Dan Beninson
Instituto de Tecnología Nuclear

Universidad Nacional de San Martín

Comisión Nacional de Energía Atómica

Nociones de Aprendizaje Automático para Aplicaciones Nucleares

Aprendizaje Supervisado

Presentación

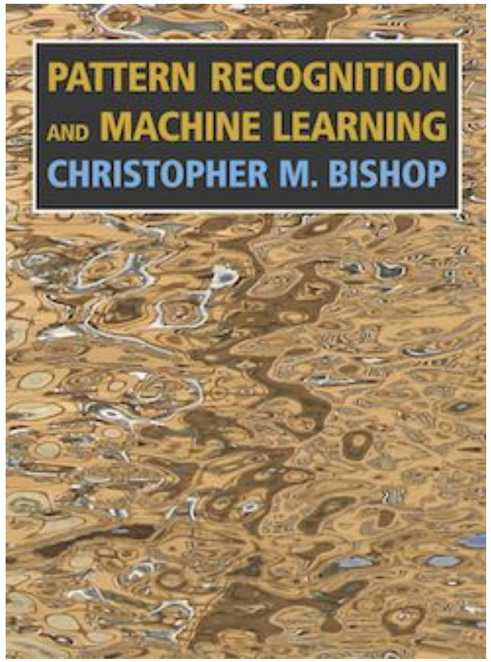
- Nosotros:
  - (Luis) Agustín Nieto: agustinnieto@cnea.gob.ar
  - Ana Lucía Marzocca: anamarzocca@cnea.gob.ar
- Metodología
  - Cuatro semanas (y una extra).
  - Teórica y práctica para hacer en clase.
  - Extras para revisar en casa (videos, notebooks, papers).
  - Evaluación la última clase (a definir).

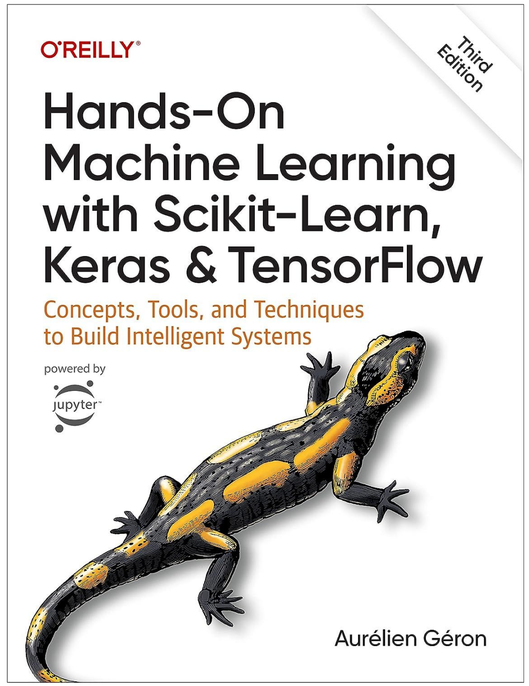
¿Qué NO es y qué SÍ es este curso?

- **NO**: “Curso de machine learning”.
- **SI**: Pensamiento crítico para aplicar aprendizaje automático en el ámbito científico:
  - “¿Puedo aplicar aprendizaje automático en este problema en particular”.
  - “¿Por donde empiezo?”.
  - “¿Cómo se llama lo que quiero hacer?”
  - “¿Qué datos necesito?”, “¿qué algoritmos puedo usar?
  - **“¿Conviene usar aprendizaje automático?. Qué se gana, que se pierde y como se usa adecuadamente.”**

Bibliografía

- Este curso está adaptado del curso "Diplomatura en Inteligencia Artificial y Ciencia de Datos" dictado por la ECyT-UNSAM en el marco del plan Argentina Programa 4.0.
- Bastante material genérico y específico, principalmente estos dos libros y los canales de youtube: DotCSV, StatQuest y 3Blue1Brown (revisen esos canales).



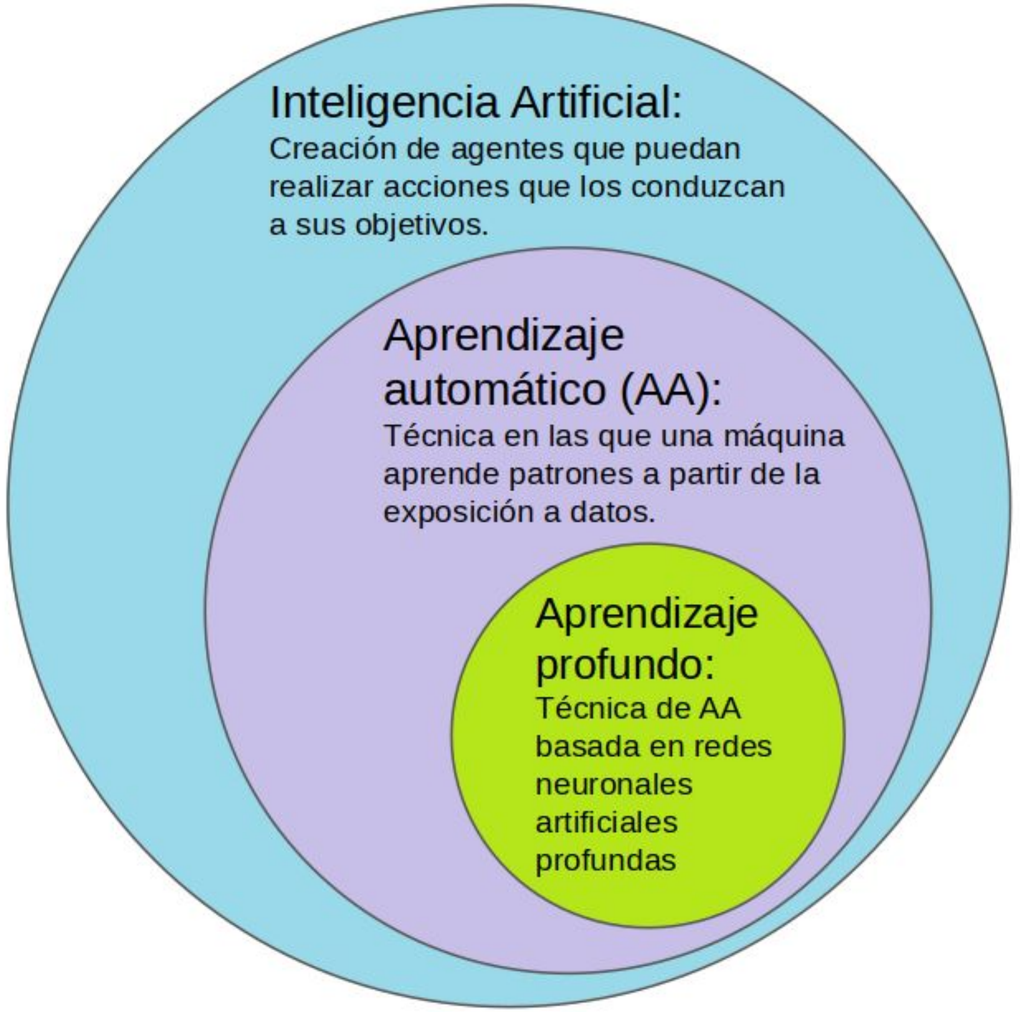


Herramientas ¿por donde empezar?

- Lenguajes: Python y R, líderes indiscutidos.
- Bibliotecas (además de numpy, pandas, etc.):
  - Scikit-learn para “*machine learning* clásico”.
  - Tensorflow + Keras / PyTorch para redes neuronales.
- Gestion de paquetes: Anaconda / Pip.
- *IDE’s*: Jupyter notebook, Visual Studio Code, Spyder.
  - Otra opción (hiper popular) trabajar en Google colab.

Visual Studio Code

¿Qué es el aprendizaje automático?



¿Qué es el aprendizaje automático?

"El aprendizaje automático es el campo de estudio que da a los ordenadores la capacidad de aprender sin ser programados explícitamente."

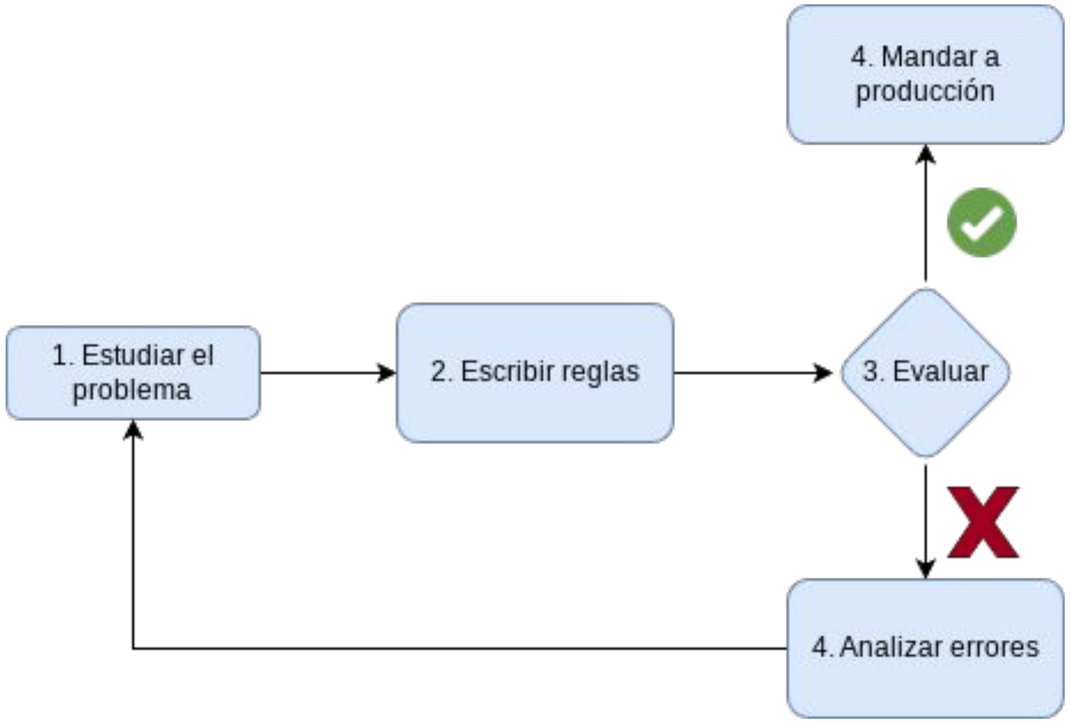
–Arthur Samuel, 1959

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

–Tom Mitchell, 1997

- **Experiencia**: Datos de *entrenamiento*.
- **Tarea**: Es **específica**, es una cosa puntual (clasifica spam, dice si hay perros o gatos en una foto, predice el precio de propiedades, etc.).
- **Medida de rendimiento**: Tenemos una forma de medir "que tan bien hace esa tarea"

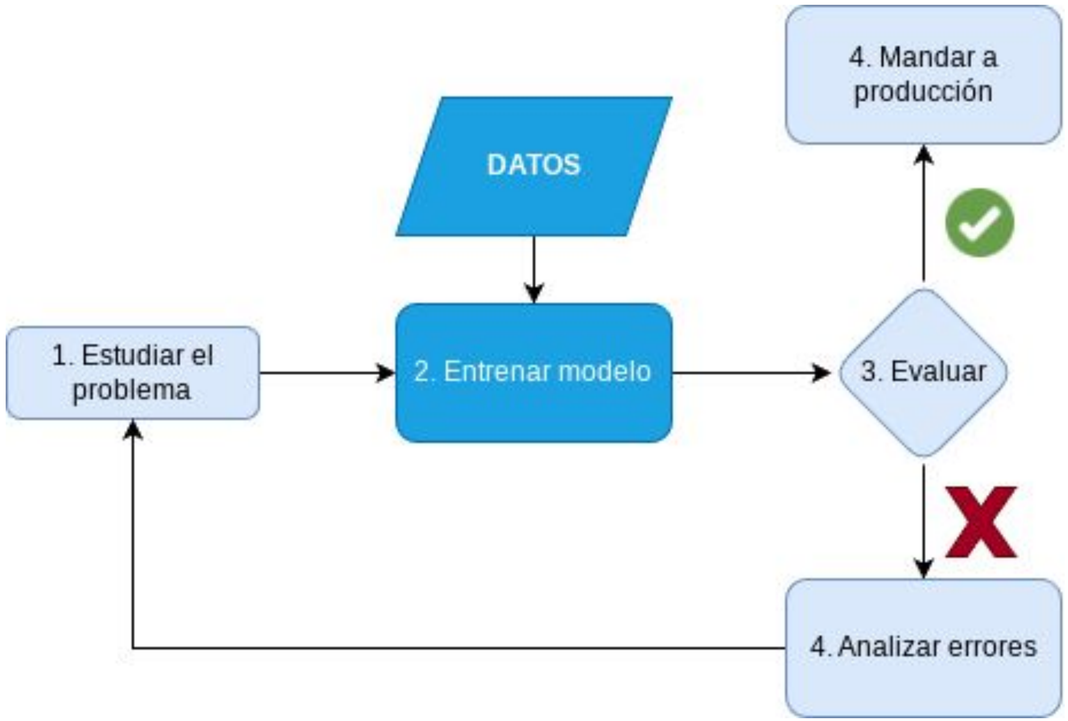
¿Por qué usar aprendizaje automático?



El *enfoque tradicional* para hacer un programa que realice una tarea (en este caso un filtro anti spam).

- Primero, examinar qué aspecto suele tener el spam (palabras o frases sospechosas, remitentes, etc)
- Escribir un algoritmo de detección para cada uno de los patrones que haya observado, el programa marcaría los correos como spam si estos patrones fueran detectados.
- Probar el sistema y repetir 1 y 2 hasta que esté listo para lanzar.

¿Por qué usar aprendizaje automático?



El “Enfoque ML”:

- En vez de pensar uno por uno que cosas son las que definen que un correo sea spam se le dan datos a un modelo y se busca que el aprenda automáticamente qué cosas son buenos predictores de spam. Pueden ser patrones de palabras, frases, remitentes, links u otras características **presentes en los datos**.

¿Ventajas y desventajas? ¿Cuando conviene y cuando no?

Fundamentos del aprendizaje supervisado

- Hay varios tipos de mecanismos de aprendizaje automático, nos vamos a enfocar en dos de las categorías más generales(*):
  - **Aprendizaje supervisado**
  - **Aprendizaje no supervisado**
- Tanto el aprendizaje supervisado como no supervisado ***aprenden de los datos*** *por lo* que para arrancar se necesita un conjunto de datos lo *suficientemente grande*.
- La diferencia es que para usar aprendizaje supervisado estos datos tienen que estar **etiquetados**.

¿Qué significa “aprenden de los datos”?

(*) Hay una tercera categoría llamada “Aprendizaje por refuerzos” en el cual un agente aprende a tomar decisiones mediante la interacción con un entorno para maximizar una recompensa. (Link)

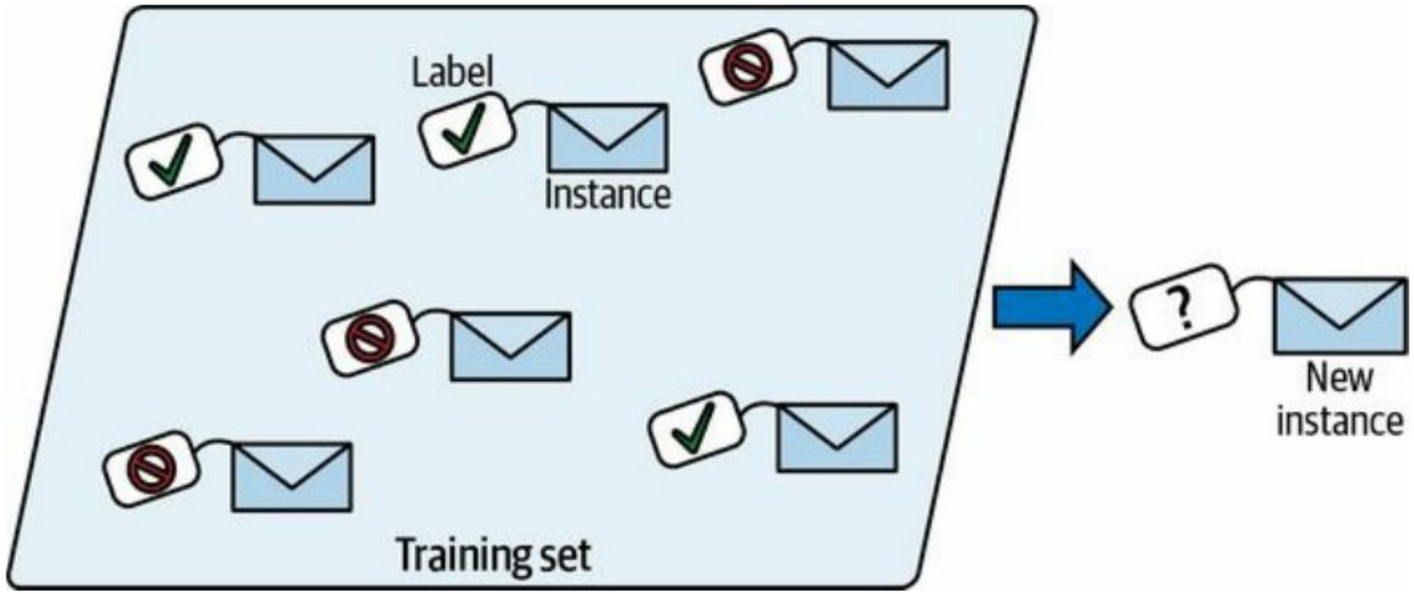
Fundamentos del aprendizaje supervisado

- ¿Qué significa que estén o no etiquetados? Los datos de entrenamiento con los que el modelo aprende **tienen la solución deseada**. Esto es, en el ejemplo del sistema antispam:

  - **Aprendizaje supervisado**: Se tienen muchos correos y tenemos marcados cuales son spam y cuáles no, esta es la etiqueta (lo que queremos que el sistema aprenda).
  - **Aprendizaje no supervisado**: Se tienen muchos correos, pero no hay diferenciación en cuales son spam y cuales no, el modelo tiene que aprender a diferenciarlos aprendiendo sus características (vamos a ver ejemplos de esto cuando lleguemos a *clustering*).

LINK: ¿Qué es el Aprendizaje Supervisado y No Supervisado? | DotCSV

Fundamentos del aprendizaje supervisado



- Resumiendo:
  - Viendo “lo que entra” y lo que “debería salir” el modelo aprende solo las características que definen cada case de correo (spam, no spam).
  - Utilizando este conocimiento general cuando reciba un correo nuevo va a poder decidir a qué clase pertenece por más que no lo haya visto antes.

Imágen del Gerón

Problemas Clásicos de ML: Clasificación y Regresión

- Por el momento sigamos en el aprendizaje supervisado (por lo que tenemos datos y sus etiquetas) y ahora pongamos nombre al **tipo de problema** que resuelven.
- En el ejemplo anterior aprendía a tomar un correo y definia si era spam o no, o sea lo ponía en una de dos clases. **Lo clasificaba -> Problema de clasificación.**
- Otra opción podría ser asignarle una probabilidad de que sea spam, esta probabilidad es un número. Este tipo de problema, donde la salida es un valor continuo, se denomina **de regresión.**
- Algunos problemas de regresión se pueden transformar en clasificación eligiendo el umbral adecuado (“Si tiene una probabilidad mayor a 90% clasificalo como spam”)

Las etiquetas del conjunto de entrenamiento tienen que ser acorde al tipo de problema

¿Cómo es un proyecto de AA?

1. Panorama general del problema (*Domain Knowledge*).
2. Obtención de datos.
3. Explorar, visualizar y analizar los datos (Exploratory Data Analysis).
4. Preparar los datos para los algoritmos de Aprendizaje Automático.
5. Seleccionar y entrenar un modelo.
6. *Fine-tunnear* el modelo.
7. Presentar solución final.
8. *Poner en marcha*, monitorear y mantener el sistema.

¿Cómo es un proyecto de AA?

1. Panorama general del problema (*Domain Knowledge*).
2. Obtención de datos.
3. Explorar, visualizar y analizar los datos (Exploratory Data Analysis).
4. **Preparar los datos para los algoritmos de Aprendizaje Automático.**
5. **Seleccionar y entrenar un modelo.**
6. ***Fine-tunnear* el modelo.**
7. **Presentar solución final.**
8. *Poner en marcha*, monitorear y mantener el sistema.

Enmarcar el problema

- Entender el objetivo que se quiere resolver (Spoiler: hacer un modelo de AA no es el objetivo...)
- ¿Es realmente un problema para resolver con AA?
  - ¿Se quiere optimizar alguna métrica?, ¿encontrar anomalias?, ¿predecir un resultado?.
  - ¿Se tienen datos?, ¿cuantos?, ¿son representativos?.
  - ¿Hay resultados previos?.
- Si definimos que alguna parte de lo que se quiere resolver se pueda atacar con algún mecanismo de aprendizaje automático tenemos que definir el tipo de problema:
  - ¿Se busca poder predecir un número? -> problema de Regresión.
  - ¿Se busca poner el dato en alguna clase? -> problema de Clasificación.
- Tenemos el problema y los datos
  - Si en los datos tenemos la respuesta explícita podemos usar aprendizaje supervisado sino es aprendizaje no supervisado.

Modelado

- Quedémonos por ahora en el aprendizaje supervisado, o sea que tenemos datos etiquetados, pero primero veamos (¿repasemos?) rápidamente el concepto más general del modelado.
- “Un modelo es una representación **simplificada(*)** de un sistema, fenómeno o proceso real para entender, analizar o predecir su comportamiento”

(*) si, simplificada, sino no es útil. Ya lo dijo Jorge...

LINK: Modelos para entender una realidad caótica | DotCSV

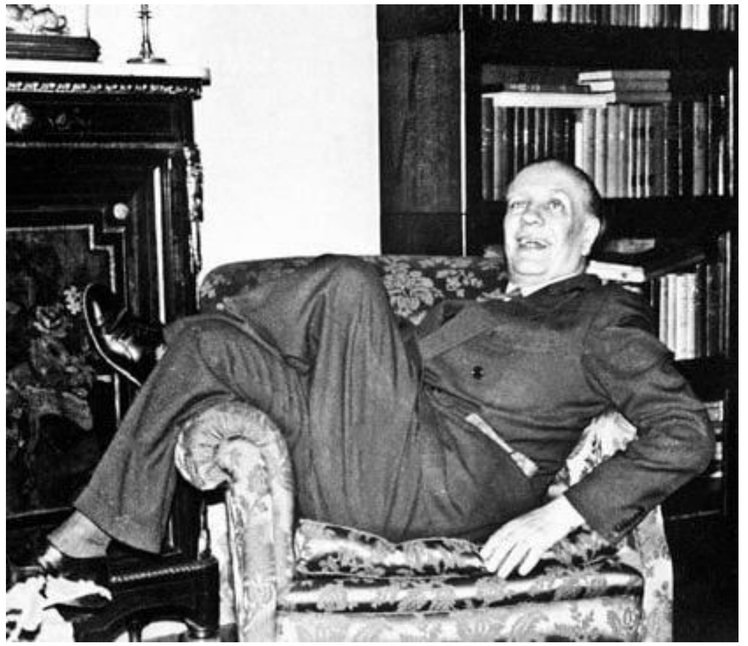
Modelado

Del rigor de la ciencia:

"En aquel Imperio, el Arte de la Cartografía logró tal Perfección que el mapa de una sola Provincia ocupaba toda una Ciudad, y el mapa del Imperio, toda una Provincia. Con el tiempo, estos Mapas Desmesurados no satisficieron y los Colegios de Cartógrafos levantaron un Mapa del Imperio, que tenía el tamaño del Imperio y coincidía puntualmente con él.

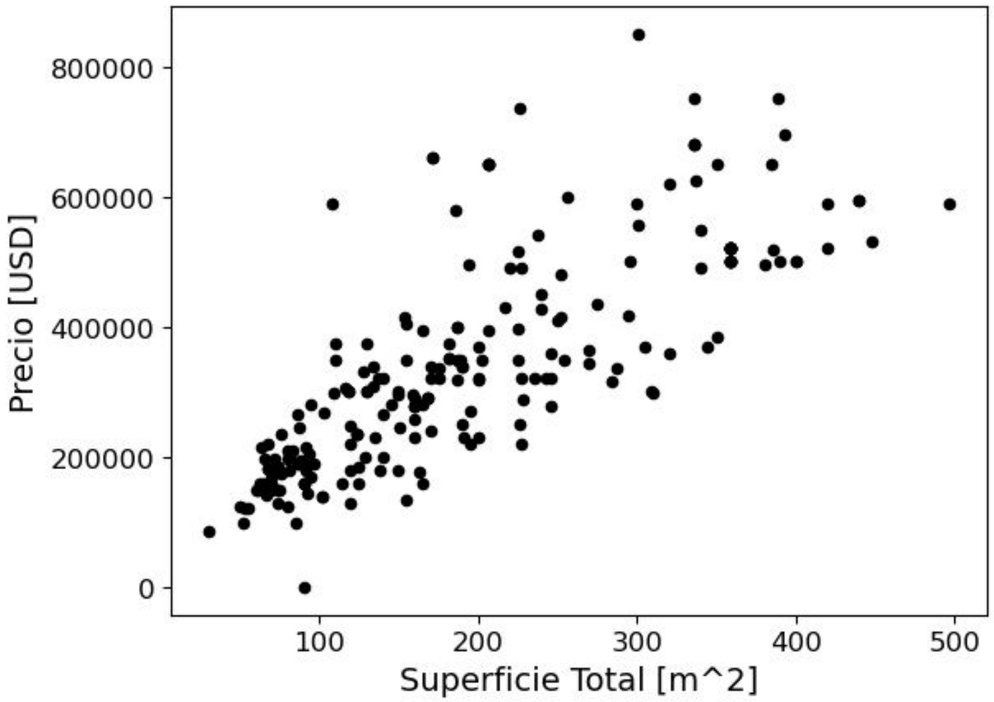
Menos Adictas al Estudio de la Cartografía, las Generaciones Siguientes entendieron que ese dilatado Mapa era Inútil y no sin Impiedad lo entregaron a las Inclemencias del Sol y los Inviernos. En los desiertos del Oeste perduran despedazadas Ruinas del Mapa, habitadas por Animales y por Mendigos; en todo el País no hay otra reliquia de las Disciplinas Geográficas."

Suárez Miranda, Viajes de Varones Prudentes, Libro Cuarto, Cap. XLV, Lérida, 1658. (Jorge Luis Borges)



Modelo de regresión

- Un modelo de regresión es aquel en el que se busca describir el comportamiento de una variable (o un conjunto de variables) contínua a partir de otras variables del dataset.
- La variable que se busca describir (o predecir) se suele llamar variable objetivo (target). Las variables a partir de las cuales se busca hacer esto se llaman covariables, o variables predictora, o variables independientes, regresores, features (características), etc.

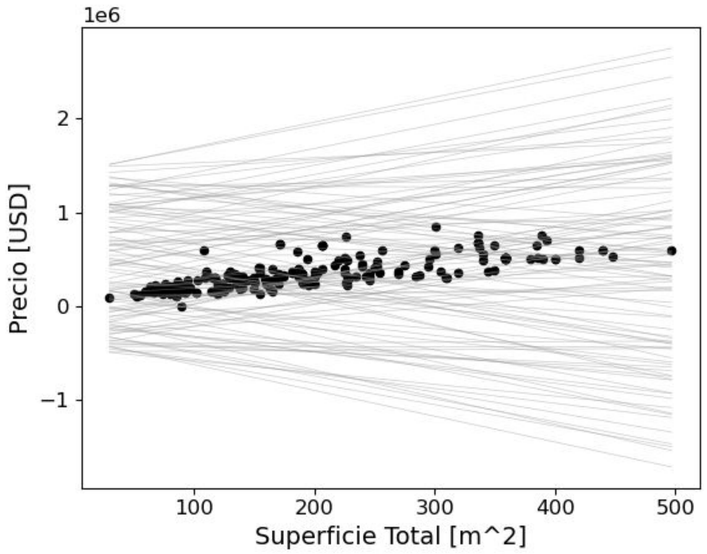


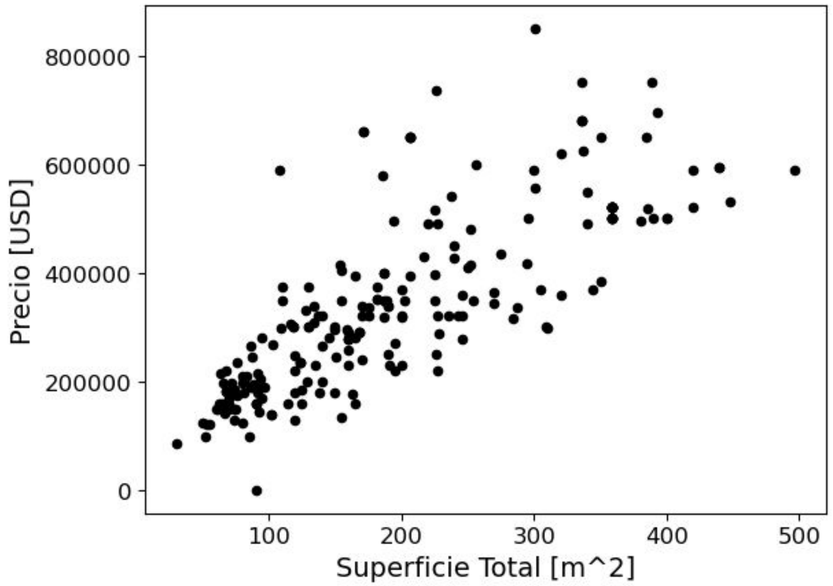
Por ejemplo queremos describir el precio en función de la superficie total de las casas de Villa Urquiza.

Regresión lineal simple

- El modelo de regresión lineal más sencillo relaciona una variable *target* (en este caso **precio**) con la covariable (en este caso la **superficie total**) utilizando esta fórmula:

$$\text{precio [USD]} = \omega_0 + \omega_1 \cdot \text{sup. total}$$



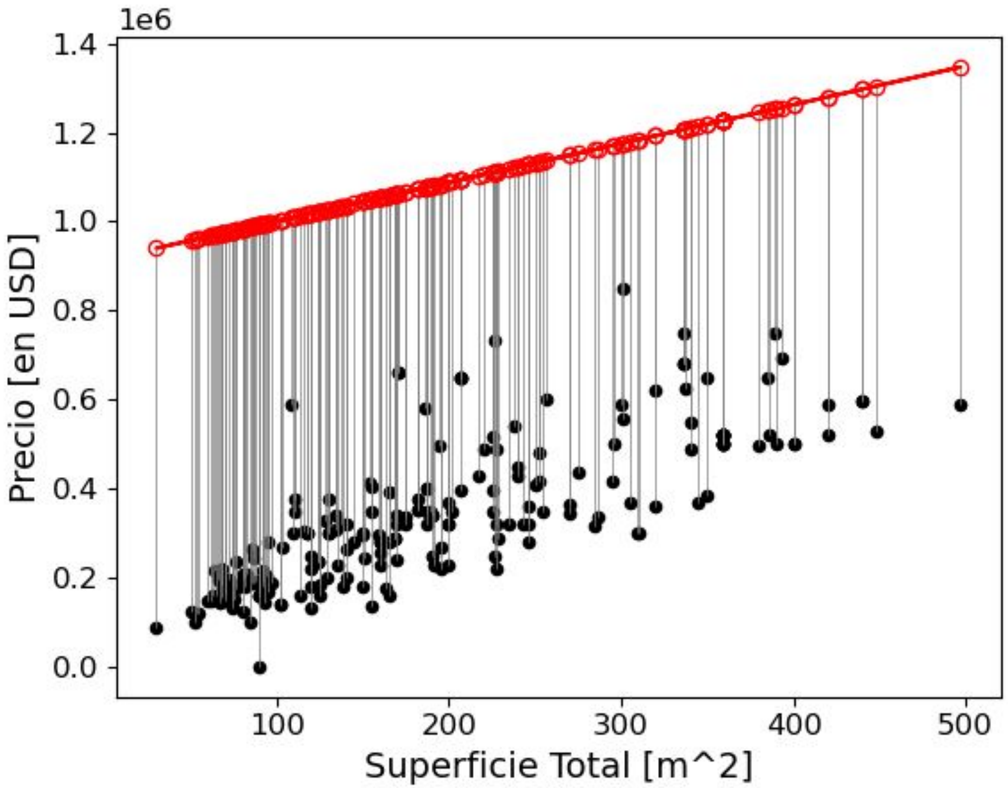


- *w0* y *w1* son los **parámetros** del modelo.
- Esta fórmula define una **familia de modelos**. Todos los modelos de esta familia se ven como rectas, pero hay pocas que tienen similitud con los datos.

LINK: Regresión Lineal y Mínimos Cuadrados Ordinarios | DotCSV

Regresión lineal simple

- Para encontrar “la mejor recta” hay que encontrar el valor adecuado de los parámetros.
- Para cualquier recta podemos ver la distancia entre los puntos y la recta como segmentos, a estas distancias se las llama **residuos.**
- Como pueden ser positivos o negativos vamos a querer que los parámetros sean aquellos nos den los valores absolutos más chicos.
- Pero como queremos que esto valga para **todos** los puntos, lo que vamos a querer es asegurarnos que **el promedio de los valores absolutos de todos los residuos sea mínimo**.



Regresión lineal simple

- Esto nos da nuestra primera **métrica** de regresión, el error absoluto medio **(MAE):**

$$\mathrm{MAE} = \frac{1}{N}(|r_1| + |r_2| + \cdots + |r_N|) = \frac{1}{N}\sum_{i=1}^{N}|r_i|$$

- Otra métrica posible es el error cuadrático medio **(MSE):**

$$\mathrm{MSE} = \frac{1}{N}(r_1^2 + r_2^2 + \cdots + r_N^2) = \frac{1}{N}\sum_{i=1}^{N}(r_i)^2$$

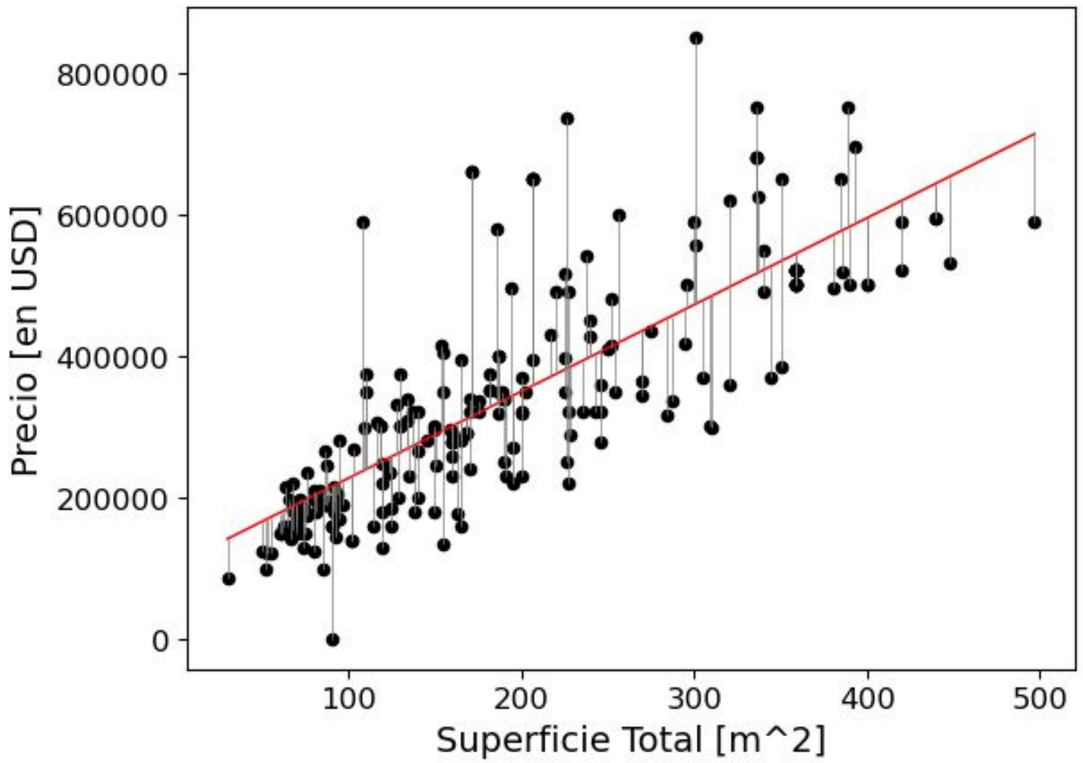
- O la raíz del error cuadrático medio **(RMSE):**

$$\mathrm{RMSE} = \sqrt{\frac{1}{N}(r_1^2 + r_2^2 + \cdots + r_N^2)} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(r_i)^2}$$

- El coeficiente de determinación (**R^2**):

$$R^2 = \frac{SC_{\mathrm{tot}} - SC_{\mathrm{res}}}{SC_{\mathrm{tot}}}$$

Regresión lineal simple



- Hay un método que nos garantiza encontrar los valores de *w* que minimizan el MSE (busquen “ecuaciones normales”) y encontrar la famosa “recta que mejor ajusta a los datos” con estos valores de sus métricas:
  - **MAE**: 72.92
  - **MSE**: 10509998967.09
  - **RMSE**: 102.52

$$\text{precio [USD]} = \omega_0 + \omega_1 \cdot \text{sup. total}$$

$$\text{precio [USD]} = 104445.604 + 1225.266 \cdot \text{sup. total}$$

Interpretemos w0 y w1 en contexto de las cosas que queríamos responder

Regresión lineal múltiple

- Podemos usar más variables para predecir un *target*, en este caso estamos ante un modelo de **Regresión Lineal Múltiple.**
- Vamos a tener un nuevo *w* a estimar para cada nueva variable que agreguemos, por ejemplo:

$$\text{precio [USD]} = \omega_0 + \omega_1 \cdot \text{sup. total} + \omega_2 \cdot \text{baños}$$

- Ahora se ajusta un *hiperplano* en vez de una línea.
- Al incluir variables relevantes podemos capturar mejor los factores que afectan al precio.
- Cuidado con la interpretación:
  - *w1: Sigue indicando cuánto cambia el precio por metro cuadrado adicional, pero ahora **manteniendo constante el número de baños**.*
  - *w2: Representa cuánto aumenta el precio por cada baño adicional, **manteniendo constante la superficie total**.*

Regresión polinomial

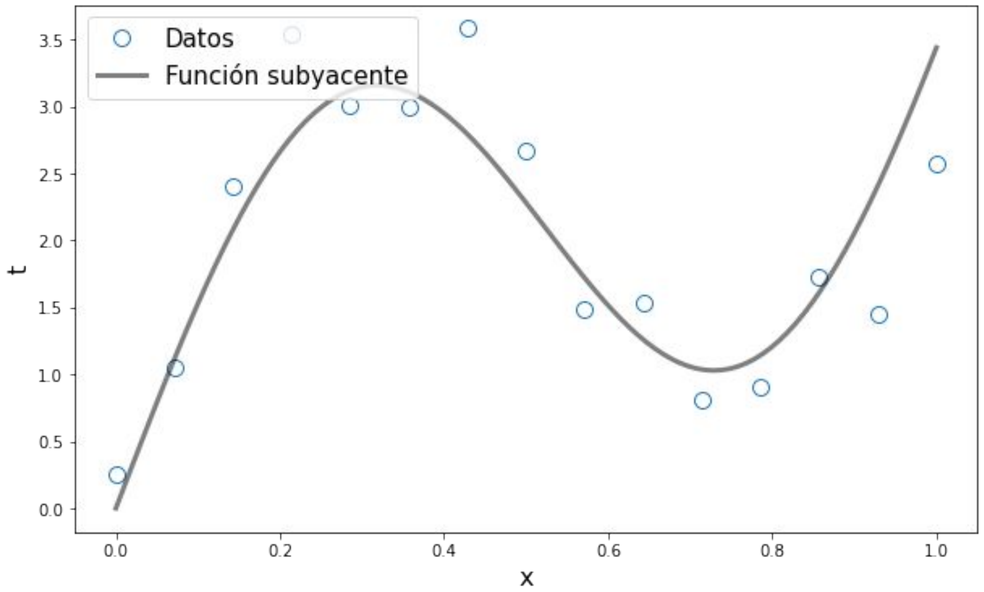
- A veces (muchas...) los datos no siguen una tendencia lineal, podemos usar una extensión de la regresión lineal donde la relación entre el *target* y la variable independiente se modela como un polinomio:

$$\text{precio [USD]} = \omega_0 + \omega_1 \cdot \text{sup. total} + \omega_2 \cdot (\text{sup. total})^2 + \omega_3 \cdot (\text{sup. total})^3 \ldots$$

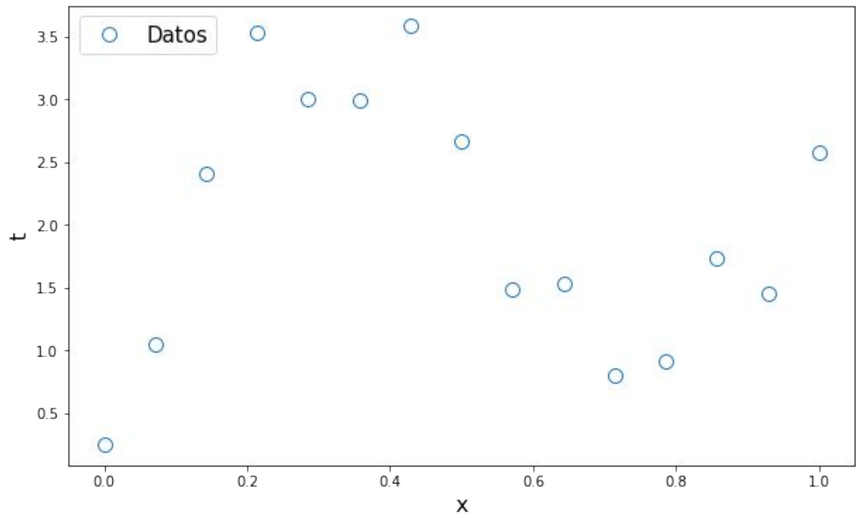
- El grado de este polinomio lo definimos nosotros, es un **hiperparámetro** del modelo.
- Para buscar cual es el grado que mejor ajusta hay que probar y evaluar las métricas, pero hay que tener cuidado con el ***overfitting(*)*** (sobreajuste).
- Veamos a que nos referimos usando datos simulados y aprovechamos para introducir el concepto de **función subyacente**.

(*) Esto nos va a perseguir a lo largo de todo el curso, y la vida...

Regresión polinomial



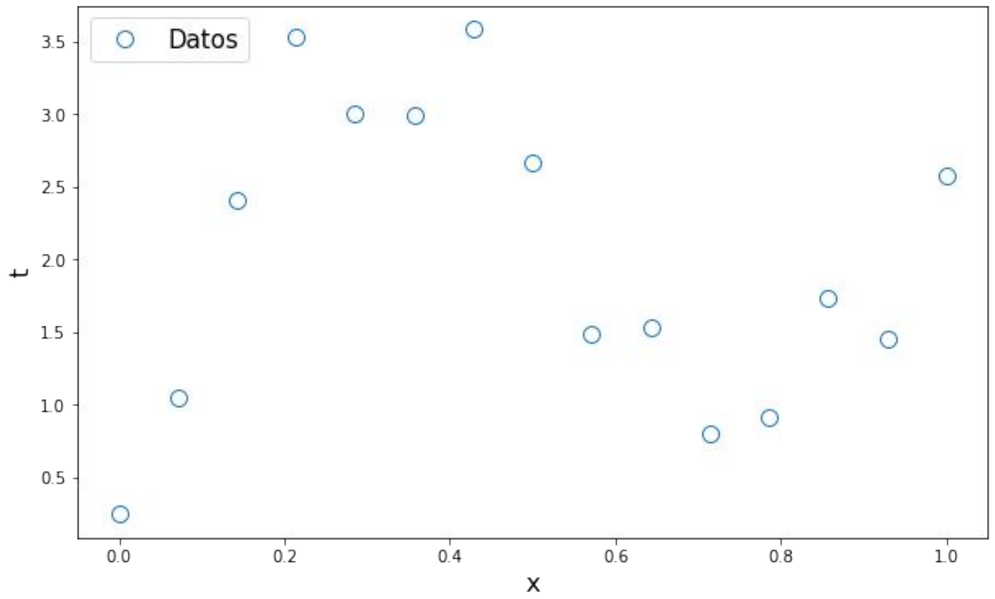
- La función subyacente es la **verdadera** relación entre las variables independientes y el *target*.
- Generalmente(*) la desconocemos y los datos que tenemos tienen errores provenientes de ruidos de diferente naturaleza.
- Es esta la función que siempre queremos aproximar.



Probemos ajustar estos datos con polinomios de diferentes grados

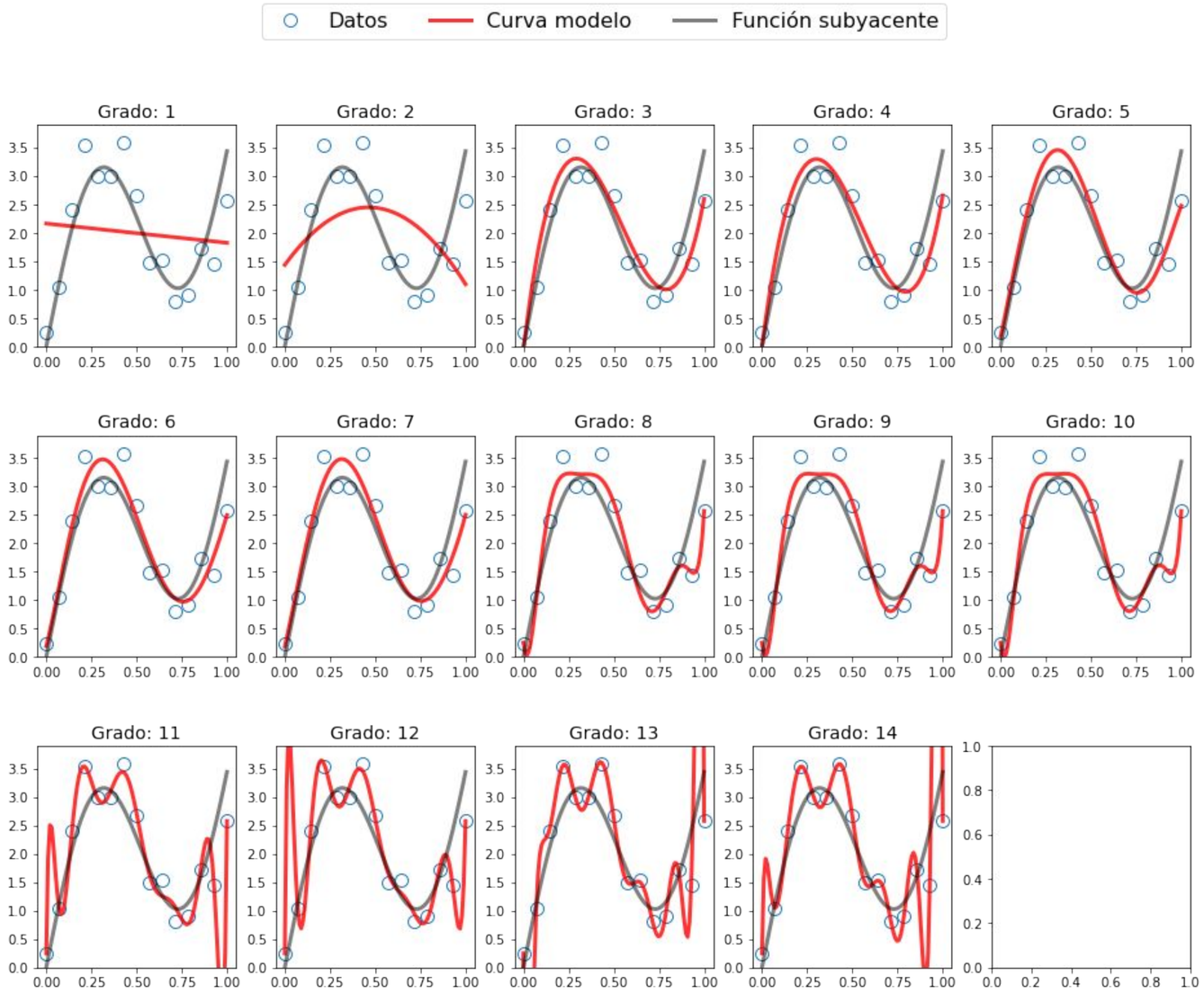
(*) Por no decir siempre...

Overfitting




```
Grado del polinomio: 1
El MSE es: 1.01
Grado del polinomio: 2
El MSE es: 0.85
Grado del polinomio: 3
El MSE es: 0.13
Grado del polinomio: 4
El MSE es: 0.13
Grado del polinomio: 5
El MSE es: 0.1
Grado del polinomio: 6
El MSE es: 0.1
Grado del polinomio: 7
El MSE es: 0.1
Grado del polinomio: 8
El MSE es: 0.06
Grado del polinomio: 9
El MSE es: 0.06
Grado del polinomio: 10
El MSE es: 0.06
Grado del polinomio: 11
El MSE es: 0.02
Grado del polinomio: 12
El MSE es: 0.01
Grado del polinomio: 13
El MSE es: 0.0
Grado del polinomio: 14
El MSE es: 0.0
```

Overfitting



- ¿Conviene aumentar el grado del polinomio?.
- ¿Qué pasaría si tuviéramos menos puntos?.
- ¿Qué pasaría si quiero usar este modelo para predecir sobre datos nuevos?
- **¿Estamos realmente aprendiendo algo sobre la función subyacente?**

Overfitting

- El *overfitting* ocurre cuando un modelo tiene tanta *flexibilidad* que se ajusta *demasiado* a los datos, se aprende el ruido en vez de encontrar la función subyacente. No aprendió a generalizar.
- Las métricas por si solas no nos van a ayudar a detectarlo, lo que hay que hacer es separar aleatoriamente el conjunto de datos en **entrenamiento** y **testeo(*):**
  - **Entrenamiento:** Como su nombre lo indica, es el que usamos para ajustar el modelo.
  - **Testeo:** Una vez que tenemos el modelo ajustado evaluamos su rendimiento en este conjunto.
- Si el modelo está *overfitteando* debería verse en las métricas del conjunto de testeo.

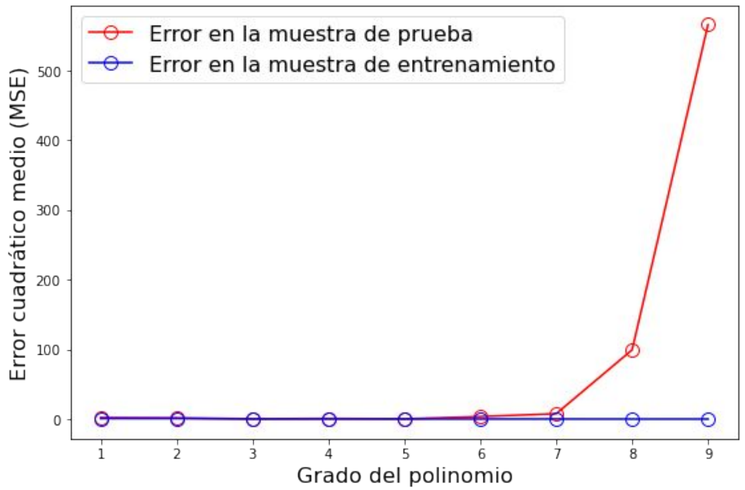
"Había aprendido sin esfuerzo el inglés, el francés, el portugués, el latín. Sospecho, sin embargo, que no era muy capaz de pensar. Pensar es olvidar diferencias, es generalizar, abstraer. En el abarrotado mundo de Funes no había sino detalles, casi inmediatos."

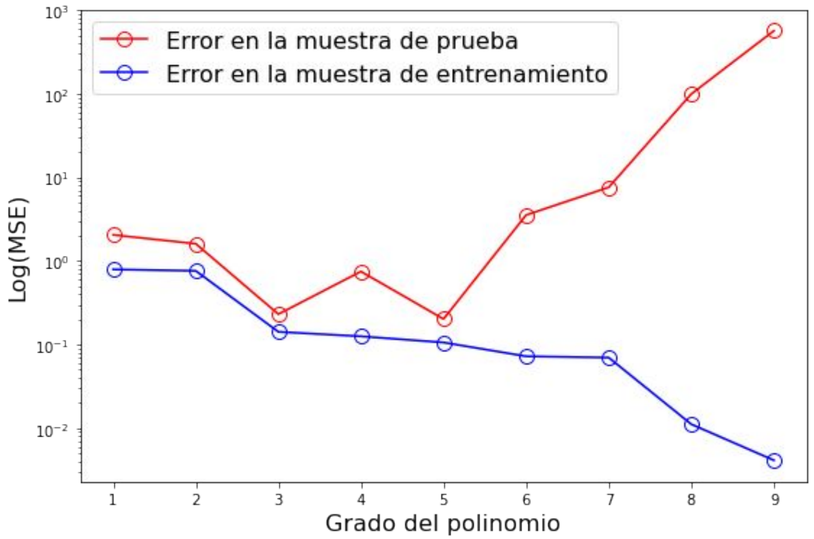
Jorge Luis Borges, Ficciones (1942)

(*) Más adelante vamos a incluir un tercer conjunto llamado de **validación**

Test / Train split

```
Grado del polinomio: 1
El MSE en el entrenamiento es: 0.79
El MSE en el testeo es: 2.05
Grado del polinomio: 2
El MSE en el entrenamiento es: 0.76
El MSE en el testeo es: 1.6
Grado del polinomio: 3
El MSE en el entrenamiento es: 0.14
El MSE en el testeo es: 0.23
Grado del polinomio: 4
El MSE en el entrenamiento es: 0.13
El MSE en el testeo es: 0.75
Grado del polinomio: 5
El MSE en el entrenamiento es: 0.11
El MSE en el testeo es: 0.2
Grado del polinomio: 6
El MSE en el entrenamiento es: 0.07
El MSE en el testeo es: 3.54
Grado del polinomio: 7
El MSE en el entrenamiento es: 0.07
El MSE en el testeo es: 7.59
Grado del polinomio: 8
El MSE en el entrenamiento es: 0.01
El MSE en el testeo es: 99.28
Grado del polinomio: 9
El MSE en el entrenamiento es: 0.0
El MSE en el testeo es: 566.14
```





Regularización

- Una forma de reducir el sobreajuste es regularizar el modelo, restringirlo, en el caso del modelo polinomial una manera sencilla es la de reducir el grado.
- Pero también se pueden restringir los pesos del modelo introduciendo una penalización para que el modelo priorice los valores de los coeficientes que sean pequeños y, por lo tanto, menos complejos.
- **Regularización Ridge (L2):** Agrega un término de penalización que “castiga” a los coeficientes grandes en el cálculo del MSE de la función de pérdida:

$$E_{\text{ridge}}(\boldsymbol{\omega}; \lambda) = \frac{1}{2} \sum_{i=1}^{N} \{y(x_i, \boldsymbol{\omega}) - t_i\}^2 + \frac{\lambda}{2} \sum_{i=1}^{M} \omega_i^2$$

- Aparece un nuevo hiperparámetro, λ.
- Si λ = 0 es una regresión lineal común, si es muy grande todos los pesos terminan muy cerca de 0.

Regularización

- **Regularización Lasso (L1):** También agrega un término de penalización pero en este caso es la norma L1, que es simplemente la suma de los valores absolutos de los parámetros del modelo.
- Tiende a hacer que algunos parámetros se reduzcan a cero pudiendo servir además para selección de características

$$E_{\text{lasso}}(\boldsymbol{\omega};\lambda) = \frac{1}{2}\sum_{i=1}^{N}\{y(x_i,\boldsymbol{\omega}) - t_i\}^2 + \frac{\lambda}{2}\sum_{i=1}^{M}|\omega_i|$$

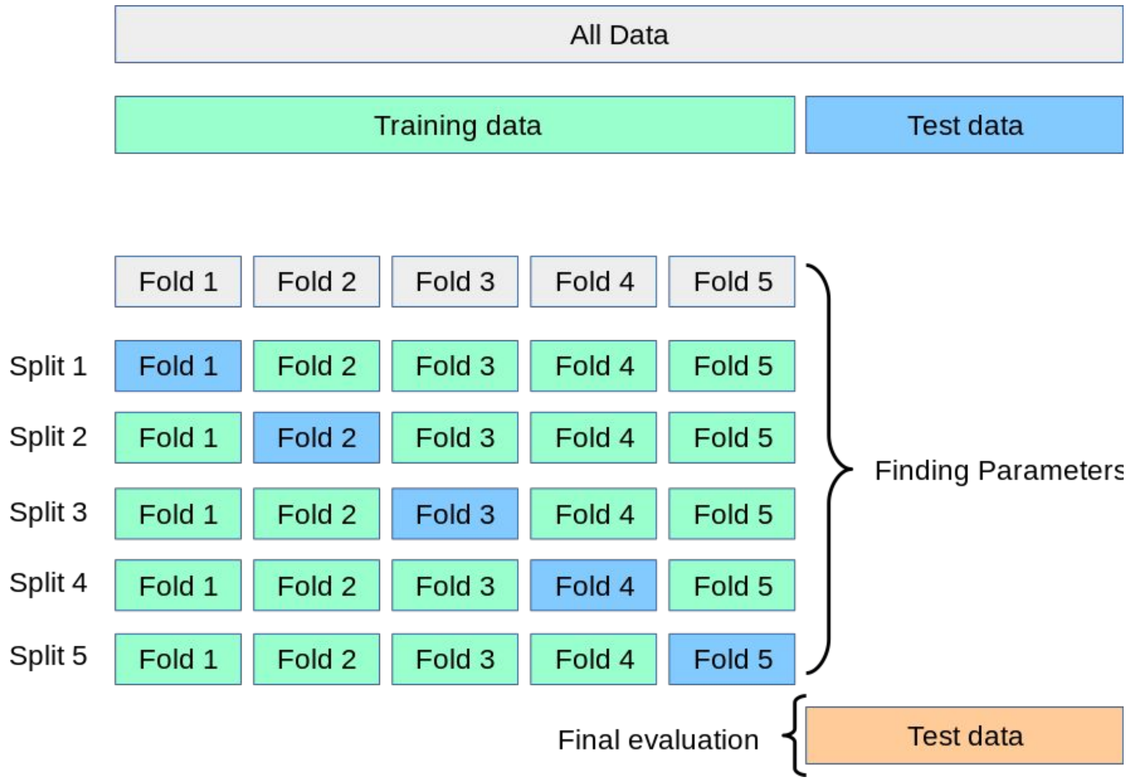
- Notemos que en ambos casos tenemos un nuevo hiperparámetro que tenemos que definir.
- Esto se puede hacer a mano (probando varios valores de grados y varios λ) o automatizarlo con *GridSearchCV* o *RandomizedSearchCV (ver notebook para detalles)*

Validación

- Para poder objetivamente asegurarnos la performance de nuestro modelo en datos nuevos nunca vistos debemos limitarnos y no mirar el conjunto de test. De otra manera sobreajustaremos y no nos servirá como estimador de la performance real.
- Por otro lado para buscar mejores hiperparámetros debemos hacerlo sobre puntos fuera del entrenamiento. Una solución sería mantener un conjunto para entrenar otro para validar hiperparámetros. Y otro más para tener un valor no sesgado de la performance de nuestro modelo, este nuevo conjunto se llama de **Validación.**

- Conjunto de Entrenamiento: Donde ajustamos el modelo.
- Conjunto de Validación: Donde evaluamos el modelo y lo usamos para escoger mejores hiperparámetros/características.
- Conjunto de Prueba: Lo usamos solamente al final del desarrollo para establecer la performance esperada del modelo en datos nuevos.

Validación Cruzada



- Otro mecanismo es el de validación cruzada (este en particular es *K-Fold Cross-validation*).
- En vez de separar en entrenamiento en dos conjuntos fijos se separan en varios de forma que se entrene en algunos pero se **valide** en otros.
- De esta forma se busca tener una mejor idea de como se comporta nuestro modelo (es una evaluación más **robusta**).

Fuente (Leer)

Parada técnica: Resumen hasta ahora

- **Modelos de aprendizaje automático**: Buscan patrones útiles en los datos.
- **Parámetros:** Valores que los modelos ajustan al entrenar.
- **Hiperparámetros:** Valores definidos por nosotros al instanciar el modelo.
- **Aprendizaje supervisado/no supervisado**: Los datos tienen etiquetas o no, o sea, tienen el resultado al que vamos a querer llegar.
- **Tipo de problema:** Clasificación o regresión.
- **Métricas de regresión:** MAE, MSE, RMSE, R2.
- **Subajuste (*underfitting*)**: El modelo es demasiado simple para capturar los patrones en los datos.
- **Sobreajuste (*overfitting*)**: El modelo es demasiado flexible y aprendió el ruido en vez de aprender a generalizar y encontrar la función subyacente (se transformó en Funes...).
- **Train, Validation, Test**: Separamos los datos en varios conjuntos, para ajuste de parámetros, ajuste de hiperparámetros, detección de sobreajuste y estimar como funcionará nuestro modelo con datos nuevos.

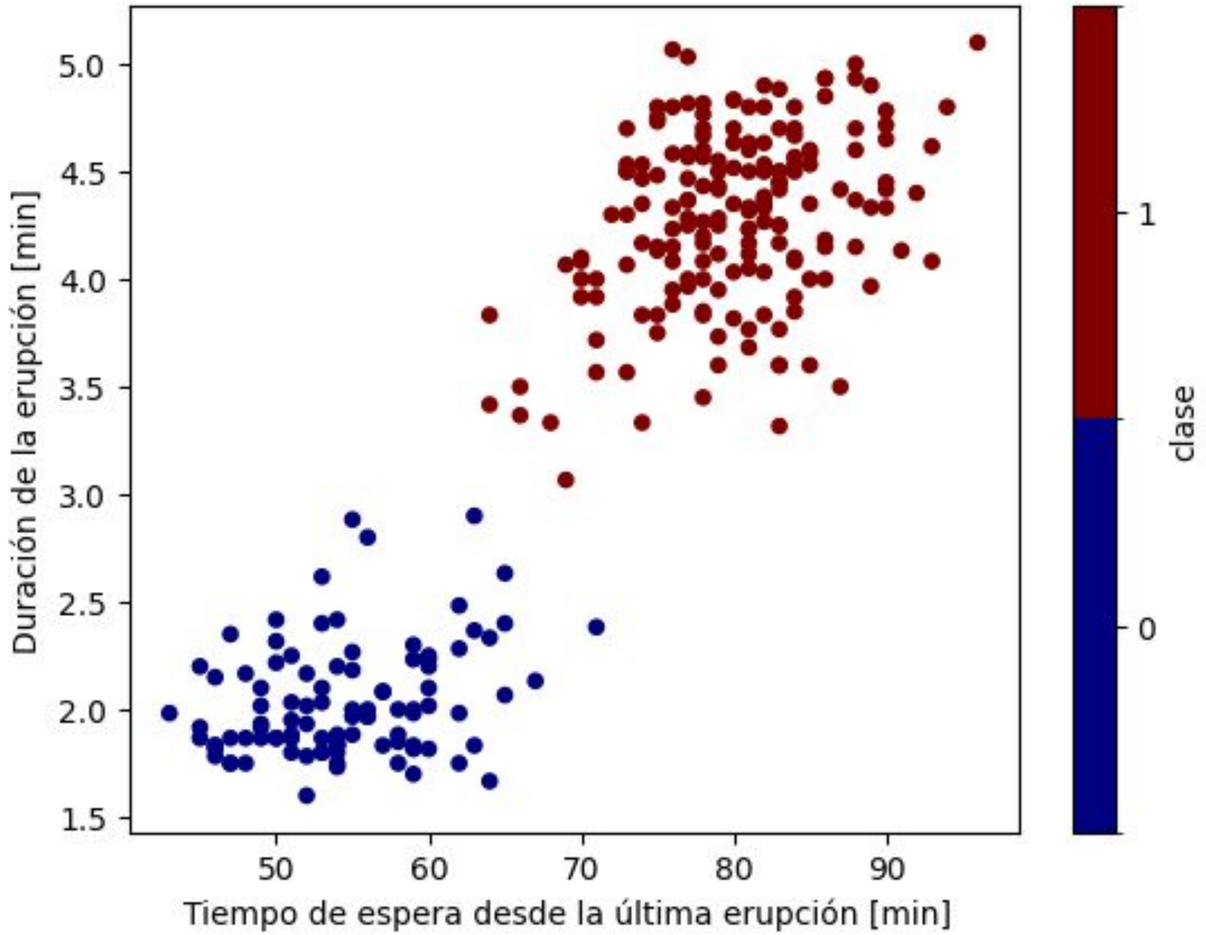
Problemas de Clasificación

- En los problemas de clasificación la variable a modelar es una *categoría* o clase.
- Vamos a quedarnos por ahora con el problema más simple de este tipo que es la **clasificación binaria** (cada dato pertenece a una de dos clases).
- Lo primero que tenemos que hacer es **codificar** las clases, ¿qué sería esto? Bueno, las clases suelen ser categorías como “perro” o “gato”, o en el ejemplo de properati “casa” o departamento, hay que convertirlas en números.
- Puede ser 0 para las casa y 1 para los departamentos (o -1 y 1), de esta manera nuestro *target* ya es numérico (que es algo que ya suena más a lo que veníamos haciendo).
- Veamos lo que queremos hacer gráficamente con un ejemplo simple.

Problemas de Clasificación

- Old Faithful es un geiser del parque Yellowstone que exhibe una notable regularidad entre erupciones. Las erupciones pueden largar entre 14,000 y 32,000 litros de agua hirviendo a una altura de más de 50 metros.
- La duración de las erupciones muestra una distribución bimodal, es decir, que hay erupciones "largas" y "cortas". Veamos los datos de duración en relación con el intervalo entre erupciones (tiempo de espera de cada erupción).
- Vamos a suponer que las erupciones de Old Faithful pueden separarse en dos clases, según el grado de salinidad de las erupciones.
- Grafiquemos duración en función del tiempo espera entre las erupciones, y agreguemos una variable según si la erupción fue clasificada como de "alta salinidad" (clase 1) o "baja salinidad" (clase 0).

Problemas de Clasificación



- Vemos fácilmente que las erupciones de salinidad diferente se separan en este plano.
- Podemos entonces pensar en un modelo que a partir del tiempo de espera y la duración de la erupción clasifique a los eventos de Old Faithful y evitarnos tener que medir la salinidad.
- Acá entra el **perceptrón simple.**

El Perceptrón

- El perceptrón es una de las arquitecturas más sencillas, inventada en 1957 por Frank Rosenblatt. Se basa en una neurona artificial (SPOILER) llamada unidad lógica umbral (TLU).
- Las entradas y salidas son números, y cada conexión de entrada está asociada a un **peso**.
- La TLU calcula primero una función lineal de sus entradas y luego aplica una **función escalón** al resultado.
- Una única TLU puede utilizarse para una **clasificación binaria lineal simple**. Calcula una función lineal de sus entradas y, si el resultado supera un umbral, emite la clase positiva. En caso contrario, selecciona la clase negativa.

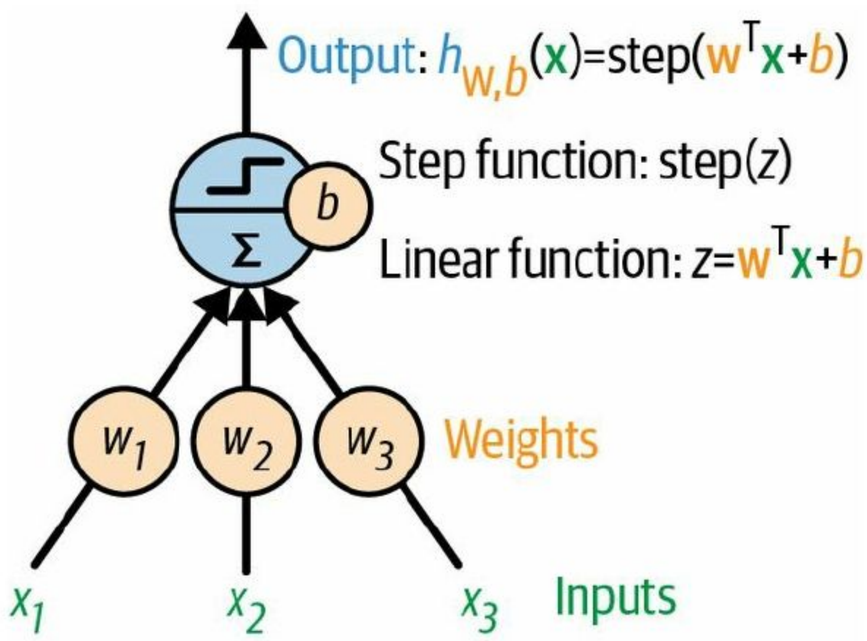
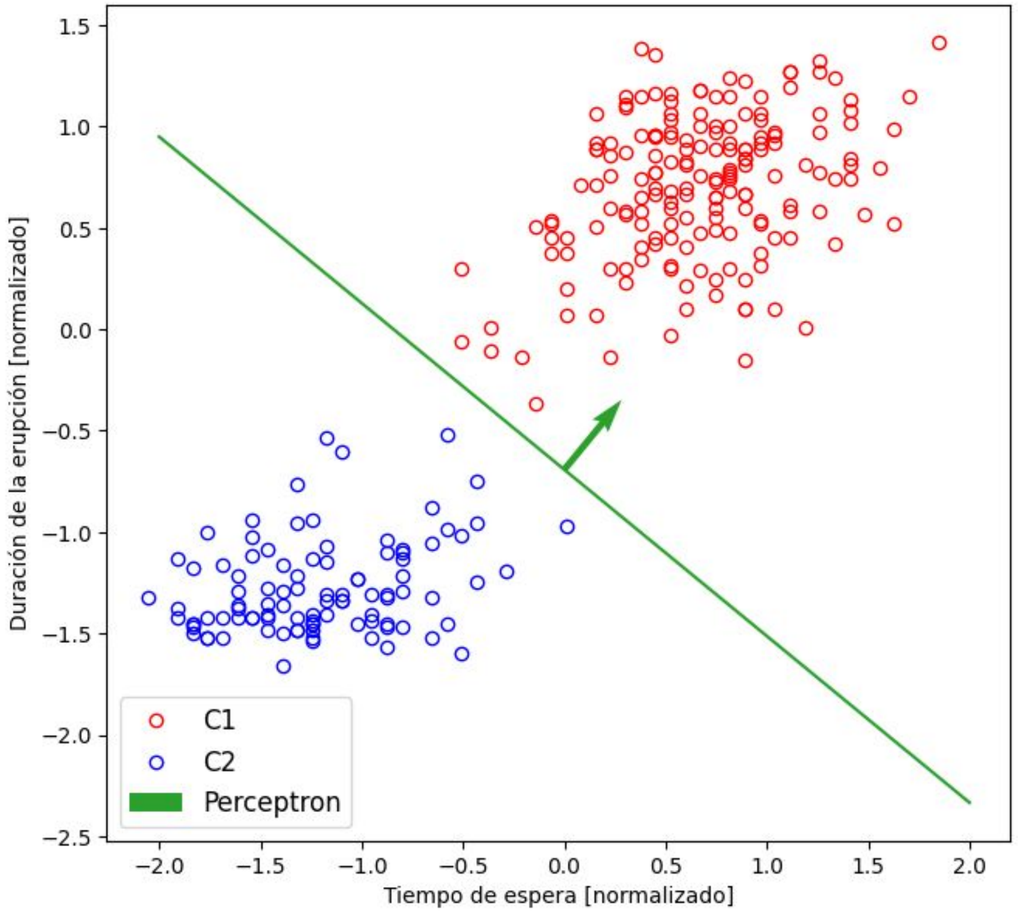


Imagen del Gerón (como la gran mayoría)

El Perceptrón

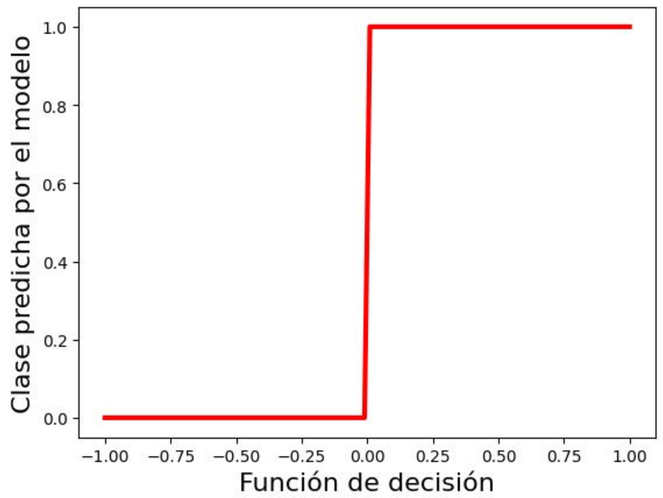
- Vean el notebook para más detalles, pero lo que va a hacer el perceptrón es encontrar la frontera de decisión que separa los datos y nos va a devolver 0 o 1 para cada dato (viejo o nuevo).
- Si los datos son **linealmente separables**(*) el perceptrón nos garantiza encontrar esta frontera.

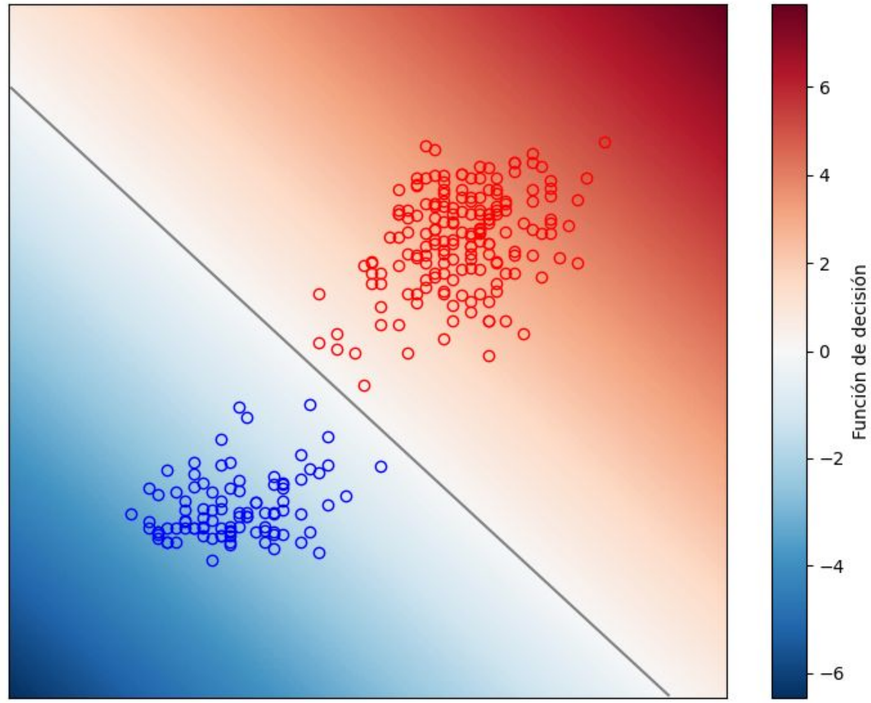


(*) Un poco ya vimos que hacer si no se cumple esto.

El Perceptrón

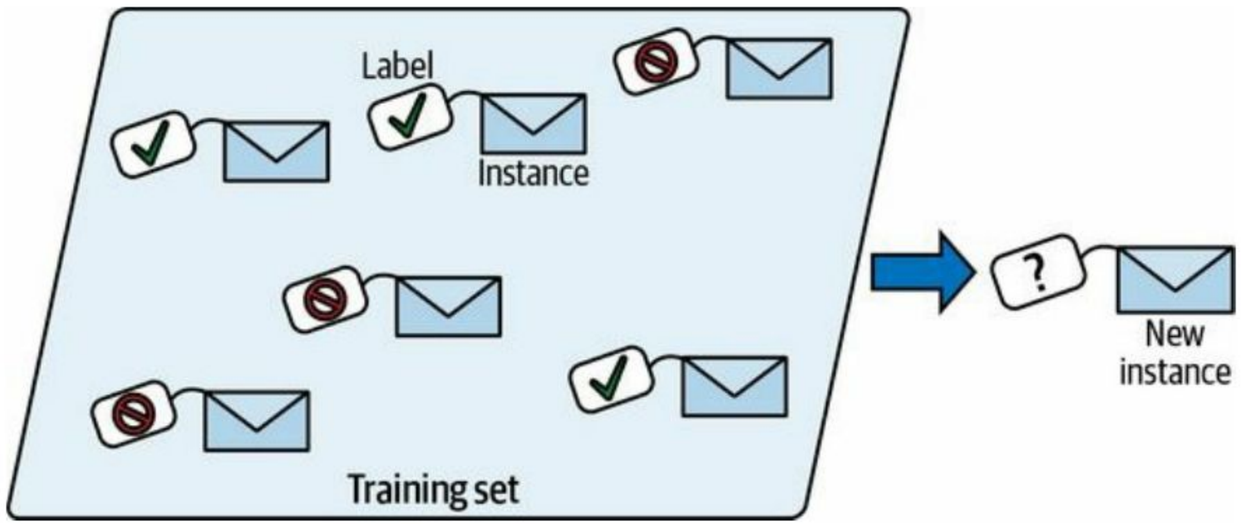
- Es importante remarcar que lo que define la salida binaria del perceptrón es la función de decisión escalón. Lo que le llega es una medida de la distancia a la frontera.





- Más adelante vamos a ver que hacer cuando se tiene más de una clase, pero quedémonos con esta idea de clasificación binaria para hablar de como medimos que tan bien funciona un modelo de clasificación.

Evaluación y métricas



- Recuperemos este ejemplo del clasificador de Spam del comienzo.
- Cada dato son características de cada correo (usar la imaginación) junto a una etiqueta de “Spam” y “no spam”.
- Hacemos un poco de *ingeniería de features* y elegimos características útiles, codificamos la clase, ajustamos un perceptrón en *train* validamos como funciona en *test*, etc.
- Pero, ¿qué medimos?, ¿qué valor reportamos?, ¿cómo comparamos estos resultados con otro método?.

Evaluación y métricas

- En regresión es facil, hay una medida de distancia entre lo que dió y lo que debería haber dado.
- La cosa se complica cuando pensamos en problemas de clasificación, hay dos tipos de aciertos:
  - **Verdadero positivo (TP)**: Decir que un correo es spam cuando **sí** lo es.
  - **Verdadero negativo (TN)**: Decir que un correo no es spam cuando **no** lo es.
- Y dos tipos de equivocaciones que se pueden cometer:
  - **Falso positivo (FP)**: Decir que un correo es spam cuando **no** lo es.
  - **Falso negativo (FN)**: Decir que un correo no es spam cuando **sí** lo es.

- Importante: La definición de estos errores o aciertos cambian dependiendo de lo que se considere la “clase positiva”, que es la que se considera verdadera en el contexto de problema:
  - Si “el correo es spam” es la clase positiva, la definición de TP es como la de arriba.
  - En cambio si el sistema está hecho para “detectar el correo que NO sea spam” un TP sería “Decir que el correo NO es spam cuando NO lo era.

Evaluación y métricas

- En este caso particular ¿qué es mejor?
  - ¿Qué el sistema sea muy estricto y detecte todos los correos basura a riesgo de que cada tanto etiquete incorrectamente un correo importante? -> baja tasa de **falsos negativos**.
  - ¿Qué sea un poco más laxo y se le escape algún spam, pero que no me haga perder correos de mi jefe? -> baja tasa de **falsos positivos**.

¿Les parece que ambos errores son igual de importantes?
¿Qué pasa si es otro problema? Por ejemplo clasificación de imágenes médicas...

Evaluación y métricas

- Una forma de ver esto con una **matriz de confusión.**
- En las filas ponemos los valores que **predijo** el modelo.
- En las columnas ponemos los resultados que **tendría** que haber dado.

Fuente (revisar para mayor detalle)

Evaluación y métricas

- **Exactitud (*accuracy*)** : Es la proporción de predicciones correctas entre **el total de predicciones realizadas**:

$$\text{Exactitud} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

"De todos los que clasifiqué como "spam" y "no spam" ¿cuantos lo eran efectivamente?"

- **Precisión** : Es la proporción de verdaderos positivos (TP) entre **todas las predicciones** positivas:

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

"De todos los que dije que eran spam, ¿cuántos eran spam efectivamente?"

Evaluación y métricas

- **Exhaustividad/Sensiblidad o *Recall*** : Es la proporción de verdaderos positivos (TP) entre **todas las instancias** que realmente son positivas:

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

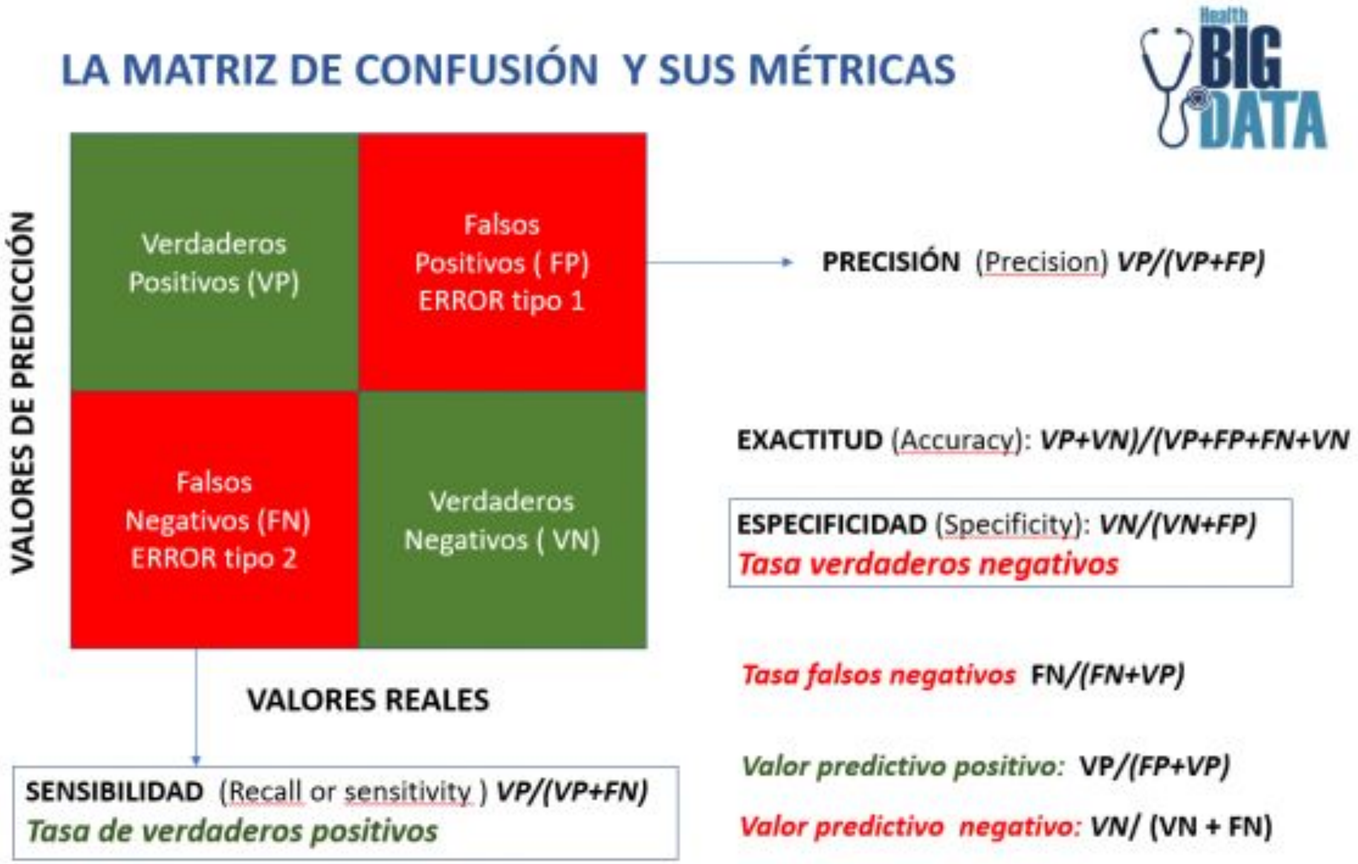
"De todos los spam que hay, ¿cuantos encontré?"

- **F1-Score**: Es la media armónica entre la precisión y la exhaustividad. Es una métrica que busca un balance entre estas dos, especialmente cuando se tiene un dataset desbalanceado:

$$\text{F1 Score} = 2 \times \frac{\text{Precisión} \times \text{Exhaustividad}}{\text{Precisión} + \text{Exhaustividad}}$$

Dependiendo de la cantidad de casos positivos y negativos que tengamos en el entrenamiento o del tipo de problema podemos mirar una métrica con más interés o todas.

Evaluación y métricas



Fuente (revisar para mayor detalle)

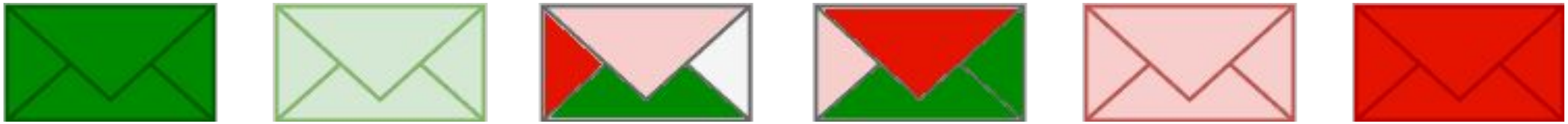
Evaluación y métricas

- Cuando evaluemos en testeo pode ajustar el umbral de decisión para mejorar alguna métrica en particular.
- Pero no siempre es posible disminuir tanto los falsos positivos (FP) como los falsos negativos (FN) simultáneamente. Esto se debe a la naturaleza de los umbrales de decisión, hay un compromiso entre la precisión y la exhaustividad.
- **Precisión:** Se centra en minimizar los falsos positivos, si se ajusta el modelo para que sea más preciso (es decir, dice “spam” solo cuando está muy seguro), se podría reducir los falsos positivos, pero a costa de aumentar los falsos negativos, ya que el modelo podría pasar por alto algunas instancias positivas.
- **Exhaustividad**: Se centra en minimizar los falsos negativos, si se ajusta el modelo para sea más exhaustivo (es decir, para capturar tantos spam como sea posible), se podría reducir los falsos negativos, pero a costa de aumentar los falsos positivos, ya que el modelo podría clasificar erróneamente algunas instancias negativas como positivas.

Hay un compromiso (*trade off*) entre la precisión y la exhaustividad, no se puede mejorar uno sin empeorar el otro

Evaluación y métricas

- Este compromiso entre la precisión y la exhaustividad es inherente al problema de clasificación.
- Para algunos correos va a estar muy seguro que son spam o no, pero para otros no tanto.
- Este **umbral de tolerancia** se puede ajustar una vez entrenado el modelo para mejorar alguna de las métricas.

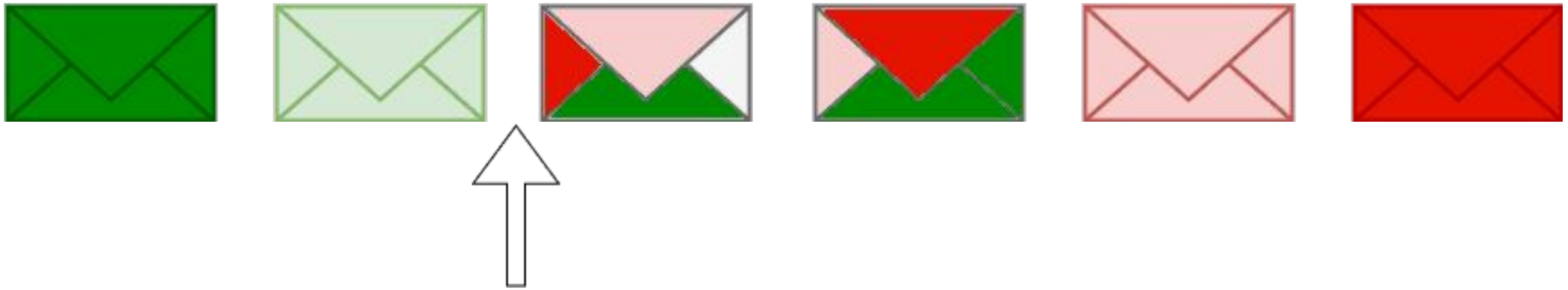


Nota: Recordar que la clase positiva acá es “el correo es spam”

Evaluación y métricas

Decimos que todo esto no es spam

Decimos que todo esto si es spam

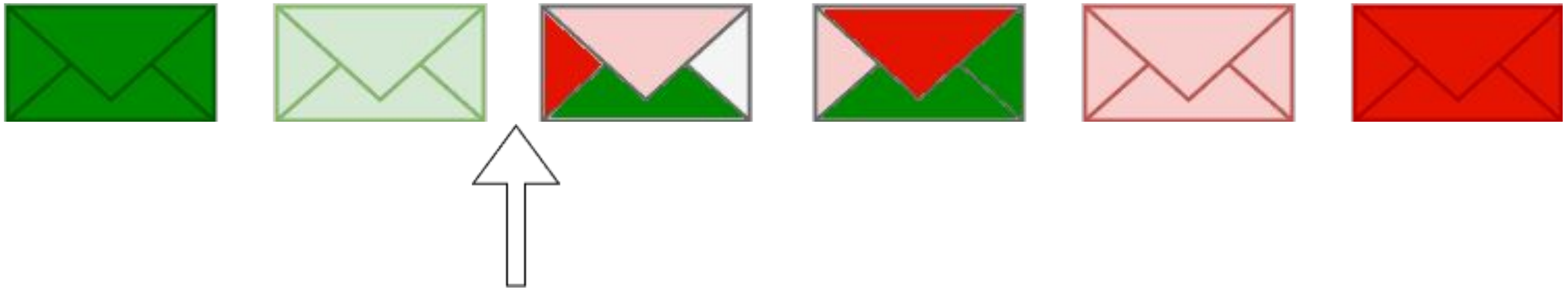


Si lo ponemos acá vamos a estar **más seguros** que no pasa ningún spam (pero se nos puede perder algún correo importante)

Evaluación y métricas

Decimos que todo esto no es spam

Decimos que todo esto si es spam



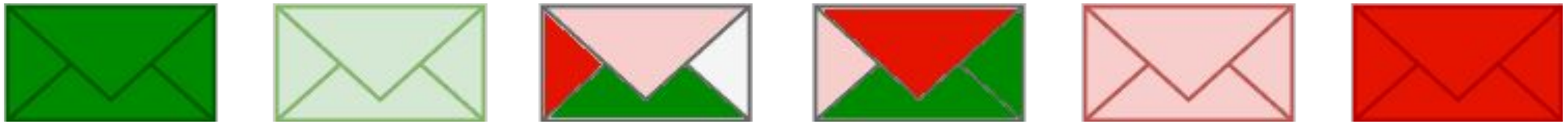
Si lo ponemos acá vamos a estar **más seguros** que no pasa ningún spam (pero se nos puede perder algún correo importante)

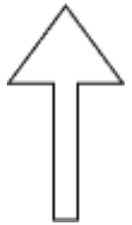
Baja precisión
Alta exhaustividad

Evaluación y métricas

Decimos que todo esto no es spam

Decimos que todo esto si es spam



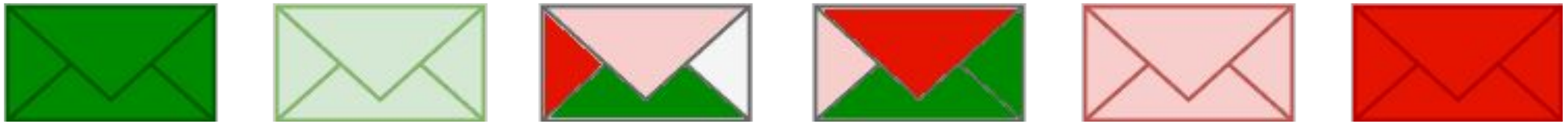


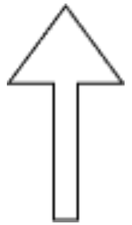
Si lo ponemos acá vamos a estar **más seguros** que no perdemos correos importantes (pero se nos puede colar algún spam).

Evaluación y métricas

Decimos que todo esto no es spam

Decimos que todo esto si es spam





Si lo ponemos acá vamos a estar **más seguros** que no perdemos correos importantes (pero se nos puede colar algún spam).

Alta precisión
Baja exhaustividad

Curva PR

- Una forma de ver esto gráficamente es con una **curva PR** (Precision/Recall Curve).
- Cada punto en la curva representa un par de valores (precisión, exhaustividad) para un umbral específico.
- En general, un modelo con un buen rendimiento tendrá una curva PR que se mantiene cerca del punto superior derecho de la gráfica, donde tanto la precisión como la exhaustividad son altas.

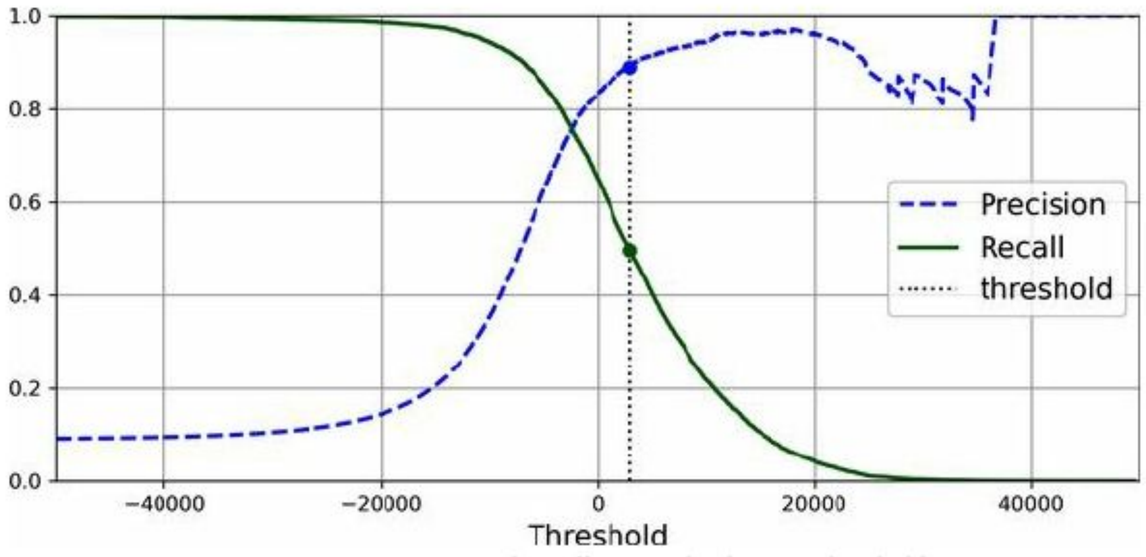


Imagen del Gerón

Curva PR

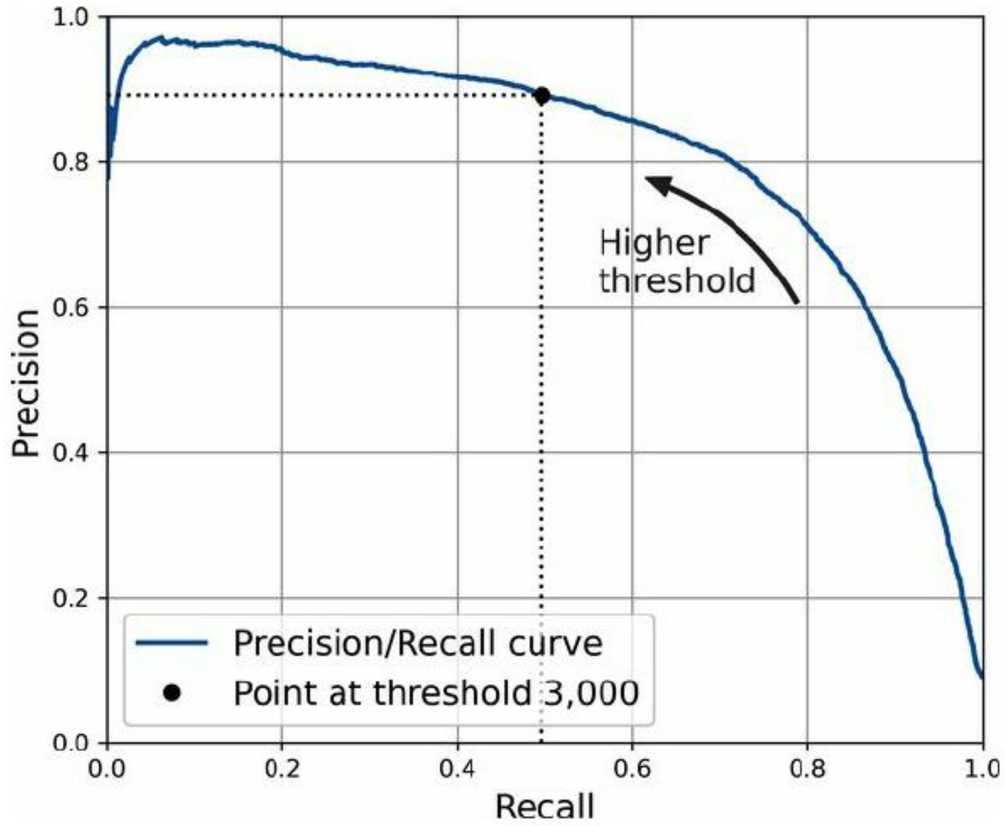


Imagen del Gerón

- Entonces si queremos que nuestro modelo tenga una precisión de 90% podemos encontrar el umbral que nos de ese valor.
- Obviamente ese umbral altera también el resultado en la exhaustividad, porque recordemos, es un compromiso entre los dos.

Curva ROC

- Otra forma de ver esto gráficamente es con una **curva ROC** (Receiver Operating Characteristic).
- También se utiliza para evaluar el rendimiento de un modelo de clasificación binaria, mostrando como cambia la capacidad del modelo para distinguir entre clases positivas y negativas a medida que se ajusta el umbral.
- En vez de mostrar la precisión y la exhaustividad muestra esta última (también llamada sensibilidad o TPR) con la especificidad.
- **Especificidad**: Es la capacidad del modelo para identificar correctamente las instancias negativas. La FPR (Tasa de Falsos Positivos) es la proporción de negativos reales que fueron incorrectamente clasificados como positivos

$$Especificidad = \frac{TN}{TN + FP}$$

Curva ROC

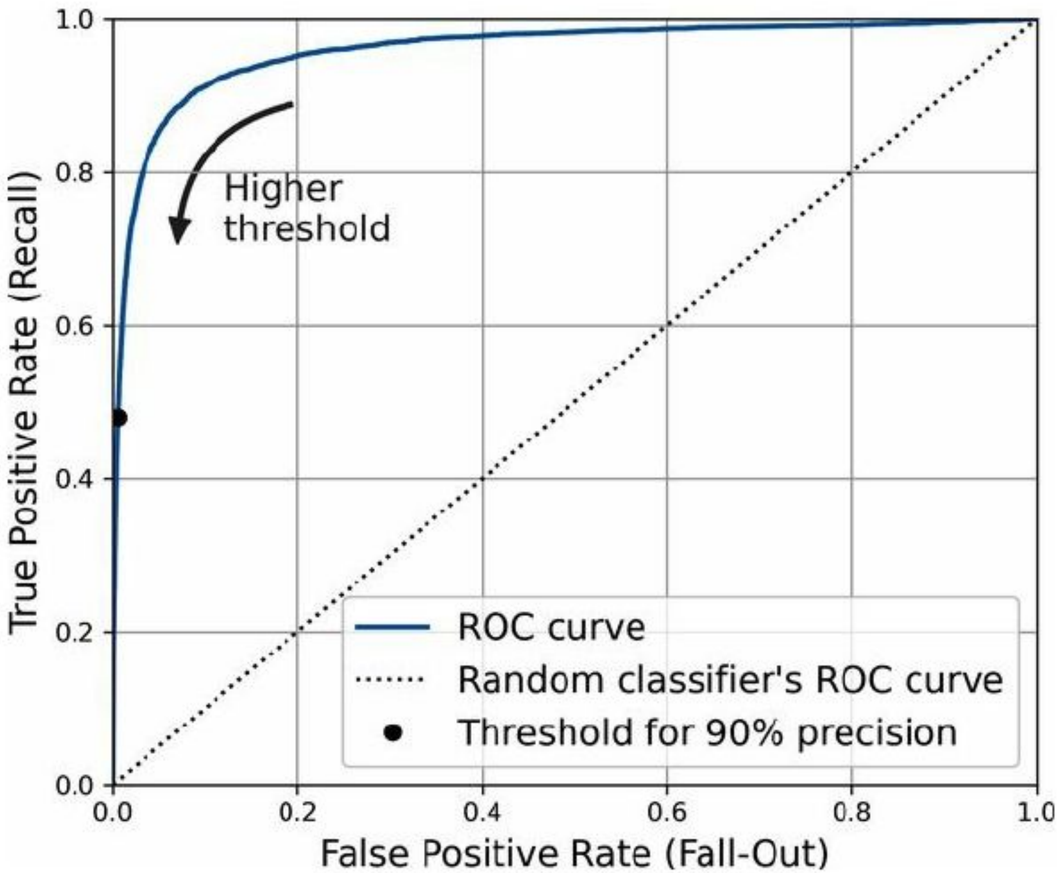


Imagen del Gerón

- Nuevamente si queremos tener un 90% de precisión podemos encontrar el umbral que nos de ese valor.
- Y, nuevamente, también tenemos este compromiso entre ambas métricas.

Curva ROC

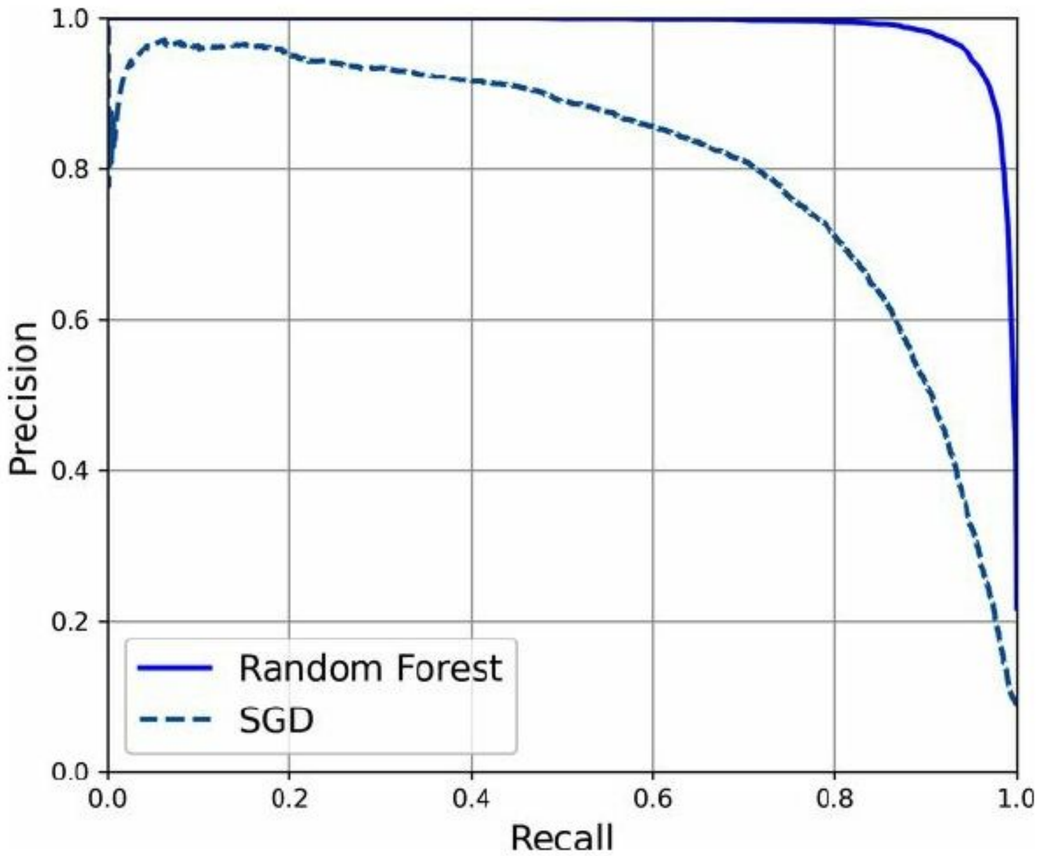


Imagen del Gerón

- Otro uso para estas curvas es medir el área, también llamada AUC (Area Under the Curve).
- Cuanto más grande sea este número mejor comportamiento general tendrá nuestro clasificador.
- Esto también es útil para **comparar diferentes modelos**, el que tenga un AUC más grande va a ser el que mejor se comporte en el caso general.

Parada técnica: Resumen hasta ahora

- **Métricas:** Formas de medir “que tan bien funciona un modelo”. Dependen del tipo de problema:
  - Regresión: Miden “que tanto se alejó el número predicho del valor real”.
  - Clasificación: Más complicado, hay **falsos positivos**, **falsos negativos, precisión, exhaustividad, exactitud, f1-Score, etc.** Dependiendo del problema nos puede interesar mejorar alguno de estos valores u otros.
- **Visualización de métricas y comparación entre modelos:** Matriz de confusión, curva PR, curva ROC.

Nombremos algunos algoritmos...

- Listemos algunos algoritmos frecuentemente usados para cada tipo de problema (aunque con pequeñas modificaciones pueden usarse en ambos casos).
- **Regresión**
  - Regresión Lineal
  - Regresión Ridge y Lasso
  - Regresión Polinomial
- **Clasificación:**
  - Perceptrón / perceptrón multicapa (ya vamos a volver)
  - Support Vector Machines (SVM)
  - Árboles de decisión
  - Random Forest
  - K-Nearest Neighbors (K-NN)
  - Regresión Logística

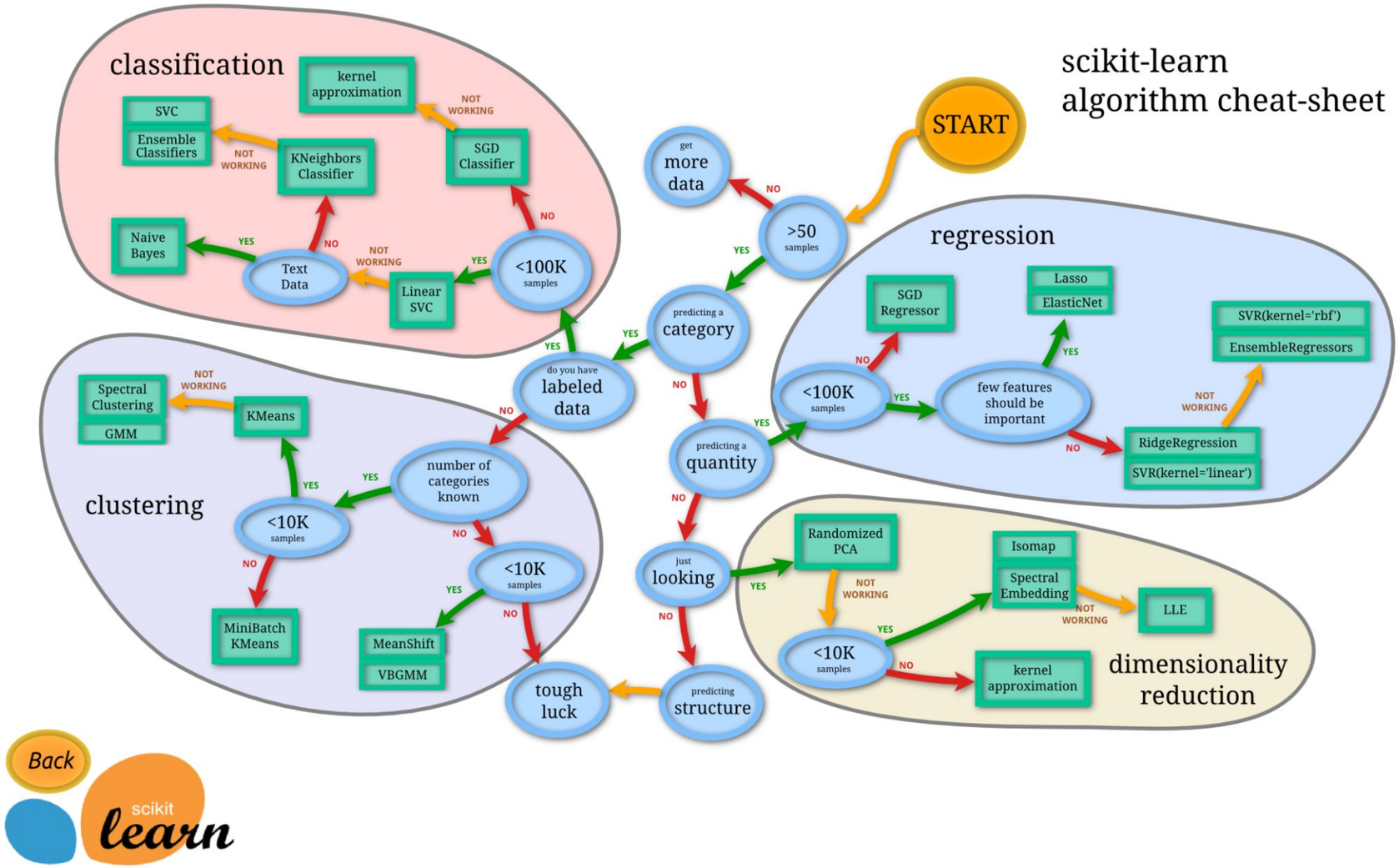
Las famosas Redes Neuronales también pueden usarse para cualquiera de estos problemas

Entonces...

1. Tenemos un problema del que conocemos el dominio y el estado del arte.
2. Identificamos donde se puede usar aprendizaje supervisado, si es regresión o clasificación y definimos las métricas que nos interesan.
3. Conseguimos datos con etiquetas. La calidad y cantidad van a acotar las técnicas que podemos usar y, sobre todo, los resultados (*garbage in*, *garbage out*).
4. Separamos en *Entrenamiento* y *Testeo*.
   a. **Testeo**: se guarda, no se mira, no se estandariza, no se hace **nada**. Tiene que ser lo más parecido posible a los datos que va a recibir en *la vida real*.
   b. **Entrenamiento**: se separa en un conjunto más de **Validación**, se prueban todos los modelos que se quieran en estos conjuntos(*).
5. Seleccionamos **EL** modelo con sus hiperparámetros que sea mejor(*) y lo aplicamos en los datos de *Test*.
6. El resultado que tengamos en *Testeo* es el resultado a reportar como solución del problema o para comparar con la solución anterior.
7. Sale paper/tesis (de preferencia compartiendo no solo el algoritmo e hiperparámetros sino los datos y hasta que semillas se usaron para las separaciones aleatorias. Todo.)

(*) Vayan del más simple al más complejo, evalúen cual sirve más en el contexto de aplicación y seleccionen en función a esto, no solo al resultado de la métrica.

Cheat sheet (idea, no tomarlo como la posta posta)

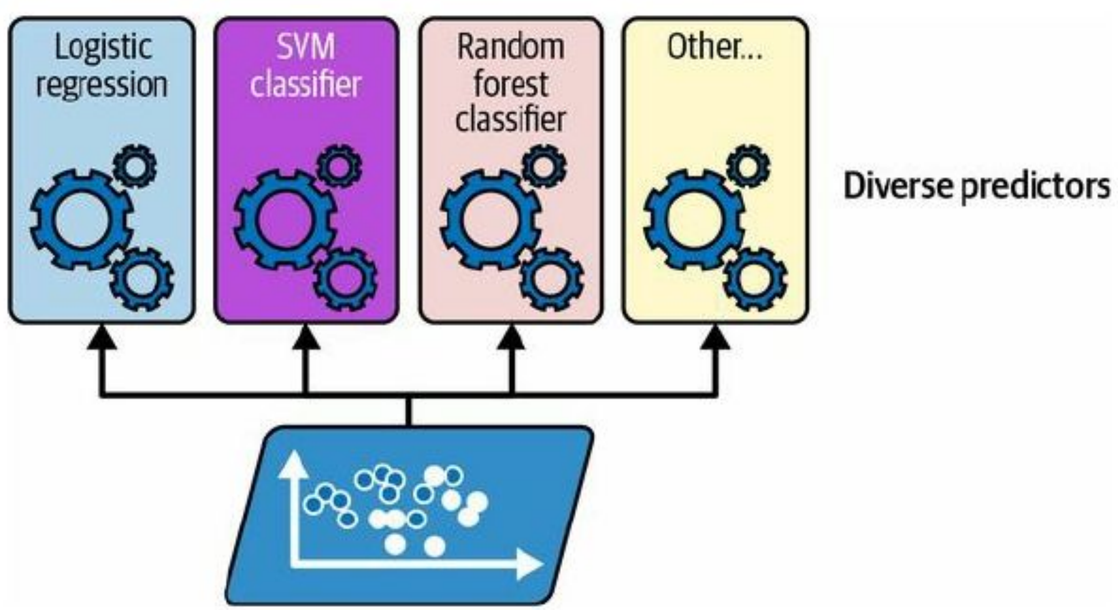


Extras: Métodos de ensamble

- Ya vimos que existen muchos algoritmos para tratar de resolver el mismo problema, todos tienen sus ventajas, desventajas y algunos funcionarán mejor que otros para nuestro problema y los datos que tengamos.
- Una cosa que también se puede hacer es combinarlos, en lo que se conoce como **métodos de ensamble.**
- La idea es combinar las predicciones de varios modelos individuales para mejorar el rendimiento general, buscando que un conjunto de modelos pueda corregir los errores individuales y producir una predicción más precisa y robusta.

Clasificadores de votación (Voting Classifiers)

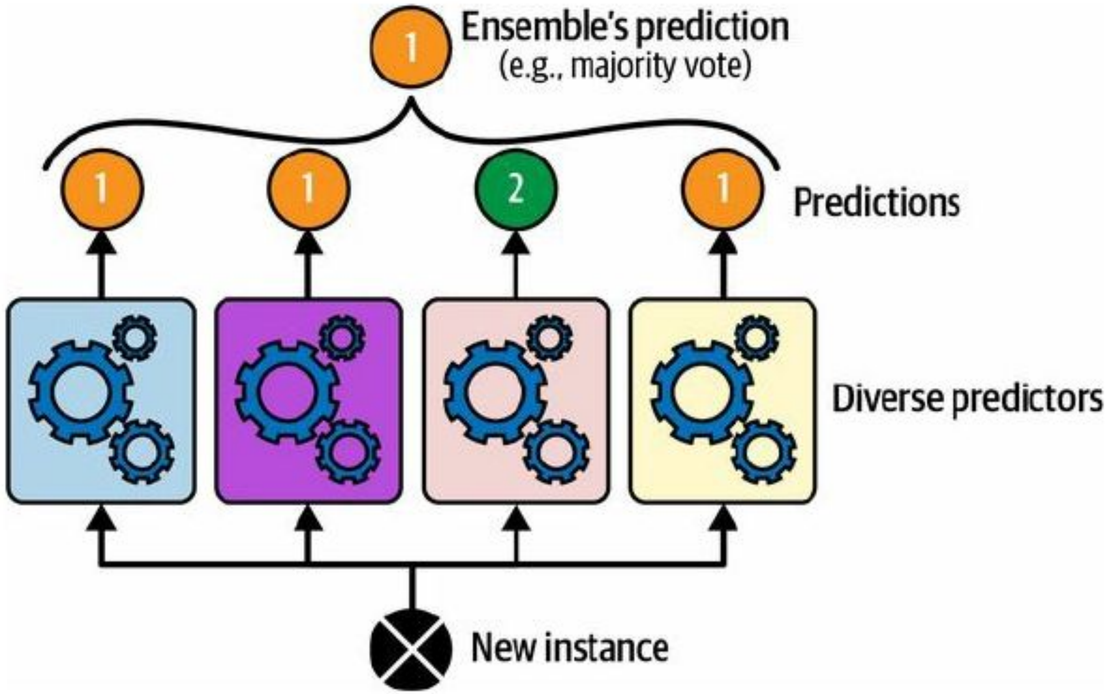
- Supongamos que tenemos algunos clasificadores entrenados (por ejemplo un regresor logístico, un SVM, un random forest, un K-NN, etc), cada uno de los cuales logra una precisión de alrededor del 80 %.



Imágen del Géron

Clasificadores de votación (Voting Classifiers)

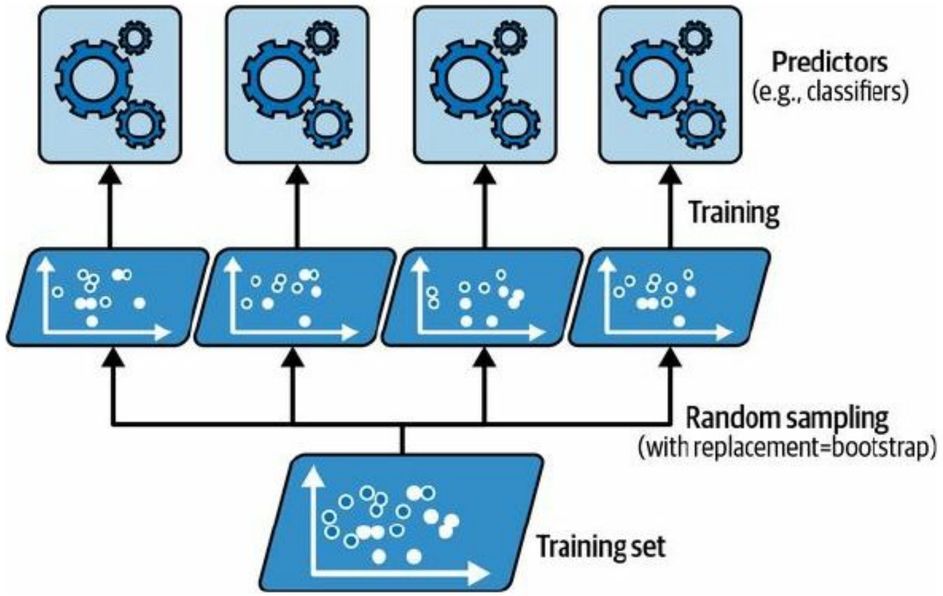
- Una forma muy sencilla de crear un clasificador aún mejor es combinar las predicciones de cada uno: la clase que obtiene más votos es la predicción del conjunto. Este clasificador de mayoría de votos se denomina **clasificador de votación dura** (*hard voting*).
- Esta combinación generalmente alcanza una mejor precisión que el mejor clasificador en el ensamble.



Imágen del Géron

Bagging and Pasting

- Otra opción es, en vez de tener varios clasificadores diferentes, usar el mismo algoritmo pero entrenado en diferentes subconjuntos aleatorios del conjunto de entrenamiento original.
- Cuando este muestreo se realiza con reemplazo se llama *bagging* (*bootstrap aggregating*).
- Cuando el muestreo se realiza sin reemplazo se llama *pasting*.



Imágen del Géron

Bagging and Pasting

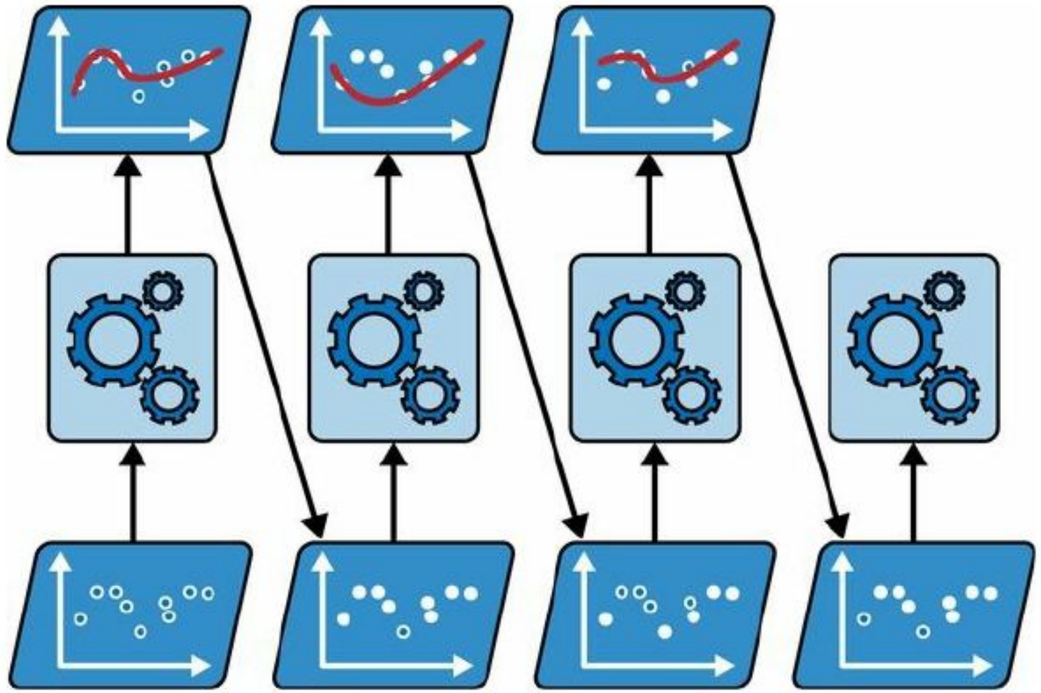
- Una vez entrenados todos los predictores, el ensamble puede realizar una predicción para una nueva instancia simplemente agregando las predicciones de todos los predictores.
- La función de agregación suele ser la moda estadística para la clasificación (es decir, la predicción más frecuente, como en un clasificador de voto duro), o la media para la regresión.

Boosting

- Boosting es un enfoque secuencial en el que los modelos se entrenan uno después del otro, y cada modelo intenta corregir los errores de su predecesor. En cada iteración, se presta más atención a los ejemplos que fueron mal predichos por los modelos anteriores. Los modelos finales se combinan ponderando sus salidas.
- Los más conocidos son **AdaBoost** y **gradient boosting.**

AdaBoost

- Una forma de que un nuevo predictor corrija a su predecesor es prestar más de atención a las instancias de entrenamiento que el predecesor falló más.
- Esto hace que los nuevos predictores se centren cada vez más en los casos difíciles.

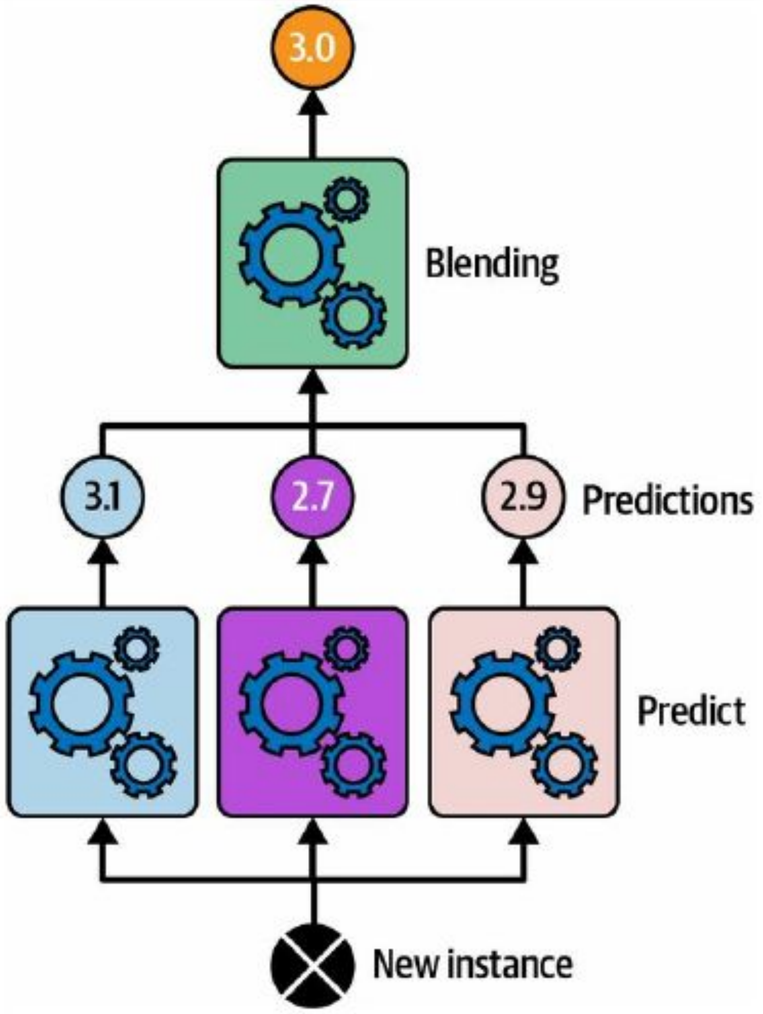


Gradient Boosting

- Otro algoritmo es el refuerzo *Gradient Boosting* (refuerzo de gradiente).
- Al igual que AdaBoost funciona añadiendo secuencialmente predictores a un conjunto, cada uno de los cuales corrige a su predecesor.
- Sin embargo, en lugar de ajustar los pesos de instancia en cada iteración como hace AdaBoost, este método intenta ajustar el nuevo predictor a los errores residuales cometidos por el predictor anterior.

Stacking

- Se basa en una idea sencilla: en lugar de utilizar funciones triviales (como el voto duro) para agregar las predicciones de todos los predictores de un conjunto, ¿por qué no entrenamos un modelo para que realice esta agregación?
- En la imagen se muestra un ejemplo de este tipo realizando una tarea de regresión sobre una nueva instancia. Cada uno de los tres predictores predice un valor diferente (3,1, 2,7 y 2,9) y el predictor final (denominado *blender* o *meta learner*) toma estas predicciones como entradas y realiza la predicción final (3,0).



Stacking

- Para entrenar al mezclador, primero hay que construir el conjunto de entrenamiento de mezclado.
- Se puede utilizar cross_val_predict() en cada predictor del conjunto para obtener predicciones fuera de muestra para cada instancia del conjunto de entrenamiento original, y utilizarlas como características de entrada para entrenar el *blender*.
- Una vez entrenado el mezclador, los predictores base se vuelven a entrenar una última vez en el conjunto de entrenamiento original completo.