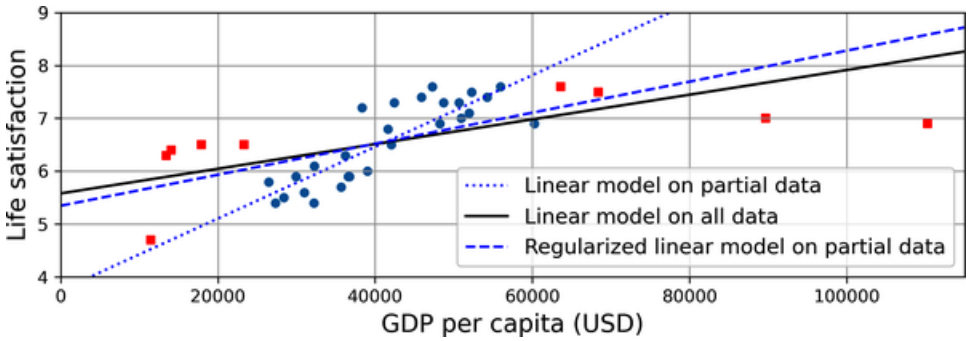
Por ejemplo, en el ejercicio anterior, el modelo lineal tenía dos parámetros θ0 (altura) y θ1 (pendiente).

Esto le da al algoritmo 2 grados de libertad.

- Si le quitamos la θ1 (pendiente), el modelo solo podrá predecir con una línea recta horizontal y lo único que podrá hacer será moverla arriba y abajo. Esto lo convierte en un modelo realmente simple. Demasiado.

- Si mantenemos ambas pero restringimos la θ1 (pendiente) para que se mantenga en valores muy pequeños, podremos conseguir un mejor ajuste a los datos, sin llegar al overfitting.

Hay que encontrar el equilibrio entre ajustar los datos de entrenamiento y mantener un modelo lo suficientemente simple.



La cantidad de regularización a aplicar durante el aprendizaje puede ser controlada por un ***hiperparámetro***.

Un hiperparámetro es un parámetro de un algoritmo de aprendizaje (no del modelo).

No se ve afectado por el algoritmo de aprendizaje en sí.

Debe establecerse antes del entrenamiento y es un valor fijo.

- Si hiperparámetro de regularización es muy grande -> Obtenemos un modelo casi plano (pendiente cercana a cero).

Mayor valor del hiperparámetro implica una mayor simplificación del modelo, lo que lleva a un modelo demasiado simple/plano para predecir si nos pasamos.

Subadaptar los datos de entrenamiento (underfitting)

Lo contrario al sobreajuste (overfitting) es el infra-ajuste (***underfitting***) y ocurre cuando el modelo es demasiado simple para aprender la estructura subyacente de datos.

Por ejemplo, un modelo de regresión lineal es demasiado simple para medir la felicidad de la población en base a determinadas características.

La realidad es más compleja que el modelo, por lo que sus predicciones serán inexactas.

Cómo se puede solucionar el underfitting?:

- Seleccionar un **modelo más potente y con más parámetros**.
- Alimentar el algoritmo con **mejores características**.
- **Reducir las restricciones** del modelo (ej: reducir el hiperparámetro de regularización).

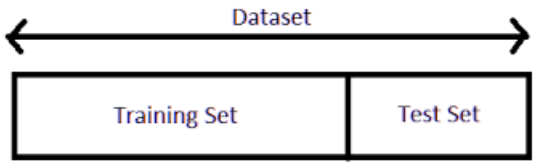
Pruebas y validación

Solo hay una forma de saber qué tan bien generaliza un modelo consiste en probarlo en nuevos casos.

Se puede poner el modelo en producción y controlar que tan bien funciona pero esto es muy arriesgado.

La mejor opción:

Dividir los datos en ***conjunto de entrenamiento*** (entreno con esto) y ***conjunto de pruebas*** (valido con esto).



El error que se obtienen al probar en los nuevos casos se llama **error de generalización** e indica que tan bien funciona el modelo.

Si error de entrenamiento es bajo y el error de generalización es alto, significa que el modelo está sobreadaptando o que tiene overfitting.

Ajuste de hiperparámetros y selección de modelos

Supongamos que estamos dudando entre dos modelos y no sabemos cuál nos combine más.

Uno es de regresión lineal y otro es un modelo polinómico.

Cómo diferenciamos entre ellos?

- Opción 1: Entrenar los dos modelos y ver lo bien que generalizan en el conjunto de pruebas

Supongamos que el modelo lineal generaliza mejor.
Aún así queremos aplicar un poco de regularización para evitar el overfitting.

Cómo se elige el valor del hiperparámetro de regularización?

- Opción 1: Entrenar 100 modelos diferentes usando 100 valores diferentes de hiperparámetro.

Supongamos que encontramos el mejor valor del hiperparámetro, el cuál tiene un error de solo el **5%**. Lanzas el modelo a producción pero no funciona tan bien, por lo que ahora tiene un error del **15%**.

Qué ha pasado?

El problema reside en que se midió el error de generalización varias veces en ese conjunto de pruebas y se adaptó el modelo e hiperparámetros para producir el mejor modelo ***para ese conjunto de pruebas en particular***.

Esto hace que el error sea bueno para el conjunto de pruebas pero no lo sea tanto en el mundo real.

SOLUCIÓN: **Validación por retención (holdout validaron)**

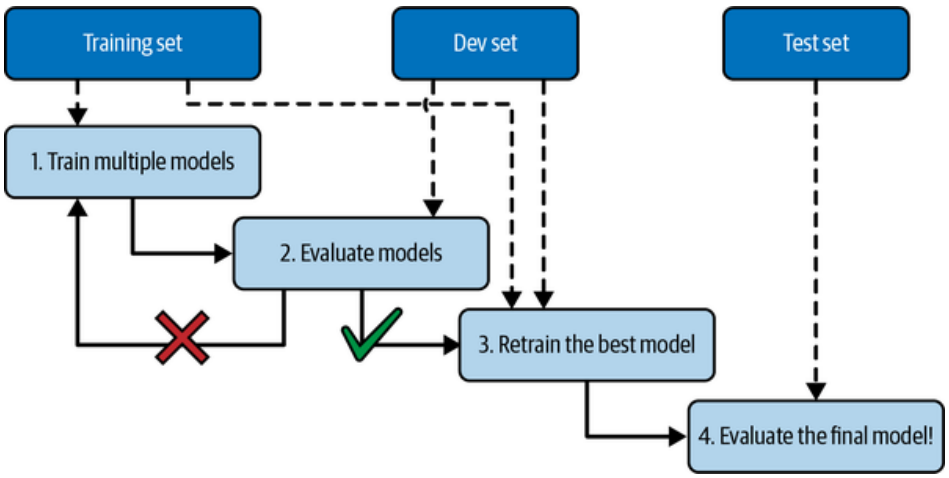
Se **mantiene parte del conjunto de entrenamiento** para evaluar varios modelos candidatos y seleccionar el mejor.

A ese nuevo conjunto retenido se le llama ***conjunto de validación*** (o conjunto de desarrollo <dev set>).

La evolución de la validación **holdout** es la validación **cruzada**.

Validación *holdout*: Es el método más simple de validación cruzada.

1. El conjunto de datos se separa en dos conjuntos, entrenamiento y prueba.
2. Se entrena usando únicamente el conjunto de datos de entrenamiento
3. Se le pide al modelo que haga predicciones sobre el conjunto de datos de prueba.
4. Los errores cometidos se acumulan para dar el *error medio total* del conjunto de prueba.
5. Este error se usa para evaluar al modelo.

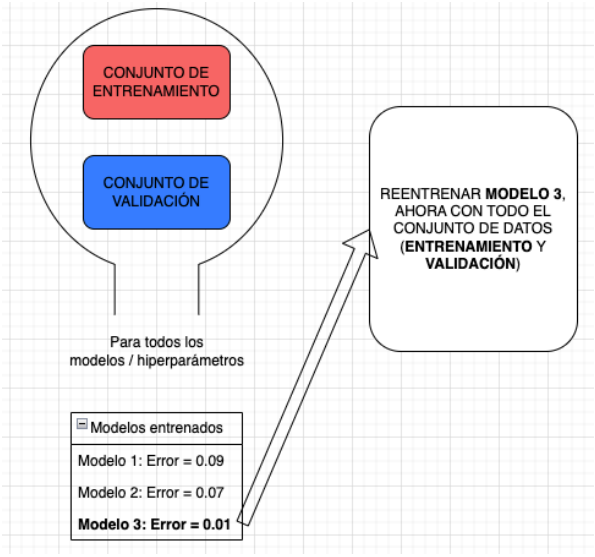


En resumen:

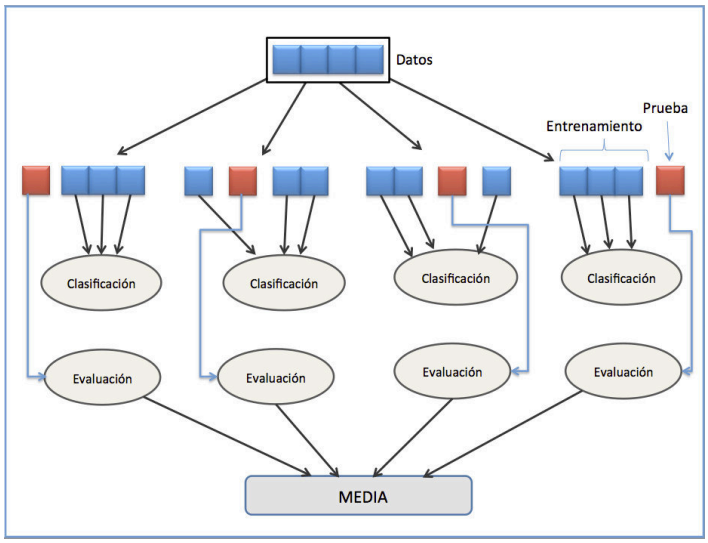
Tomas el set de datos de entrenamiento, lo entrenas y comparas con el set de validación.

Esto se hace para todos los modelos/hiperparámetros que queramos probar.

Se toma el modelo que ha entrenado con menos error y ahora se reestrena pero con todos los conjuntos de datos (validación y entrenamiento).



La alternativa a esto es la **validación cruzada**.



* **Ventajas de validación cruzada**: Se usan muchos conjuntos de datos de validación **pequeños** y cada modelo **se evalúa una vez por conjunto de validación** después de entrenar.

Se hace una media con todos los errores de las evaluaciones y se obtiene una medida mucho más precisa que holdout.

* **Inconveniente de validación cruzada**: El **tiempo** de entrenamiento se multiplica por el número de conjuntos de validación.

Desajuste de datos

Es fácil obtener una gran cantidad de datos pero difícil que estos sean perfectamente representativos respecto a los datos que se usarán en la producción.

Por ejemplo, queremos crear una aplicación que detecte el tipo de especie de flor al tomar una foto.

Podemos descargarnos un dataset de miles de fotos pero no serán tan representativas como las que podamos haber tomado con nuestra aplicación dentro de nuestro entorno.

Si a la hora de la verdad no hace buenas predicciones puede ser por:

- **Overfitting.**
- **Las imágenes del dataset no representan fielmente a las reales.**

Solución:

Mantener algunas de las imágenes de entrenamiento (de la web de descarga) en otro conjunto llamado *conjunto de desarrollo de entrenamiento*.

Después de entrenar el modelo (en el conjunto de entrenamiento y NO en el conjunto de desarrollo de entrenamiento) se puede evaluar en el conjunto de desarrollo entrenamiento.

Si el modelo funciona mal, entonces debe de haber overfitting para el **conjunto de entrenamiento** por lo que hay que tratar de simplificar o regularizar el modelo, obtener más datos de entrenamiento y limpiar los datos.

Pero si funciona bien en el **conjunto de desarrollo de entrenamiento** entonces se puede evaluar el modelo en el conjunto de desarrollo.

Si funciona mal, el problema puede venir de un desajuste de datos.

Lo ideal sería preprocesar esas imágenes que hemos descargado y solo tomar las que más se parezcan a nuestras flores reales.

Una vez obtenido un modelo que funciona bien tanto en el conjunto de desarrollo de entrenamiento como en el conjunto de desarrollo, se puede evaluar por última vez en el conjunto de prueba para saber qué tan bien es probable que funcione en producción.

Cuando los datos reales son escasos (derecha), puede usar datos abundantes similares (izquierda) para el entrenamiento y mantener algunos de los mismos en un conjunto de desarrollo de tren para evaluar el sobreajuste; los datos reales se utilizan para evaluar el desajuste de datos (conjunto de desarrollo) y para evaluar el rendimiento del modelo final (conjunto de pruebas)

TEOREMA DEL ALMUERZO SIN REGALO (nada es gratis):

Un modelo es una representación simplificada de los datos, por lo que se descartan los detalles superfluos que no van a generalizar para nuevas instancias.

El tipo de modelo nos dirá el tipo de datos que tenemos.

Si usamos un modelo lineal estamos asumiendo implícitamente que los datos son lineales y que la distancia entre las instancias y la línea recta es solo ruido, que se puede ignorar de forma segura.

David Wolpert demostró que si no haces absolutamente ninguna suposición sobre los datos, entonces no hay razón para preferir un modelo sobre cualquier otro.

A esto se le conoce como el teorema de **No Free Lunch** (NFL).

La única manera de saber qué modelo es mejor es probando con todos hasta dar con el ideal y como esto no es posible en al vida real (gasto de recursos y tiempo), se siguen unas directrices para saber qué modelo usar en cada tipo de dato.

- Problemas simples: Modelos lineales con varios niveles de regularización.
- Problema complejo: Varias redes neuronales.