

Introducción a Machine Learning

Verónica Ruiz Méndez
vruiz@afi.es

27 y 29 de septiembre de 2022
Afi Escuela de Finanzas

Índice

Introducción a Python

Procesamiento de datos

Modelos de Machine Learning

- Aprendizaje no supervisado

 - Análisis de Componentes Principales

 - Clustering

- Aprendizaje Supervisado

 - Regresión Lineal

 - Regresión Logística

 - K-vecinos

 - SVM

 - Árboles de decisión

 - Ensembles

 - Bagging y Boosting

 - Random Forest

 - XGBoost

 - Voting de modelos

 - Redes Neuronales

- Metodología



¿Qué es Python?

¿Qué es Python?

Historia de Python

- Creado en 1990 por Guido van Rossum (actualmente en Microsoft).
- El nombre está basado en los humoristas británicos Monty Python.
- A partir del año 2001, pasa a ser administrado por Python Software Foundation, una compañía sin ánimo de lucro con un funcionamiento similar al de Apache Software Foundation.



¿Qué es Python?

- Es un lenguaje de programación **de alto nivel**.
- Es un lenguaje de programación **de propósito general**.
- Es un lenguaje de programación **open source**.

¿Qué es *Open Source*?

¿Qué es Open Source?

Historia de Python

- Normalmente el código fuente y los derechos son exclusivos para quienes poseen los derechos de autor.
- *Open Source* es aquel *software* que tiene tanto el código fuente como los derechos sin ninguna restricción.
 - Cualquiera puede acceder al código fuente y modificarlo si lo desea (para su propio uso)
 - El uso no está restringido para ningún uso y/o usuario.
- La FSF (*Free Software Foundation*) establece cuatro libertades para considerar un *software Open Source*:
 - Libertad para **usar** el software como se desee y con cualquier propósito.
 - Libertad para **analizar** el funcionamiento del *software* y poder cambiarlo para lo que se desee.
 - Libertad para **distribuir** copias.
 - Libertad para **modificar** y distribuir copias de versiones modificadas a terceros.

“With software there are only two possibilities: either the users control the programme or the programme controls the users.” Richard Stallman.

¿Qué es Python?

- Es un lenguaje de programación **de alto nivel**.
- Es un lenguaje de programación **de propósito general**.
- Es un lenguaje de programación **open source**.
- Es un lenguaje de programación **orientado a objetos**.
- Es un lenguaje de programación **dinámicamente tipado y fuertemente tipado**.
- Es un lenguaje de programación **conciso**.
- Es un lenguaje de programación **con una comunidad muy activa**.
- Es un lenguaje de programación **con infinidad de módulos orientado a muy diferentes dominios** (tratamiento de imágenes, videojuegos, bases de datos, **análisis de datos**, etc.).




¿Qué es Python?

Ventajas

- Python es un **lenguaje de alto nivel multipropósito**. También se utiliza en otros campos más allá del análisis de datos: desarrollo web, scripting ...
- Es un lenguaje más rápido en ejecución (respecto a otros más basados en estadística, por ejemplo R).
- Es **muy fácil de aprender para los participantes**: curva de aprendizaje menos dura.
- La sintaxis del lenguaje te ayuda a ser un mejor programador: código más condensado y legible.
- Más rápido en el manejo de grandes conjuntos de datos y puede cargar los archivos con facilidad.

¿Qué es Python?

¿Cómo usar Python?

Distribución	Descripción	url_descarga
	<ul style="list-style-type: none">- Core de Python.- Incluye únicamente los paquetes básicos y el intérprete de la consola de comandos.	https://www.python.org/
	<ul style="list-style-type: none">- Distribución más extendida y reconocida de las existentes.- Incluye más de 300 modulos preinstalados desde análisis de datos, hasta desarrollo web pasando por librerías matemáticas.- Incluye una consola gráfica para Python, iPython, Jupyter Notebooks, Spyder y VisualStudioCode.	https://www.anaconda.com/products/individual
	<ul style="list-style-type: none">- Distribución más orientada al análisis científico, con paquetes matemáticos mucho más específicos.- También muy centrada en la visualización de datos, incluyendo paquetes de visualización de datos avanzados.	https://assets.enthought.com/downloads/

¿Qué es Python?

¿Cómo usar Python?

- Existen muchos IDEs (*Integrated Development Environment*) para Python.
- Cada uno de ellos está diseñado para dar soporte a una forma de trabajo en función del dominio (análisis de datos, desarrollo general, programación reproducible...) al que se orienten.
- Se pueden encontrar desde consolas básicas (tipo R o Matlab) hasta entornos completos de desarrollo y despliegue de aplicaciones y servicios (tipo Visual Studio, Eclipse, NetBeans, etc.).



¿Qué es Python?

Instalación de paquetes

PIP

Administrador estándar de paquetes de Python. El instalador de Python instala pip automáticamente. Para instalar paquetes con pip basta con escribir en la consola “pip install nombre_paquete”. Los paquetes se descargan del repositorio oficial de paquetes de Python, llamado PyPi.

CONDA

Gestor de paquetes y entornos de Python. Conda está incluido al instalar la distribución Anaconda. Para instalar paquetes, debemos escribir “conda install nombre_paquete”. Los paquetes se descargan del repositorio de Anaconda.

POETRY

Herramienta para gestionar proyectos en Python. Permite definir los paquetes necesarios en nuestro proyecto y los instala/actualiza.

Ejemplo en Python



IntroduccionML_BasicsPython.ipnyb

¿Qué es Python?

Librerías - NUMPY

- [NumPy](#) es un módulo de Python, abreviatura de Numerical Python.
- Ofrece estructuras de datos y funciones matemáticas complejas de replicar en el core de Python (computación matricial, operaciones matemáticas sobre grandes conjuntos de información, etc.).
- Dicen que...

Python + NumPy + SciPy + Matplotlib -> MATLAB

*Scipy es una librería de Python que se compone de herramientas y algoritmos matemáticos.

*Matplotlib es una librería de Python para generar gráficos.

¿Qué es Python?

Librerías - NUMPY

Capacidades principales:

- Estructura de datos para almacenamiento en forma de matrices n-dimensionales: ndarray.
- Funciones matemáticas estándar con rendimiento optimizado para ser aplicadas a matrices complejas sin necesidad de usar bucles.
- Herramientas para la lectura y escritura de información matricial a disco.
- Funciones para aplicación de álgebra lineal.
- Herramientas para integrar código escrito en C, C++ o Fortran.

Ejemplo en Python



`IntroduccionML_Numpy.ipnyb`

¿Qué es Python?

Librerías - PANDAS

- [Pandas](#) es un módulo de Python orientado al análisis de datos.
- Creado por Wes McKinney. La primera versión se publicó en 2008.
- Una de las librerías con más evolución y seguimiento por parte de la comunidad (más de 200 contribuidores).

¿Qué es Python?

Librerías - PANDAS

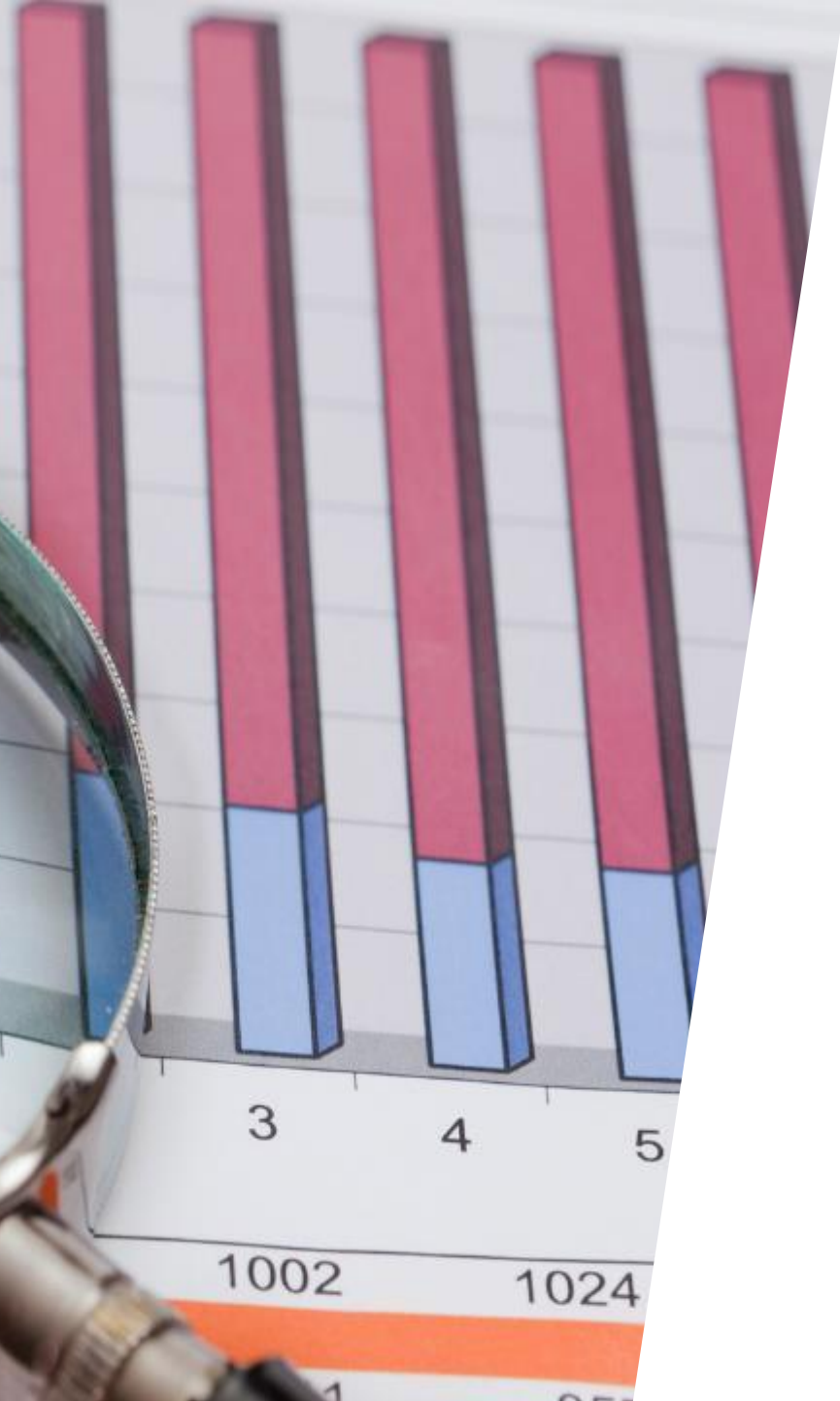
Características:

- Capacidad de almacenamiento y procesamiento de diferentes estructuras de datos.
- Facilidad para la carga de información desde diferentes fuentes: ficheros CSV, bases de datos relacionales...
- Capacidad para el tratamiento de missing values.
- Utilidad tanto para carga y tratamiento de datos, como para el análisis estadístico, exploratorio y modelado.
- Integración con otras librerías.

Ejemplo en Python



IntroduccionDS_Pandas.ipnyb



Preprocesamiento de datos

Preprocesamiento de datos

Problemas que pueden presentar los datos

- **Incompletos:** atributos que carecen de valores, atributos sin interés... (missing values).
- **Ruidosos:** contienen errores o “outliers”.
- **Inconsistentes:** discrepancias en códigos o nombres.
- **Tamaño excesivo:** filas y/o columnas.

Sin datos de calidad, no hay calidad en los resultados

Preprocesamiento de datos

Limpieza de datos

La **limpieza de datos** es el conjunto de operaciones que:

- Corrigen datos erróneos.
- Filtran datos incorrectos.
- Reducen un innecesario nivel de detalle.
- Detectan y resuelven discrepancias.

El proceso de limpieza de datos intenta completar los valores perdidos, suavizar el ruido al identificar valores atípicos y corregir inconsistencias en los datos.

Preprocesamiento de datos

Limpieza de datos

La **limpieza de datos** es el conjunto de operaciones que:

- Corrigen datos erróneos.
- Filtran datos incorrectos.
- Reducen un innecesario nivel de detalle.
- Detectan y resuelven discrepancias.

El proceso de limpieza de datos incluye la validación y corrección de datos, para obtener datos de calidad.

La **calidad de los datos** se consigue cuando se cumplen:

- **Integridad:** deben cumplir requisitos de entereza y validez.
- **Consistencia:** corrección de contradicciones.
- **Uniformidad:** relacionado con las irregularidades.
- **Densidad:** valores omitidos sobre el número de valores totales.
- **Unicidad:** no tener duplicados ni registros inconsistentes.



Preprocesamiento de datos

Transformación de datos

La **discretización** es el proceso por el cual, se convierten variables continuas en variables categóricas.

Al realizar esta transformación, se pierde información. Sin embargo, hay **algoritmos** (por ejemplo, algoritmos de clasificación) que **necesitan** que los datos de entrada sean **variables categóricas**.

Esto hace que **los datos sean más fáciles de estudiar y analizar**, y **mejora la eficiencia** de las tareas que queramos realizar con ellos.

Preprocesamiento de datos

Normalización de datos

La **normalización** trata de conseguir que todas las variables estén expresadas en una escala similar, para que todas ellas tengan un peso comparable.

Algunos ejemplos habituales:

Normalización Z-score

$$z = \frac{x - \bar{x}}{s}$$

Los datos serán estandarizados siguiendo una distribución Normal(0,1), donde

- media = 0
- desviación típica = 1

Normalización Min-Max

$$x = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Los datos serán escalados en un rango fijo, normalmente entre 0-1.

Preprocesamiento de datos

Missing values

Un **missing value** (valor faltante, perdido o desconocido) es un atributo que no está almacenado. En la librería Pandas de Python, se representan como None y Nan (acrónimo de Not a Number, valor especial de punto flotante).

Tratamiento de missing values

- Eliminar filas, cuando la mayoría de los atributos de cierta observación son missing values.
- Eliminar columnas, cuando el atributo no representa valores para la mayoría de las observaciones.
- Imputar su valor, en situaciones intermedias, cuando sea necesario.

Lo más habitual a la hora de imputar missing values es rellenarlo según alguna medida de resumen (media, moda, mediana...), lo que puede presentar problemas:

- Reduce la dispersión del atributo.
- Se utiliza menos información de la disponible.

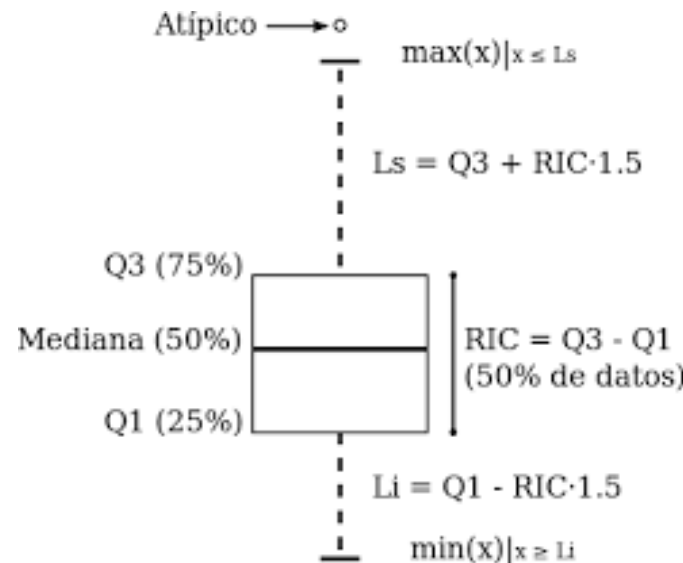
Preprocesamiento de datos

Outliers

Barnet y Lewis (1994) definen **outlier** o valor atípico en un conjunto de datos como una observación (o conjunto de observaciones) que parecen ser inconsistentes con ese conjunto de datos.

Outliers no es igual a error. Los valores atípicos deben ser detectados. Su inclusión o no en el análisis depende del estadístico.

Gráficamente, podemos usar el **diagrama de cajas y bigotes de Tuckey**, conocido como box-plot.



Preprocesamiento de datos

Acciones sobre outliers

- **Ignorar:** algunos algoritmos son robustos a outliers.
- **Filtrar, eliminar o reemplazar la columna.**
- **Filtrar la fila:** sesga los datos.
- **Reemplazar el valor:** se debe analizar cuál es el método más adecuado, pudiendo reemplazar por un valor “nulo”, máximo, mínimo, media... e incluso, se puede predecir utilizando alguna técnica de Machine Learning.
- **Discretizar.**

Ejemplo en Python



IntroduccionML_PreprocesadoInformacion.ipynb



Modelos de Machine Learning

Modelos de Machine Learning

Aprendizaje supervisado y no supervisado

APRENDIZAJE SUPERVISADO

Para cada una de las observaciones de las variables explicativas, tenemos una variable respuesta.

El objetivo es predecir la respuesta de futuras observaciones (predicción) o de comprender mejor la relación entre la variable respuesta y las variables predictivas (inferencia).

El aprendizaje supervisado busca patrones en datos históricos relacionando todos los campos con un campo objetivo.

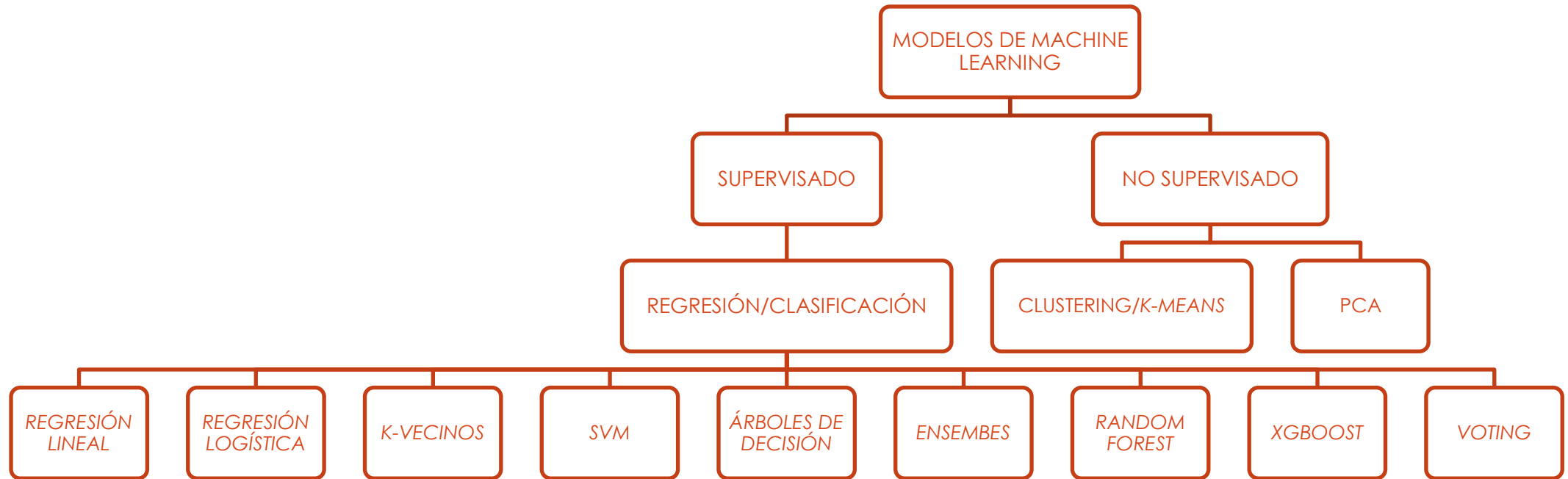
- **Regresión:** trata de predecir un número.
- **Clasificación:** trata de predecir una categoría.

APRENDIZAJE NO SUPERVISADO

Para cada observación, tenemos un vector de medidas que no lleva asociada una respuesta.

No tenemos una variable que predecir, ni que pueda supervisar nuestro análisis. El objetivo es entender los datos y organizarlos en grupos.

Modelos de Machine Learning

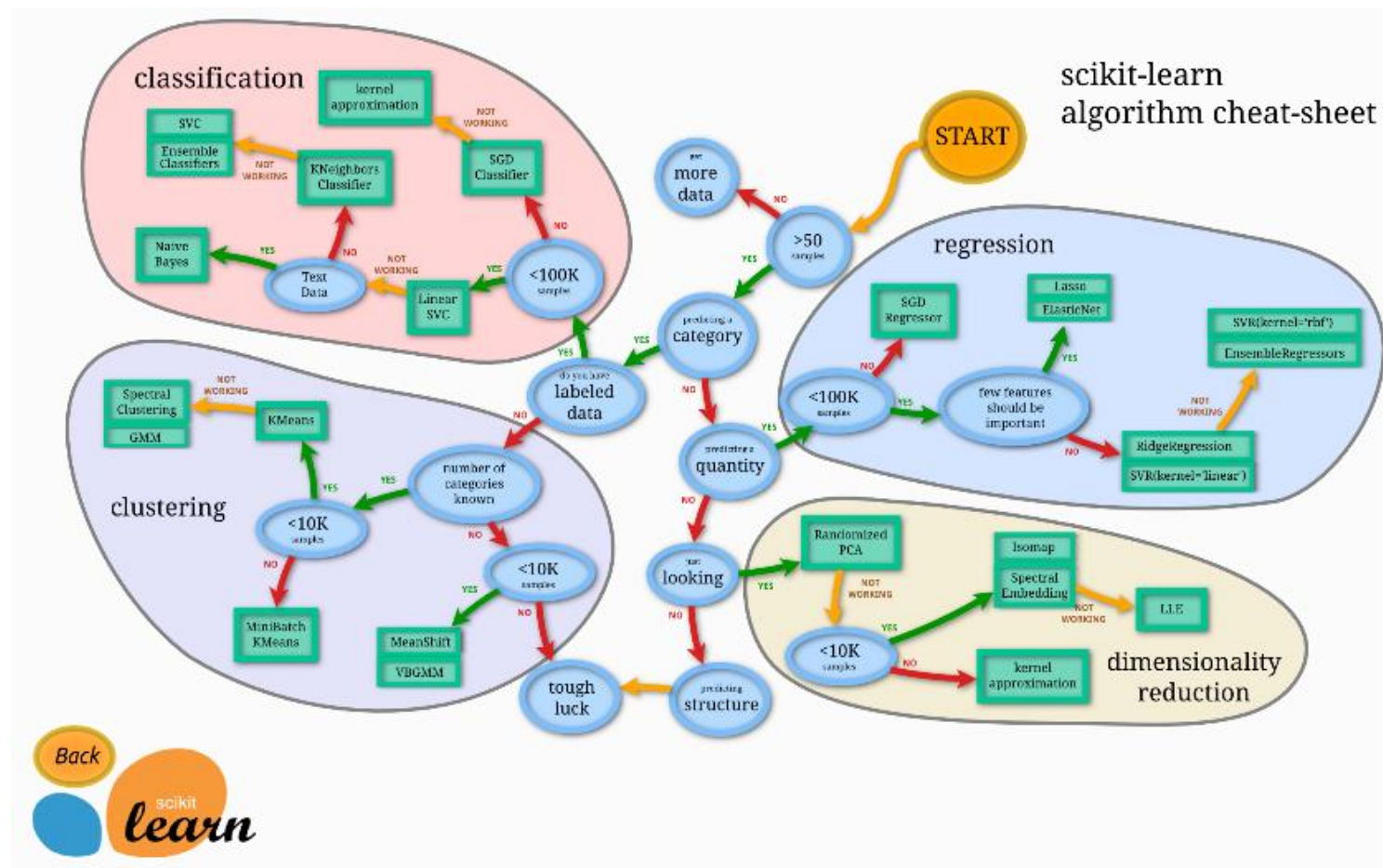


Y... ¡muchos más!

Modelos de Machine Learning en Python

Scikit-learn

Librería de Python utilizada para aprendizaje automático.



https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html

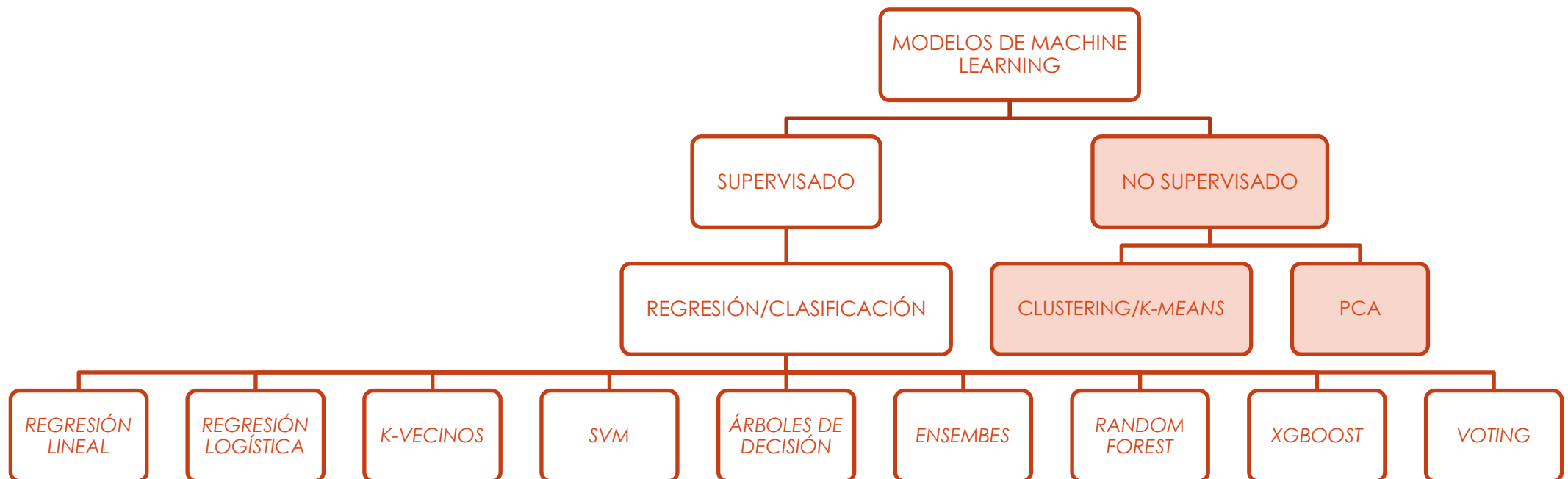



Aprendizaje no supervisado

Aprendizaje no supervisado

Para cada observación, tenemos un vector de medidas que no lleva asociada una respuesta.

No tenemos una variable que predecir, ni que pueda supervisar nuestro análisis. El objetivo es entender los datos y organizarlos en grupos.





Aprendizaje no supervisado Análisis de Componentes Principales

Aprendizaje no supervisado

Análisis de Componentes Principales (PCA)

El Análisis de Componentes Principales fue introducido por primera vez por **Pearson** en **1901** y desarrollado por **Hotelling** en **1933** como método descriptivo de simplificación o reducción de la dimensión de un conjunto de datos con la menor pérdida de información posible.

El **objetivo** es la reducción de la dimensión de las variables observadas encontrando combinaciones lineales de las variables que tienen varianza máxima y no están correlacionadas entre sí.

Cuando las variables originales se encuentran muy correlacionadas entre sí, la mayor parte de su variabilidad puede explicarse con muy pocas componentes. Si las variables originales son incorreladas entre sí, el análisis de componentes principales carece de interés ya que las componentes principales coincidirán con las variables originales.

Aprendizaje no supervisado

Obtención de las Componentes Principales

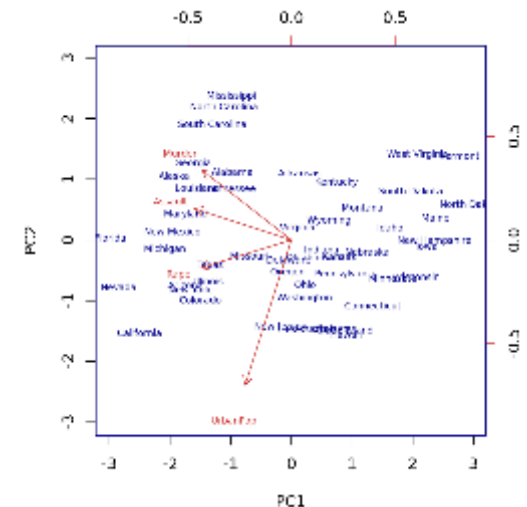
Sea X la matriz de datos, se desea obtener un conjunto de variables incorreladas z_1, \dots, z_p (donde p es el rango de X), y combinaciones lineales de las variables originales X_1, \dots, X_p .

$$z_j = a_{j1}x_1 + a_{j2}x_2 + \dots + a_{jp}x_p$$
$$\text{var}(z_j) = \text{var}(a_j x) = a_j^T S a_j$$

donde S es la matriz de varianzas covarianzas de las variables originales.

Buscamos encontrar la primera componente principal, z_1 , que maximice la varianza, de forma que a_1 esté normalizado, es decir:

$$\text{Máx}_{a_1} \text{var}(z_1) \text{ sujeto } a_1^T a_1 = 1$$



Aprendizaje no supervisado

Obtención de las Componentes Principales

Sea una muestra con n individuos, cada uno con p variables (X_1, X_2, \dots, X_p). PCA permite encontrar un número de factores ($z < p$) que expliquen aproximadamente lo mismo que las p variables originales. Cada una de las z nuevas variables se llaman **componente principal**.

Dado un set de datos, el proceso a seguir para calcular la primera componente principal (Z_1) es:

- Centrar las variables: se resta a cada valor la media de la variable. Con esto se consigue que todas las variables tengan media cero.
- Se resuelve el problema de optimización para encontrar los valores que maximizan la varianza. Para resolver esta optimización, se calculan autovalores y autovectores mediante la matriz de covarianzas.

Una vez calculada la primera componente (Z_1), se calcula la segunda (Z_2) repitiendo el mismo proceso, añadiendo la condición de que la combinación lineal no esté correlacionada con la primera componente, es decir, que Z_1 y Z_2 sean perpendiculares. Se repite el proceso de forma iterativa. El orden de las componentes viene dado por el autovalor asociado a cada autovector.

Aprendizaje no supervisado

Componentes Principales en Python

Para calcular PCA en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(n_components=2)
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.0075...]
```


Aprendizaje no supervisado

Componentes Principales en Python

Para calcular PCA en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(n_components=2)
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.0075...]
```

Aprendizaje no supervisado

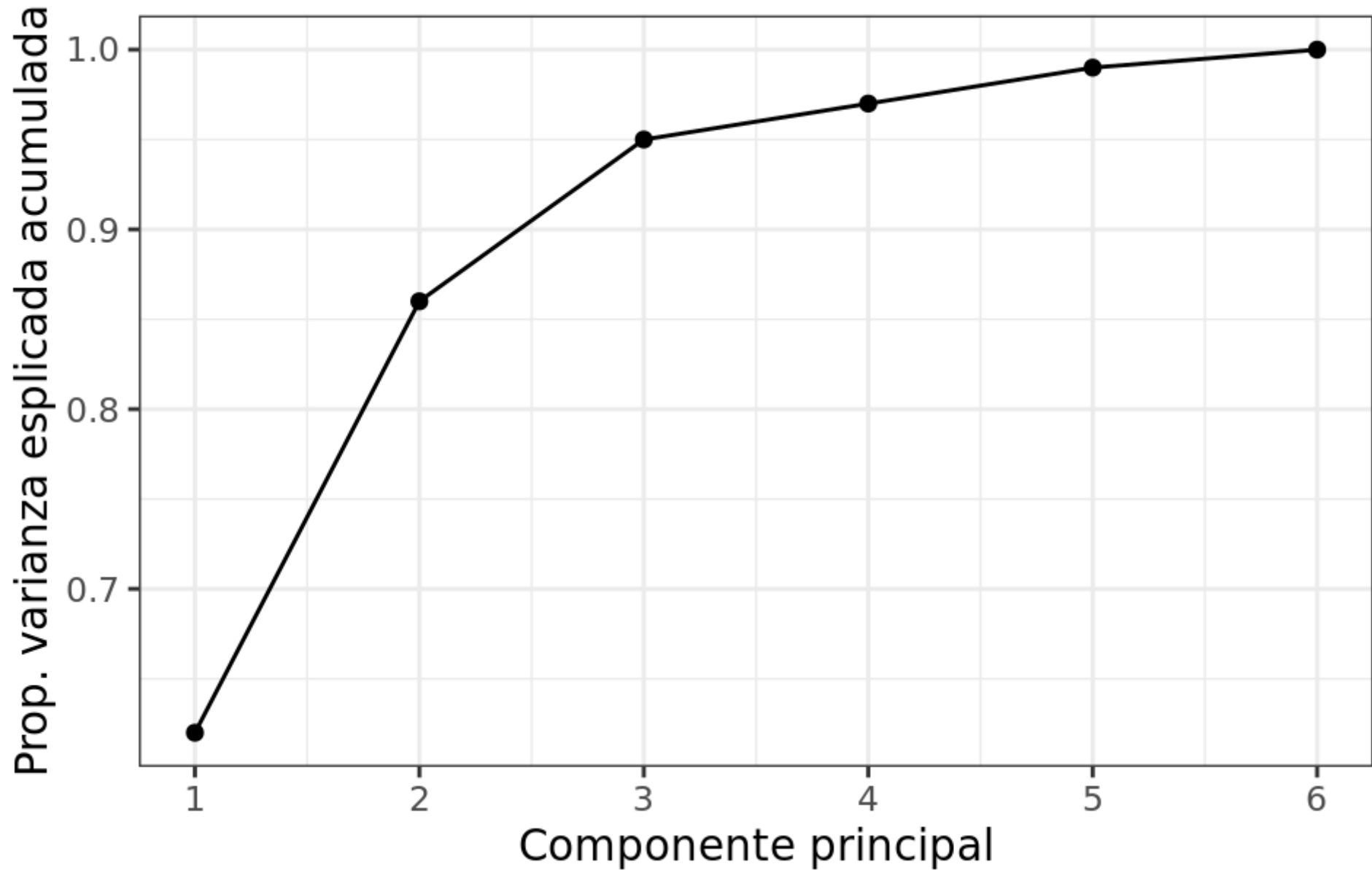
Componentes Principales en Python

Para calcular PCA en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> import numpy as np
>>> from sklearn.decomposition import PCA
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> pca = PCA(n_components=2)
>>> pca.fit(X)
PCA(n_components=2)
>>> print(pca.explained_variance_ratio_)
[0.9924... 0.0075...]
```

Porcentaje de varianza explicada para cada componente principal.

Aprendizaje no supervisado



Ejemplo en Python



IntroduccionML_PCA.ipnyb



Aprendizaje no supervisado Clustering

Aprendizaje no supervisado

Clustering

El objetivo del clustering es agrupar los elementos en bloques homogéneos en función de las similitudes entre ellos. Siendo G el número de grupos y p el número de variables, se quiere conseguir:

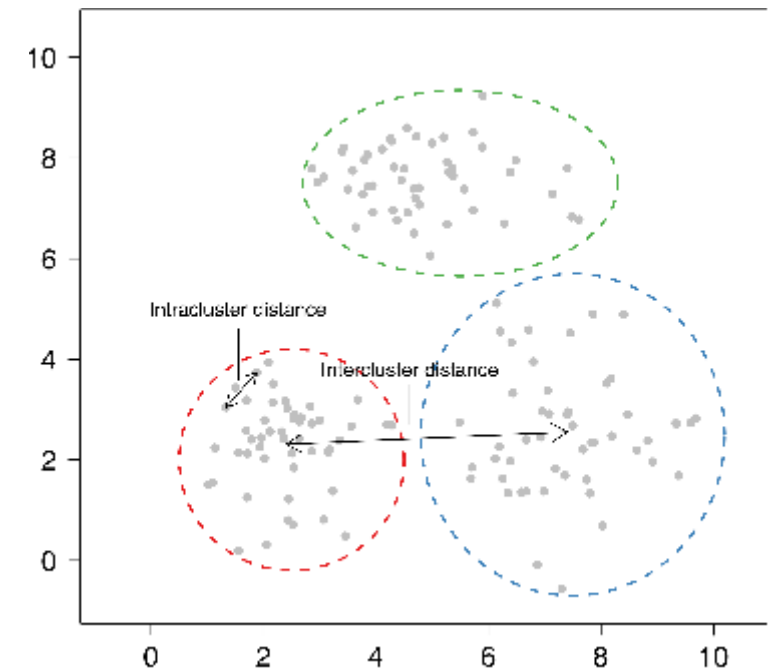
- Minimizar la varianza dentro de cada grupo.

$$WSS = \sum_{i=1}^{n_g} \sum_{j=1}^p (x_{ij} - \bar{x}_j)^2$$

WSS: Within Cluster Sum of Square

- Maximizar la varianza entre grupos.

$$\text{Intercluster dissimilarity (distance)} = \sum_{g=1}^G \sum_{j=1}^p (\bar{x}_{gj} - \bar{x}_j)^2$$

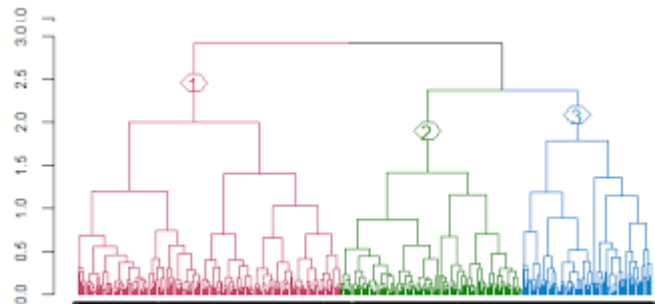


Aprendizaje no supervisado

Clustering – algoritmos de clasificación

Métodos jerárquicos

Durante su aplicación se construye una jerarquía representable a través de dendrogramas.



Métodos de partición

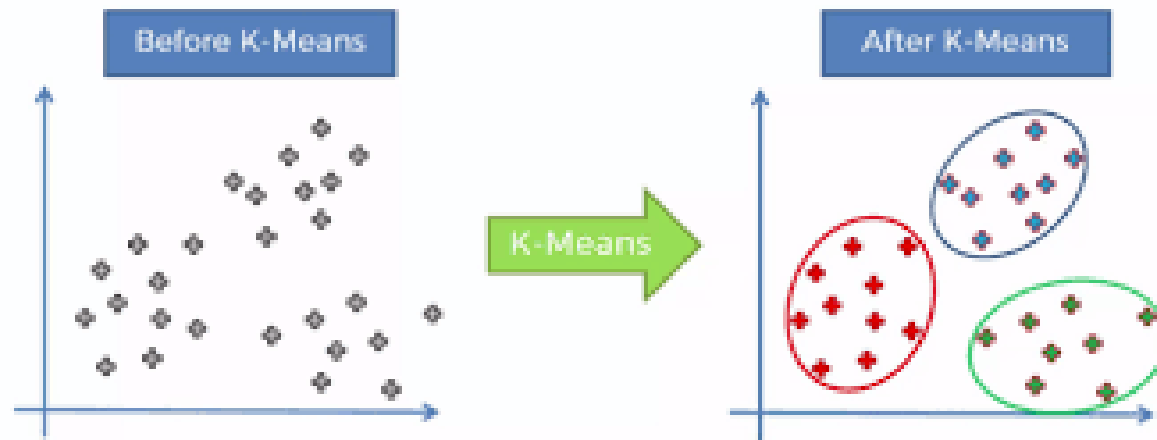
Dividen las observaciones en un número de grupos pre-especificado.



Aprendizaje no supervisado

Clustering – k-means

Algoritmo que asigna aleatoriamente a cada observación uno de los K grupos. Calcula las K medias de cada muestra (centroide) de las observaciones de cada grupo.



<http://shabal.in/visuals/kmeans/2.html>

Aprendizaje no supervisado

Clustering en Python

Para calcular k-means en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

Aprendizaje no supervisado

Clustering en Python

Para calcular k-means en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

Obtener etiquetas,
a qué cluster
pertenece cada
observación.

Aprendizaje no supervisado

Clustering en Python

Para calcular k-means en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn.cluster import KMeans
>>> import numpy as np
>>> X = np.array([[1, 2], [1, 4], [1, 0],
...               [10, 2], [10, 4], [10, 0]])
>>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
>>> kmeans.labels_
array([1, 1, 1, 0, 0, 0], dtype=int32)
>>> kmeans.predict([[0, 0], [12, 3]])
array([1, 0], dtype=int32)
>>> kmeans.cluster_centers_
array([[10.,  2.],
       [ 1.,  2.]])
```

Centroides de
cada uno de
los clusters.

Ejemplo en Python



IntroduccionML_Clustering.ipnyb

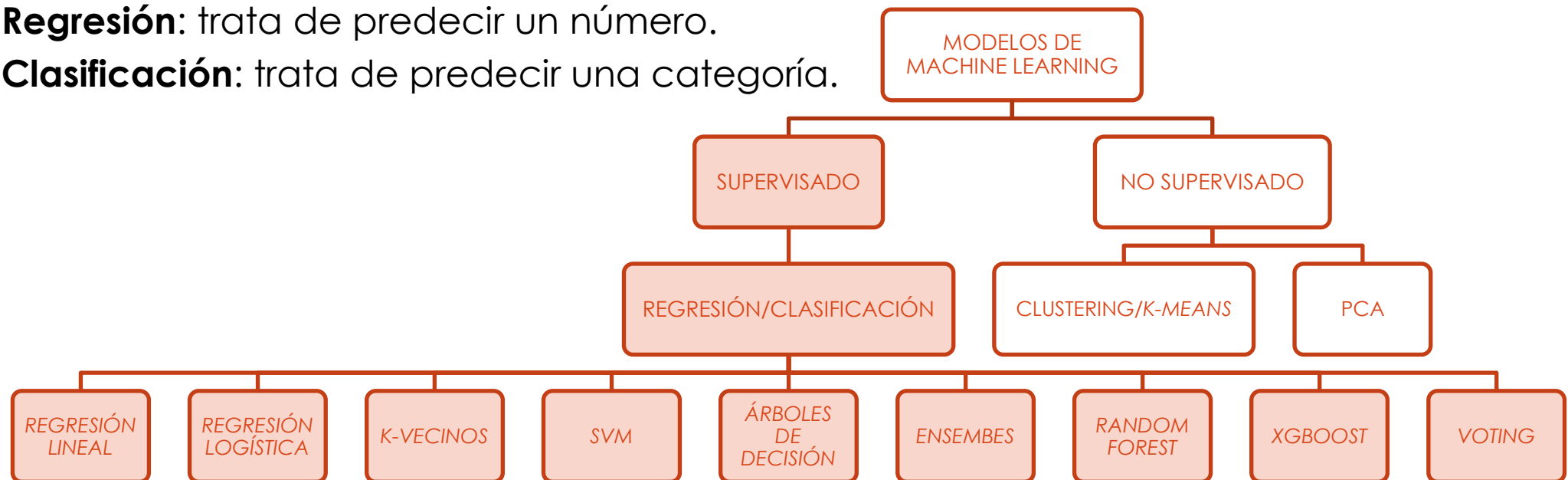
Aprendizaje supervisado

Aprendizaje supervisado

Para cada una de las observaciones de las variables explicativas, tenemos una variable respuesta.

El objetivo es predecir la respuesta de futuras observaciones (predicción) o de comprender mejor la relación entre la variable respuesta y las variables predictivas (inferencia). El aprendizaje supervisado busca patrones en datos históricos relacionando todos los campos con un campo objetivo.

- **Regresión:** trata de predecir un número.
- **Clasificación:** trata de predecir una categoría.



Aprendizaje supervisado

Regresión lineal

Aprendizaje supervisado

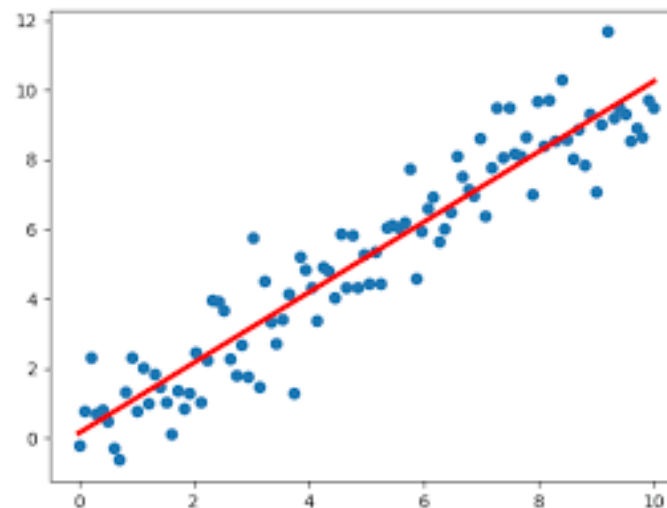
Regresión Lineal

El modelo de **regresión lineal** es una de las formas más sencillas de estimar nuestra variable Y a partir de los predictores. La regresión lineal resuelve un problema de **regresión**, donde la variable respuesta es continua.

Asumimos que existe una relación: $Y \approx \beta_0 + \beta_1 X$

A partir de los datos disponibles estaremos los **coeficientes** $\hat{\beta}_0$ y $\hat{\beta}_1$, obteniendo como función h :

$$Y \approx \hat{Y} = h(X) = \hat{\beta}_0 + \hat{\beta}_1 X$$



Aprendizaje supervisado

Regresión Lineal

Para calcular la recta de regresión usaremos el método de **mínimos cuadrados**. Para ello, definimos el **residuo** como:

$$e_i = y_i - (\beta_0 + \beta_1 x_i)$$

Tendremos el **error cuadrático medio** como:

$$MSE = \frac{1}{n} \sum_{i=1}^n e_i^2$$

Para minimizar este error, los coeficientes deben ser:

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

Aprendizaje supervisado

Regresión Lineal

Para medir la precisión entre los coeficientes $\hat{\beta}_0$ y $\hat{\beta}_1$, debemos asumir la existencia de una relación lineal entre X y Y verificando:

$$Y = \beta_0 + \beta_1 X + \varepsilon, \varepsilon \sim N(0, \sigma^2), \varepsilon \perp X$$

Nos referimos a la expresión anterior como:

$$SE(\hat{\beta}_0)^2 = \sigma^2 \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right], SE(\hat{\beta}_1)^2 = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

En estos casos, se estima el **error residual estándar** como:

$$\sigma \approx RSE = \sqrt{\frac{1}{n-2} \sum_{i=1}^n e_i^2}$$

Aprendizaje supervisado

Regresión Lineal

Las expresiones anteriores permiten calcular **intervalos de confianza** y realizar contrastes de hipótesis relativos a los coeficientes del modelo, ya que:

$$\frac{\hat{\beta}_j - \beta_j}{SE(\hat{\beta}_j)} \sim t_{n-2}, \quad j = 0, 1$$

Esto nos permite contrastar si la respuesta está significativamente influida por alguna variable, mediante el contraste con hipótesis nula:

$$H_0: \beta_0 = 0$$

Aprendizaje supervisado

Regresión Lineal

La medida más utilizada para la bondad es la **raíz cuadrada del error cuadrático medio**:

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

Otra alternativa muy utilizada es el estadístico R^2 que se define como:

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

Que representa la proporción de variabilidad de la variable respuesta que queda explicada por el modelo.

Aprendizaje supervisado

Regresión Lineal - Regularización

Problemas del modelo de regresión lineal (Machine Learning):

- Problemas en el modelo al incorporar predictores correlacionados.
- No se realiza una selección de predictores en función de la relevancia de la información que poseen.
- No puede ajustarse un modelo cuando el número de predictores es superior al de observaciones.

Aprendizaje supervisado

Regresión Lineal - Regularización

SOLUCIÓN: Regularización

- Fuerzan a que los coeficientes del modelo tiendan a cero.
- Minimizan el riesgo de overfitting.
- Reducen la varianza.
- Disminuyen el efecto de la correlación entre predictores.
- Reducen la influencia de los predictores menos relevantes.

Aprendizaje supervisado

Regresión Lineal – Regularización – Regresión Ridge

Norma L2: $||\omega||_2^2 = \sum_i^n \omega_i^2$

La **Regularización Ridge** es también denominada regresión L2.

$$\text{mín } ||y - X\beta||_2^2 + \lambda ||\beta||_2^2$$

Características:

- Penaliza la suma de los coeficientes elevados al cuadrado.
- Reduce el valor de los coeficientes del modelo sin que lleguen a cero.
- Ventaja: reducción de varianza. (Empleando un λ adecuado, Ridge consigue reducir la varianza sin a penas aumentar el sesgo).
- Desventaja: el modelo final incluye todos los predictores.

Aprendizaje supervisado

Regresión Lineal – Regularización – Regresión Lasso

Norma L1: $||\omega||_1 = \sum_i^n |\omega_i|$

La **Regularización Lasso** es también denominada regresión L1.

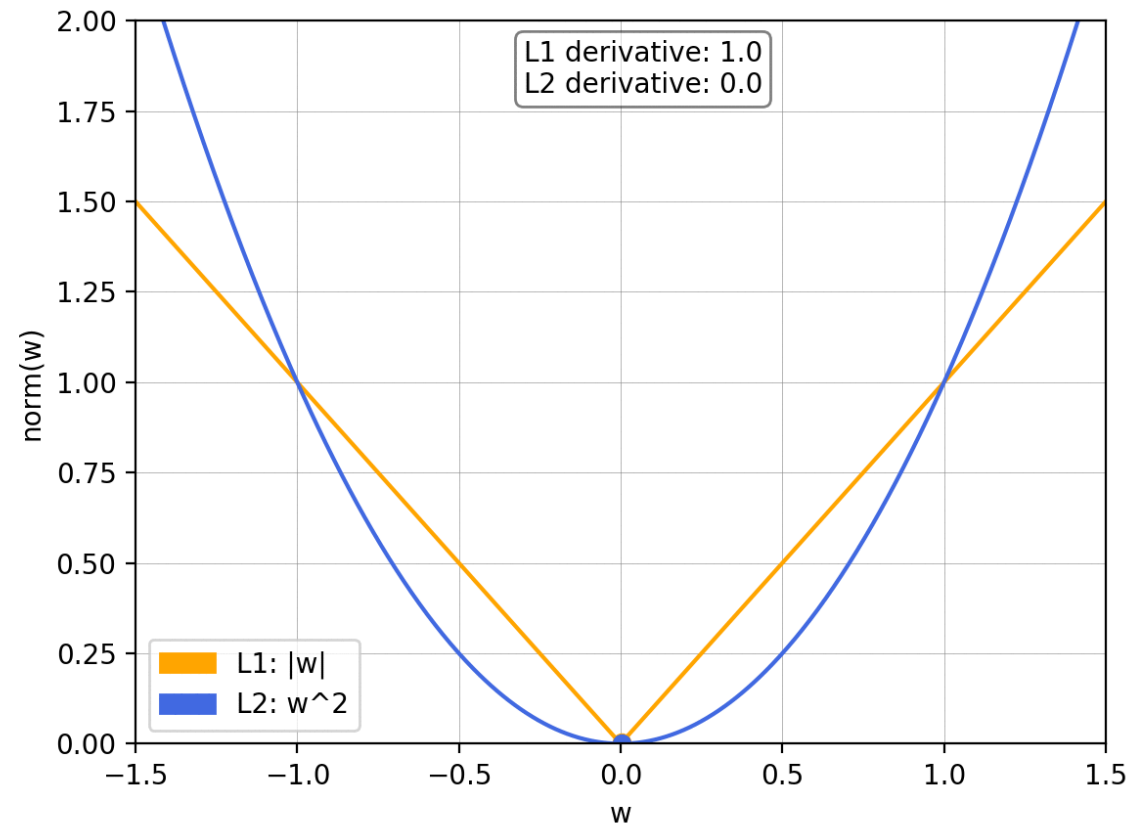
$$\min \frac{1}{2} ||y - X\beta||_2^2 + \lambda ||\beta||_1$$

Características:

- Penaliza la suma del valor absoluto de los coeficientes de regresión.
- Fuerza a que los coeficientes predictores tiendan a cero.
- Consigue excluir los predictores menos relevantes (cuando $\lambda = 0$ el resultado es equivalente al modelo lineal por mínimos cuadrados ordinarios, a medida que λ aumenta, la penalización es mayor y más predictores quedan excluidos).

Aprendizaje supervisado

Regresión Lineal – Regularización



Aprendizaje supervisado

Regresión Lineal – Regularización – Elastic Net

La **Regularización Elastic Net** combina la norma L1 y L2.

$$\min_{\beta} ||y - X\beta||_2^2 + \alpha\lambda||\beta||_1 + \frac{1}{2}\alpha(1 - \lambda)||\beta||_2^2$$

El grado en el que influye cada una de las penalizaciones está controlado por el hiperparámetro α , cuyo valor está comprendido entre $[0,1]$.

- Si $\alpha=0$, se aplicar Ridge.
- Si $\alpha=1$, se aplica Lasso.

Aprendizaje supervisado

Regresión Lineal en Python

Para calcular un modelo de regresión lineal en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)

>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Aprendizaje supervisado

Regresión Lineal en Python

Para calcular un modelo de regresión lineal en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Aprendizaje supervisado

Regresión Lineal en Python

Para calcular un modelo de regresión lineal en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)
>>> reg.coef_
array([1., 2.])
>>> reg.intercept_
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Coeficientes
de la recta de
regresión

Aprendizaje supervisado

Regresión Lineal en Python

Para calcular un modelo de regresión lineal en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> import numpy as np
>>> from sklearn.linear_model import LinearRegression
>>> X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
>>> y = np.dot(X, np.array([1, 2])) + 3
>>> reg = LinearRegression().fit(X, y)

>>> reg.coef_
array([1., 2.])
>>> reg.intercept
3.0...
>>> reg.predict(np.array([[3, 5]]))
array([16.])
```

Término independiente de la recta de regresión.

Ejemplo en Python



IntroduccionML_RegresionLineal.ipnyb



Aprendizaje supervisado

Regresión logística

Aprendizaje supervisado

Regresión Logística

El modelo de **regresión logística** resuelve un problema de **clasificación**, ya que ahora nuestra variable respuesta es cualitativa (categórica).

Si la variable dependiente tiene dos categorías, se trata de un modelo de regresión logística **binaria**, mientras que, si tiene más de dos categorías, será un modelo **multinomial**.

El modelo de regresión logística es:

$$\Pr(y = 1|x) = \frac{\exp(b_0 + \sum_{i=1}^n b_i x_i)}{1 + \exp(b_0 + \sum_{i=1}^n b_i x_i)}$$

donde, $\Pr(y=1 | x)$ es la probabilidad de que y tome el valor 1 dados los valores de las covariables $X = (x_1, x_2, \dots, x_n)$, b_0 es la constante del modelo y b_i los pesos asociados a cada una de las covariable x_i .

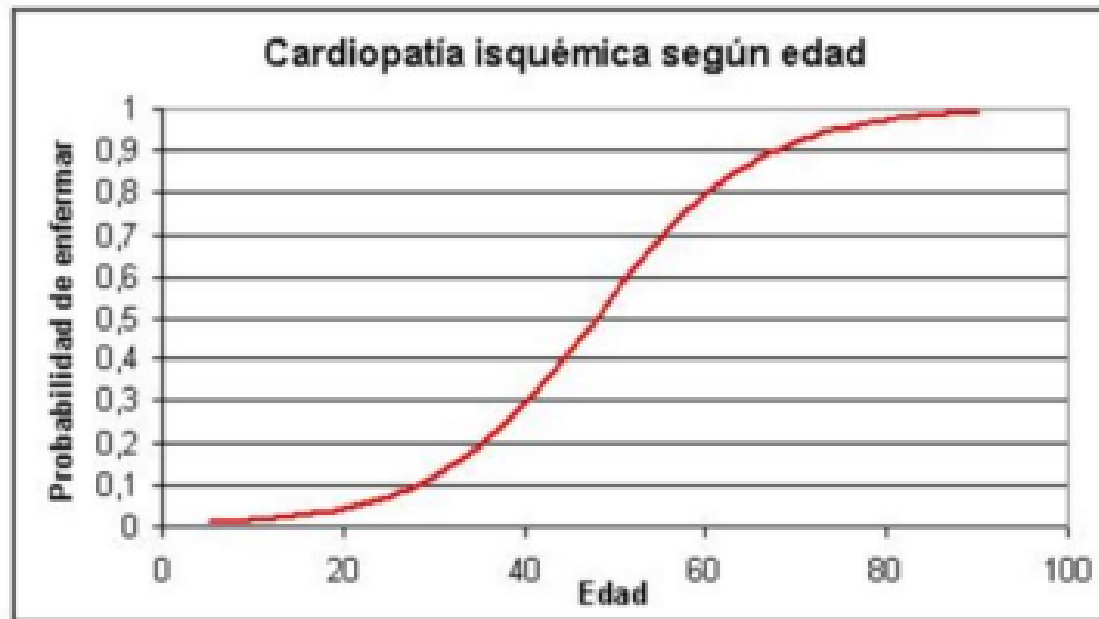
Aprendizaje supervisado

Regresión Logística

El modelo de **regresión logística** resuelve un problema de **clasificación**, ya que ahora nuestra variable respuesta es cualitativa (categórica).

Si la variable de regresión logística es **binaria**, **multinomial**.

El modelo de regresión



modelo de regresión
ías, será un modelo

donde, $\Pr(y=1 | x)$ dados los valores de las
covariables $X = (x_1, x_2, \dots, x_n)$, b_0 es la constante del modelo y b_i los pesos asociados a
cada una de las covariable x_i .

Aprendizaje supervisado

Regresión Logística

Dividiendo la expresión anterior por su complementario, obtenemos Odds. La expresión es más manejable:

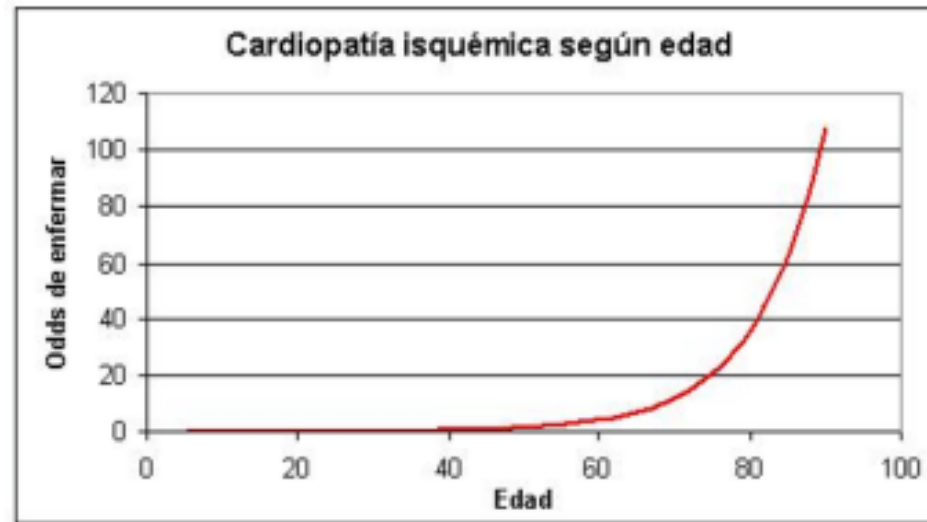
$$\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} = \exp(b_0 + \sum_{i=1}^n b_i x_i)$$

Aprendizaje supervisado

Regresión Logística

Dividiendo la expresión anterior por su complementario, obtenemos Odds. La expresión es más manejable:

$$\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} = \exp(b_0 + \sum_{i=1}^n b_i x_i)$$



Aprendizaje supervisado

Regresión Logística

Dividiendo la expresión anterior por su complementario, obtenemos Odds. La expresión es más manejable:

$$\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} = \exp(b_0 + \sum_{i=1}^n b_i x_i)$$

Explicación Odds:

Supóngase que la probabilidad de ser verdadero es $p=0,8$. Al tratarse de una variable binaria, la probabilidad de ser falso es $q=1-0,8=0,2$. Los odds se definen como el ratio entre la probabilidad de evento verdadero y probabilidad de evento falso $\frac{p}{q}$. En nuestro caso, los odds son $\frac{0,8}{0,2} = 4$, lo que quiere decir que se esperan 4 eventos verdaderos por cada evento falso.

Al estar las probabilidades acotadas entre $[0,1]$, los odds están acotados entre $[0, \infty]$.

Aprendizaje supervisado

Regresión Logística

Dividiendo la expresión anterior por su complementario, obtenemos Odds. La expresión es más manejable:

$$\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} = \exp(b_0 + \sum_{i=1}^n b_i x_i)$$

Aplicando el logaritmo, se obtiene una ecuación lineal, que corresponde al logaritmo de la razón de proporciones:

$$\log\left(\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)}\right) = b_0 + \sum_{i=1}^n b_i x_i \Rightarrow \log\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \beta_i x_i$$

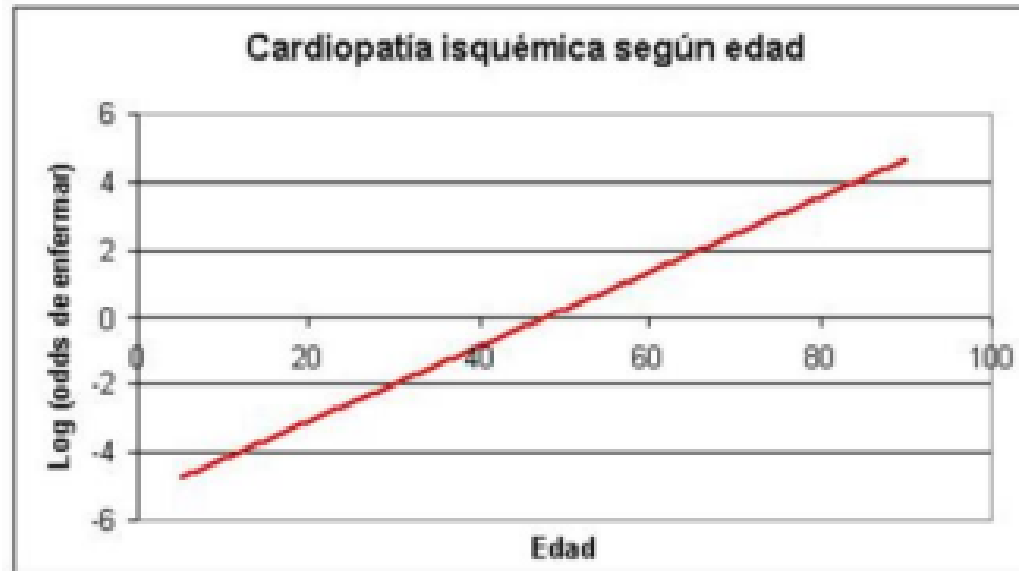
$\underbrace{\hspace{10em}}_{\Pr(y = 0|x)}$

Aprendizaje supervisado

Regresión Logística

Dividiendo la expresión es más manejable:

Aplicando el logaritmo de la razón de propo



os Odds. La expresión

esponde al logaritmo

$$\log\left(\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)}\right) = b_0 + \sum_{i=1}^n b_i x_i \Rightarrow \log\left(\frac{p_i}{1 - p_i}\right) = \beta_0 + \beta_i x_i$$

Aprendizaje supervisado

Regresión Logística

La probabilidad de que $y=1$ dados los predictores $X = x_1, \dots, x_n$.

$$\Pr(y = 1|x) = \frac{\exp(b_0 + \sum_{i=1}^n b_i x_i)}{1 + \exp(b_0 + \sum_{i=1}^n b_i x_i)}$$

Al ser un caso de probabilidad binaria:

$$\Pr(y = 0|x) = 1 - \Pr(y = 1|x) = 1 - \frac{\exp(b_0 + \sum_{i=1}^n b_i x_i)}{1 + \exp(b_0 + \sum_{i=1}^n b_i x_i)} = \frac{1}{1 + \exp(b_0 + \sum_{i=1}^n b_i x_i)}$$

Luego,

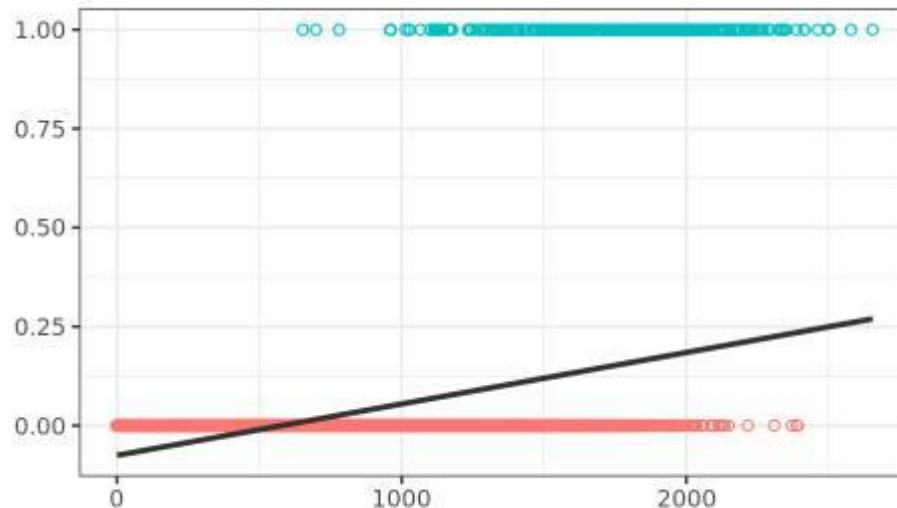
$$\log\left(\frac{\Pr(y = 1|x)}{\Pr(y = 0|x)}\right) = \log\left(\frac{\frac{\exp(b_0 + \sum_{i=1}^n b_i x_i)}{1 + \exp(b_0 + \sum_{i=1}^n b_i x_i)}}{\frac{1}{1 + \exp(b_0 + \sum_{i=1}^n b_i x_i)}}\right) = \log(\exp(b_0 + \sum_{i=1}^n b_i x_i)) = b_0 + \sum_{i=1}^n b_i x_i$$

Aprendizaje supervisado

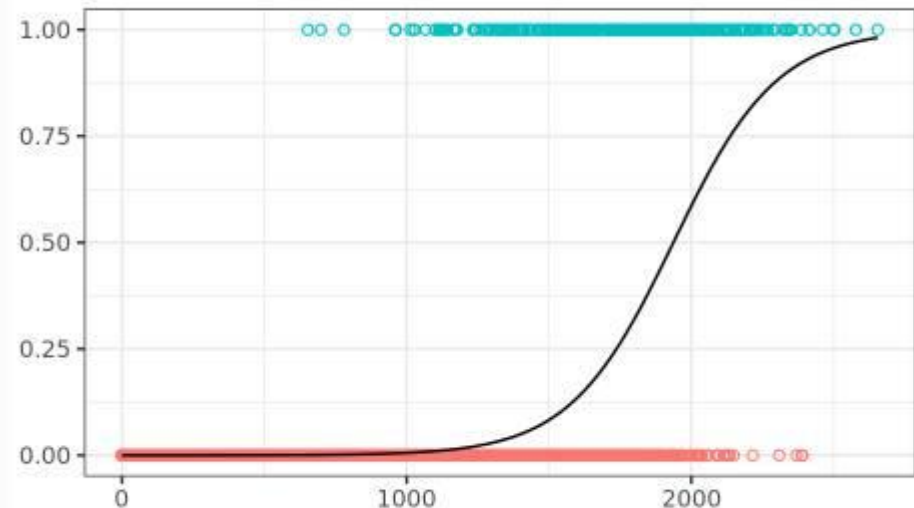
Regresión Logística

Hemos conseguido convertir un problema de clasificación no lineal en un problema de regresión lineal.

Regresión lineal por mínimos cuadrados



Regresión logística



Aprendizaje supervisado

Regresión Logística en Python

Para calcular un modelo de regresión logística en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
```

Aprendizaje supervisado

Regresión Logística en Python

Para calcular un modelo de regresión logística en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
```

Predecir la
clase de la
observación.

Aprendizaje supervisado

Regresión Logística en Python

Para calcular un modelo de regresión logística en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.8...e-01, 1.8...e-02, 1.4...e-08],
       [9.7...e-01, 2.8...e-02, ...e-08]])
```

Probabilidad de la clase de la observación.

Ejemplo en Python



IntroduccionML_RegresionLogistica.ipnyb

Aprendizaje supervisado

K-vecinos

Aprendizaje supervisado

K-vecinos

El modelo *k-nearest-neighbor*, *kNN*, puede usarse para resolver problemas de **regresión** y de **clasificación**.

Se trata de un modelo basado en **instancias**. Quiere decir que nuestro algoritmo no aprende explícitamente un modelo, memoriza las instancias de entrenamiento y las usa para la fase de predicción.

Pasos kNN:

- 1.- Calcular la distancia entre el ítem y el resto de ítems del set de datos de entrenamiento.
- 2.- Seleccionar los k elementos más cercanos, haciendo uso de la función de distancias deseada.
- 3.- Realizar una “votación de mayoría” entre los k puntos. El valor de k será muy importante para definir a qué grupo pertenecen los puntos.

Aprendizaje supervisado

K-vecinos - matemáticamente

El modelo *k-nearest-neighbor*, consiste en memorizar una serie de instancias para después aplicarla a la predicción.

Se tiene m pares memorizados del tipo (x_i, y_i) , donde el valor a asociar para un punto x sería:

$$\hat{y}(x) = \frac{1}{k} \sum_{j=1}^k y(x_j)$$

con x_j , los puntos cuya distancia sea la menor posible a x y con $y(x_j)$ igual al valor de la variable explicada en dicho punto x_j .

Aprendizaje supervisado

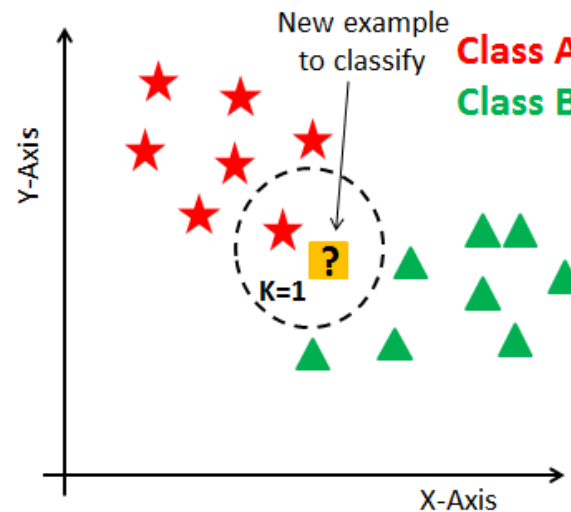
K-vecinos

VENTAJAS

- Simplicidad
- Efectividad
- Fácil implementación

INCONVENIENTES

No se crea un modelo, sino que se genera un conjunto de instancias sobre las que se toman distancias. Esto genera un rápido proceso de entrenamiento, pero un lento proceso de predicción.



Aprendizaje supervisado

K-vecinos en Python

Para calcular un modelo de K-vecinos en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
```

Aprendizaje supervisado

K-vecinos en Python

Para calcular un modelo de K-vecinos en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
```

Número de vecinos

Aprendizaje supervisado

K-vecinos en Python

Para calcular un modelo de K-vecinos en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
```

Predecir la
clase de la
observación.

Ejemplo en Python



IntroduccionML_KNN.ipnyb

Aprendizaje supervisado

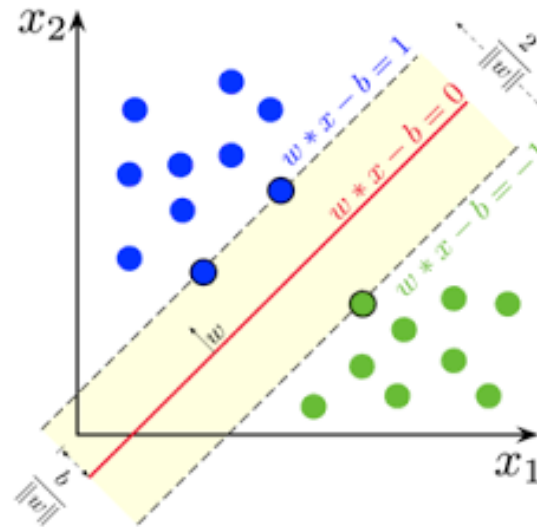
SVM

Aprendizaje supervisado

SVM

Algoritmo de aprendizaje supervisado que se utiliza en problemas de **regresión** y **clasificación**.

El objetivo es encontrar un hiperplano que separe de la mejor forma posible dos clases diferentes de datos. El margen se define como la anchura máxima de la región paralela al hiperplano que no tienen puntos de datos interiores. El algoritmo solo puede encontrar este hiperplano en problemas que permiten separación lineal. En la mayoría de casos prácticos, el algoritmo maximiza el margen flexible permitiendo un pequeño número de clasificaciones erróneas.



Aprendizaje supervisado

SVM - matemáticamente

Problema binario a resolver:

$$S = \{(x^p, y^p), 1 \leq p \leq N\}$$

Con p-dimensiones, x^p son los patrones e $y^p = \pm 1$, buscamos el hiperplano de dimensión p-1.

Asumimos que S es linealmente separable, donde existen ω, b :

$$\begin{aligned} \omega \cdot x^p + b &> 0 \text{ si } y^p = 1 \\ \omega \cdot x^p + b &< 0 \text{ si } y^p = -1 \end{aligned}$$

Es decir, buscamos $y^p(\omega \cdot x^p + b) > 0$.

Aprendizaje supervisado

SVM en Python

Para calcular un modelo de SVM en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
SVC()
>>> clf.predict([[2., 2.]])
array([1])
```

Aprendizaje supervisado

SVM en Python

Para calcular un modelo de SVM en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn import svm
>>> X = [[0, 0], [1, 1]]
>>> y = [0, 1]
>>> clf = svm.SVC()
>>> clf.fit(X, y)
SVC()
>>> clf.predict([2., 2.])
array([1])
```

Predecir la
clase de la
observación.

Ejemplo en Python



IntroduccionML_SVM.ipnyb



Aprendizaje supervisado

Árboles de decisión

Aprendizaje supervisado

Árboles de decisión

Los árboles de decisión resuelven problemas de **regresión** y de **clasificación** haciendo uso de un conjunto de reglas de división utilizadas para segmentar el espacio de las variables predictoras.

Ventajas:

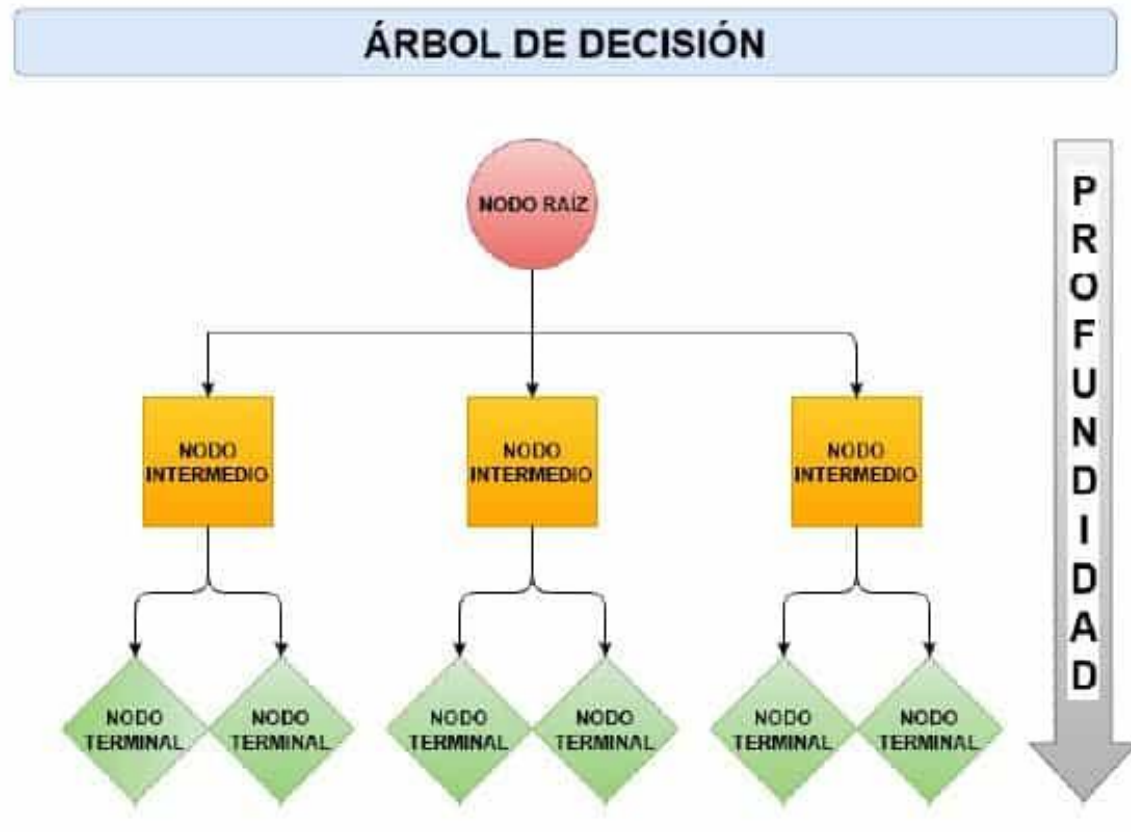
- Transparencia: fácil de interpretar.
- Portabilidad: las pautas que se extraen del camino a una hoja del árbol se expresan fácilmente en distintos formatos.
- Modelización: pueden utilizar tanto variables continuas como categóricas.
- No se ven muy influenciados por outliers.

Desventajas:

- Se debe utilizar un gran volumen de datos para asegurarnos que la cantidad de casos en un nodo terminal es significativa.
- Son sensibles a datos de entrenamiento desbalanceados.

Aprendizaje supervisado

Árboles de decisión - terminología



Aprendizaje supervisado

Árboles de decisión - Parámetros

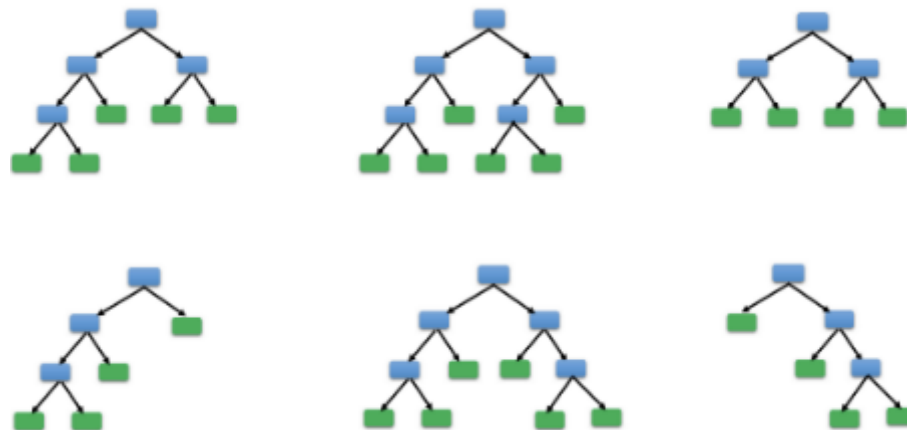
- **Función de error:** mide la calidad de una división.
- **Profundidad:** número de niveles de mayor orden que se le va a permitir crecer al árbol.
- **Cantidad mínima para subdividir:** permite controlar si hay suficiente muestra para hacer una subdivisión o parar el algoritmo tenemos menos cantidad que la mínima establecida.
- **Mínima muestra por hoja:** controla si la cantidad de datos en cada hoja final es representativa.
- **Máximo número de nodos.**

Aprendizaje supervisado

Árboles de decisión

¿Cómo mejorar los resultados en árboles de decisión?

- **Prepoda:** decidimos cuándo queremos parar de construir el árbol (observaciones mínimas de nodo terminal, profundidad máxima del árbol, número máximo de nodos terminales, etc.).
- **Poda:** dejamos crecer el árbol, detectamos qué ramas generan overfitting y las eliminamos.
- **Bagging:** en lugar de ajustar un único árbol, se ajustan mucho en paralelo formando un “bosque”. Como valor se toma la media de las predicciones (variables continuas) o la clase más frecuente (variables cualitativas). Uno de los métodos de bagging más conocidos es Random Forest.



Aprendizaje supervisado

Árboles de decisión en Python

Trabajar con árboles de decisión en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

Aprendizaje supervisado

Árboles de decisión en Python

Trabajar con árboles de decisión en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

Si buscamos un árbol de regresión, utilizamos la función DecisionTreeRegressor

Ejemplo en Python



`IntroduccionML_ArbolesDecision.ipnyb`

Aprendizaje supervisado

Ensembles

Aprendizaje supervisado

Ensembles

¿Qué ocurre cuando no obtenemos resultados buenos con nuestros modelos?

Una posible solución puede ser la combinación de modelos para obtener uno que pueda ofrecer un mejor rendimiento.



Aprendizaje supervisado

Ensembles

Un **ensemble** (o conjunto de clasificadores) es una agrupación de varios modelos de aprendizaje que se construyen para resolver el mismo problema de aprendizaje automático y toman decisiones en conjunto para realizar predicciones.

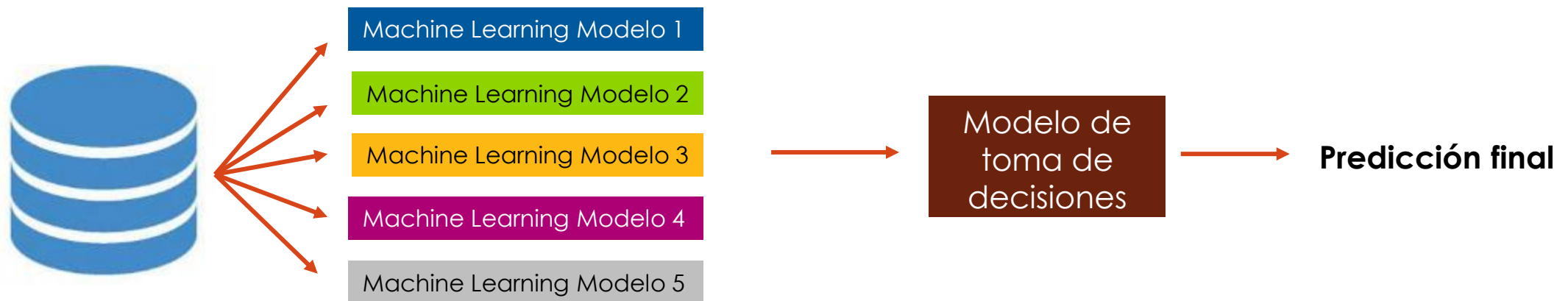
Construir un ensemble implica entrenar una serie de modelos base y luego combinar sus predicciones de alguna manera.

Los modelos base de un ensemble se construyen de manera que se maximice la combinación de dos métricas:

- La **precisión** de la predicción combinada de un ensemble depende de la precisión de cada uno de los modelos base.
- Si todos los modelos base son idénticos o muy parecido, combinar sus predicciones no aporta nada, por lo que buscamos **diversidad** de los modelos base.

Aprendizaje supervisado

Ensembles





Aprendizaje supervisado

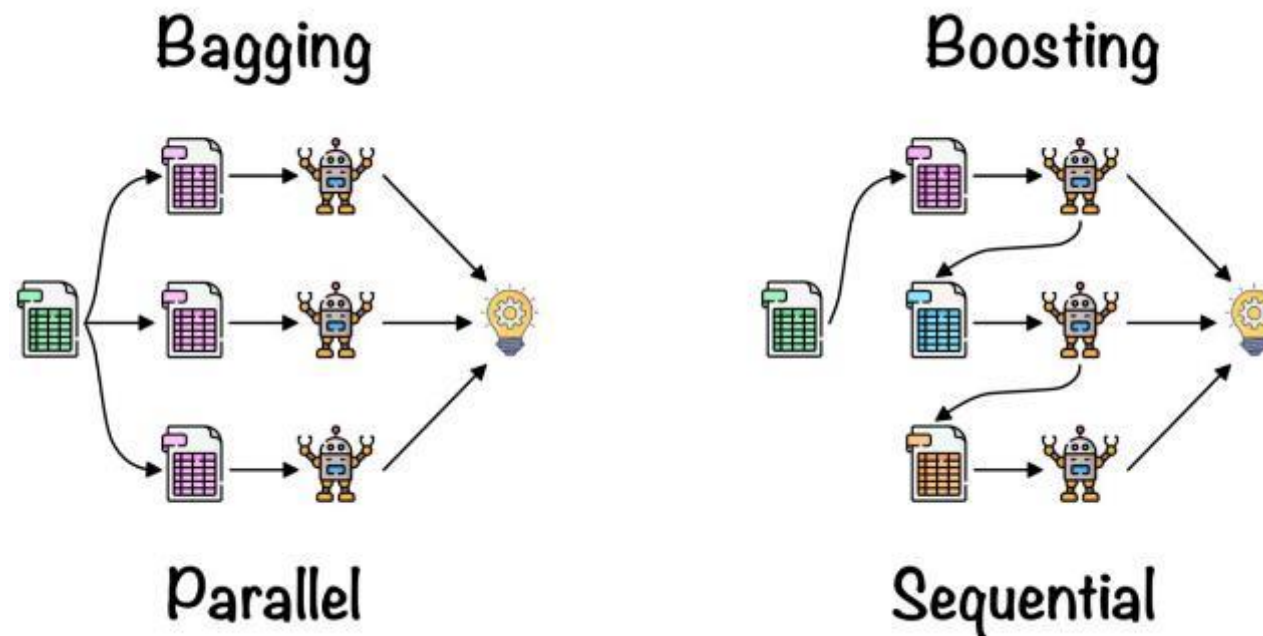
Bagging y Boosting

Aprendizaje supervisado

Bagging y Boosting

Tanto el **Bagging** como el **Boosting** son procedimientos generales para la reducción de la varianza.

La idea básica consiste en combinar modelos de predicción sencillos, con poca capacidad predictiva, para obtener un método de predicción muy potente y robusto. Esta idea se puede aplicar a problemas de **regresión** y de **clasificación**.



Aprendizaje supervisado

Bagging

Uno de los primeros métodos de *ensembles* (métodos combinados) que se empezó a utilizar fue **Bagging**, propuesto por Breiman en 1996.

Se basa en utilizar Bootstrap junto con un modelo de regresión o de clasificación.

Aprendizaje supervisado

Bagging

Uno de los primeros métodos de *ensembles* (métodos) que se empezó a utilizar fue **Bagging**, propuesto por Breiman.

Se basa en utilizar Bootstrap para crear un modelo de regresión o de clasificación.

¿QUÉ ES BOOTSTRAP?

Aprendizaje supervisado

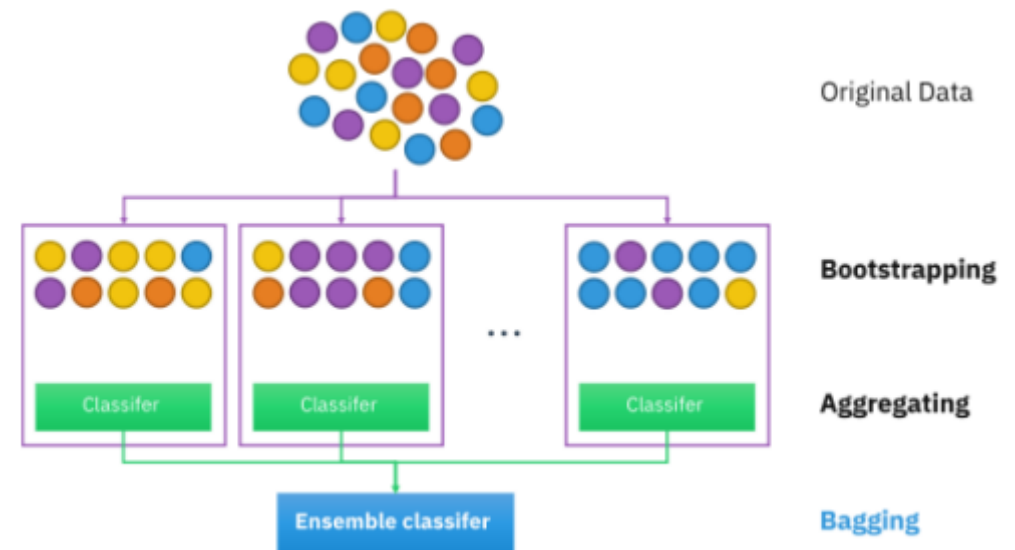
Bagging

Uno de los primeros métodos de *ensembles* (métodos) que se empezó a utilizar fue **Bagging**, propuesto por Breiman.

Se basa en utilizar Bootstrap para crear un modelo de regresión o de clasificación.

¿QUÉ ES BOOTSTRAP?

Bootstrap es una técnica de remuestreo conocida como empaquetado, diseñado para mejorar la estabilidad y precisión de algoritmos de aprendizaje automático. Reduce la varianza y evita el sobreajuste. Implica la extracción de datos de muestra repetidamente con reemplazo de una fuente de datos para estimar un parámetro de población.



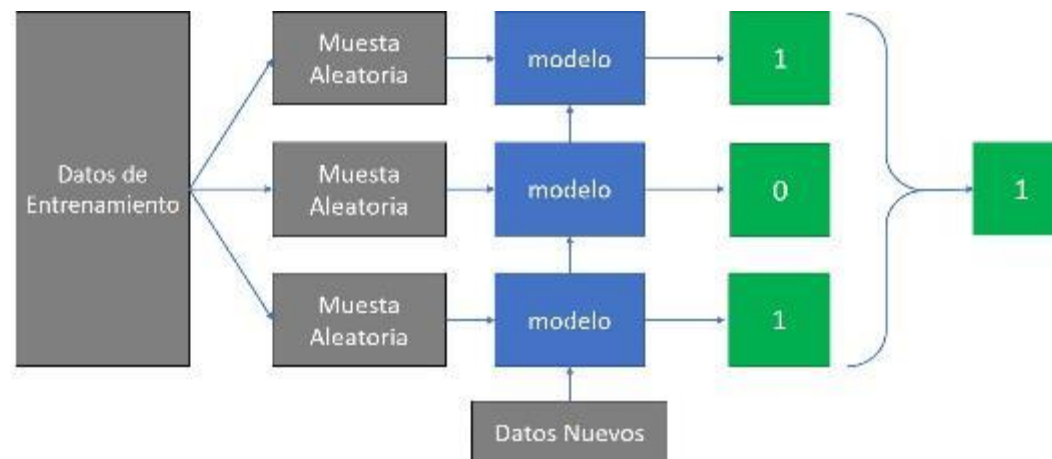
Aprendizaje supervisado

Bagging

Uno de los primeros métodos de *ensembles* (métodos combinados) que se empezó a utilizar fue **Bagging**, propuesto por Breiman en 1996.

Se basa en utilizar Bootstrap junto con un modelo de regresión o de clasificación.

Idea intuitiva: Dado el conjunto de entrenamiento, se generan muestras Bootstrap. Se utiliza cada una de ellas como muestra de entrenamiento, generando una predicción para cada una. De esta forma, tenemos tantas predicciones como muestras de entrenamiento. Se promedian las predicciones de los modelos (en problemas de clasificación será la clase mayoritaria).



Aprendizaje supervisado

Boosting

El algoritmo **Boosting** intenta mejorar las capacidades de predicción entrenando varios modelos sencillos. Es muy utilizado, ya que reduce sesgo y varianza.

Boosting surgió de la pregunta de Kearns y Valiant (1988, 1989): ¿pueden un conjunto de métodos con poca capacidad predictiva crear un clasificador robusto? La implementación efectiva se realizó en 1996 (Freund y Schapire) con el algoritmo AdaBoost.

En boosting, cada modelo intenta arreglar los errores de los modelos anteriores. Para conseguirlo, se da más peso a las muestras mal clasificadas y menos peso a las muestras bien clasificadas. De esta forma, cada modelo se centra en corregir los errores del modelo anterior.

Implementaciones de modelos que usan boosting: AdaBoost, XGBoost, CatBoost, LightGBM.

Aprendizaje supervisado

Random Forest

Aprendizaje supervisado

Random Forest

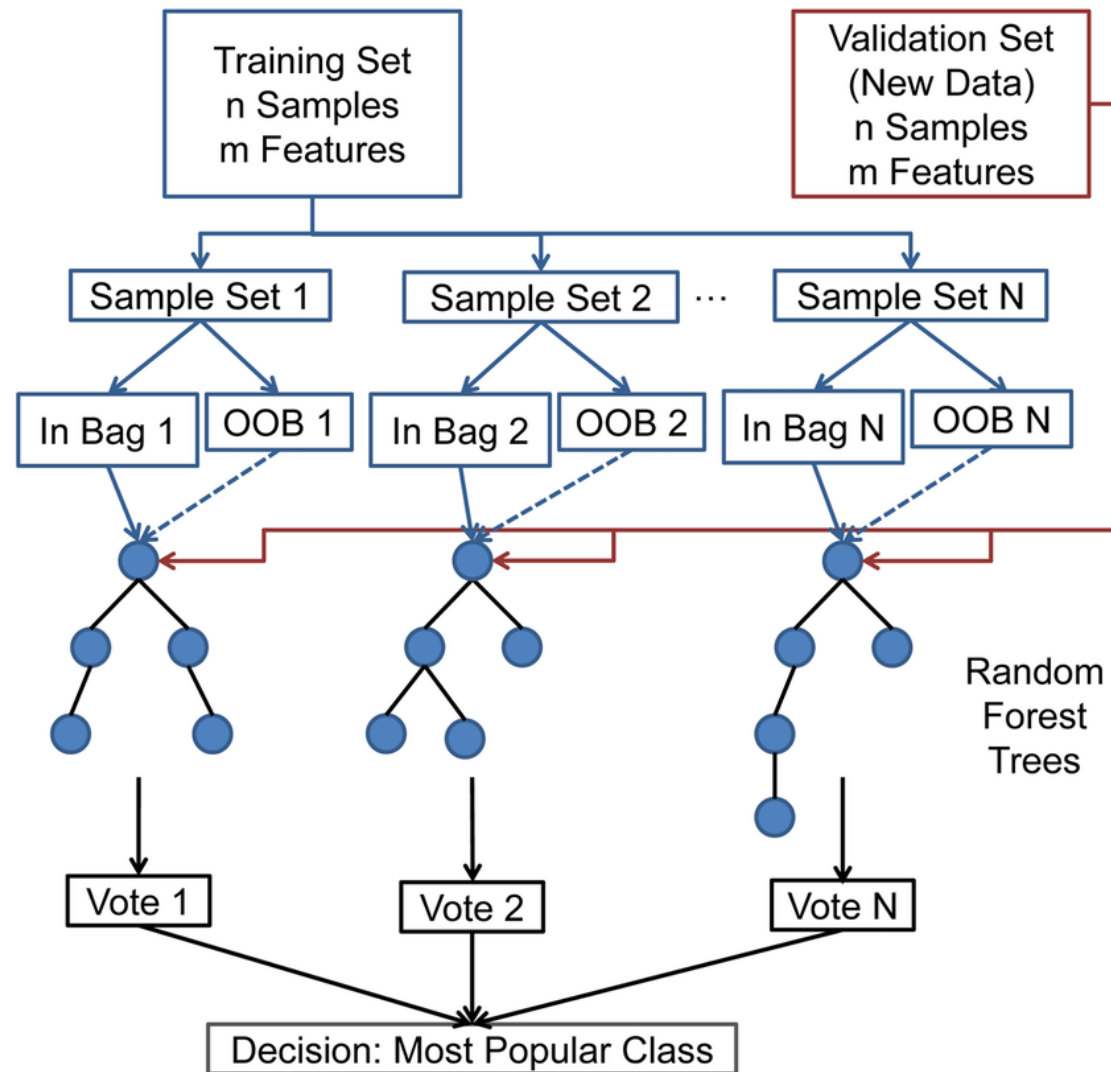
Un **Random Forest** es un conjunto de árboles de decisión combinados con bagging. Al igual que los árboles, resuelve problemas tanto de **regresión** como de **clasificación**.

Construcción Random Forest:

- Sea el número de casos en entrenamiento N . Se toma una muestra aleatoria de esos N casos con reemplazo. Esta muestra será el conjunto de entrenamiento para construir el árbol i .
- Si existen N variables de entrada, con $m > M$, se especifica tal que para cada nodo, m variables se seleccionan aleatoriamente de M . La mejor división de estos m atributos es usado para ramificar el árbol. El valor de m se mantiene durante la generación de todo el bosque.
- Cada árbol crece hasta su máxima extensión posible, sin poda.
- Las nuevas instancias se predicen a partir de la agregación de las predicciones de los x árboles.

Aprendizaje supervisado

Random Forest



Aprendizaje supervisado

Random Forest

VENTAJAS

- Es utilizado como método de reducción de dimensionalidad. Puede utilizar miles de variables de entrada e identificar las más significativas.
- Incorpora métodos para estimar valores faltantes.
- Relativamente rápido.

DESVENTAJAS

- Pérdida de interpretación.
- Poco control del modelo.
- Funcionan muy bien en problemas de clasificación, en regresión introducen ruido.

Aprendizaje supervisado

Random Forest en Python

Trabajar con Random Forest en Python, podemos usar la librería sklearn ([documentación](#)).

```
>>> from sklearn.ensemble import RandomForestClassifier
>>> from sklearn.datasets import make_classification
>>> X, y = make_classification(n_samples=1000, n_features=4,
...                           n_informative=2, n_redundant=0,
...                           random_state=0, shuffle=False)
>>> clf = RandomForestClassifier(max_depth=2, random_state=0)
>>> clf.fit(X, y)
RandomForestClassifier(...)
```

Ejemplo en Python



`IntroduccionML_RandomForest.ipnyb`

Aprendizaje supervisado

XGBoost

Aprendizaje supervisado



XGBoost

El **eXtreme Gradient Boost** se trata de una librería/algoritmo de moda en problemas supervisados.

Originalmente se desarrolló en C++, dados sus buenos resultados se ha ido desarrollando en los lenguajes más utilizados para el análisis de datos (Python, R, etc.).

Podemos consultar todos los detalles del algoritmo en la [documentación oficial](#).

Buscamos el mejor modelo que minimiza una función de error + regularización:

$$F^* = \arg \min_{F \in \mathcal{H}} \Phi(F) = \arg \min_{F \in \mathcal{H}} \mathbb{E}_{(x,y)} [L(y, F(x))] + \Omega(F)$$

con Ω alguna función que penaliza la complejidad del modelo F .

Aprendizaje supervisado

XGBoost

VENTAJAS

- Algoritmo paralelizable. Se pueden utilizar múltiples núcleos a la vez.
- Rapidez de entrenamiento.
- Permite entrenar modelos en set de datos de grandes tamaños.
- Mejora el rendimiento de la mayoría de algoritmos (es uno de los algoritmos preferidos en las competiciones de Machine Learning).

DESVENTAJAS

- Utiliza grandes cantidades de memoria RAM.
- Si se quiere trabajar con todas las variables, se requieren equipos de más de 8GB.
- Se debe ajustar correctamente los parámetros para minimizar el error de precisión.

Aprendizaje supervisado

Voting de modelos

Aprendizaje supervisado

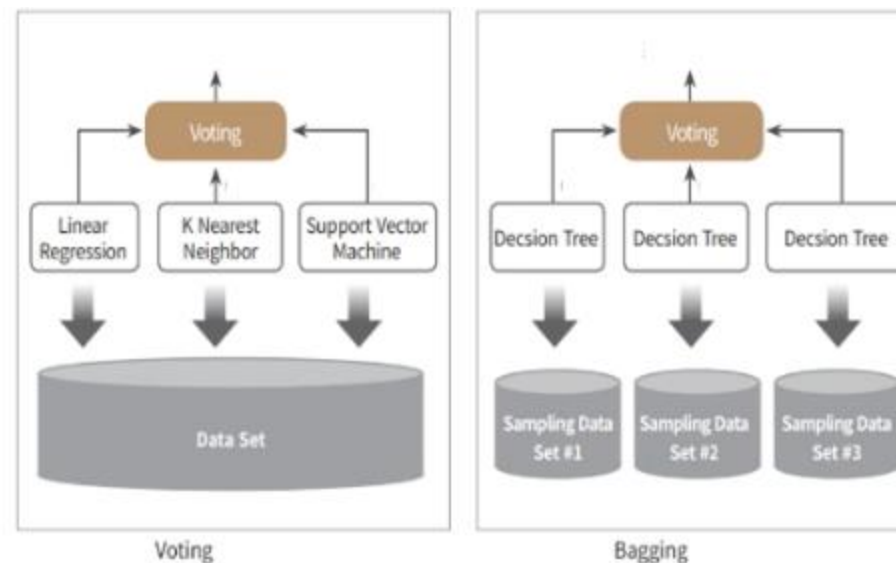
Voting classifier

Entre los métodos de ensemble más utilizados se encuentran los de votación mayoritaria. Donde cada uno de los clasificadores utilizados realiza una predicción independiente y se selecciona la seleccionada por la mayoría.

Entre los métodos de votación destacan dos familias:

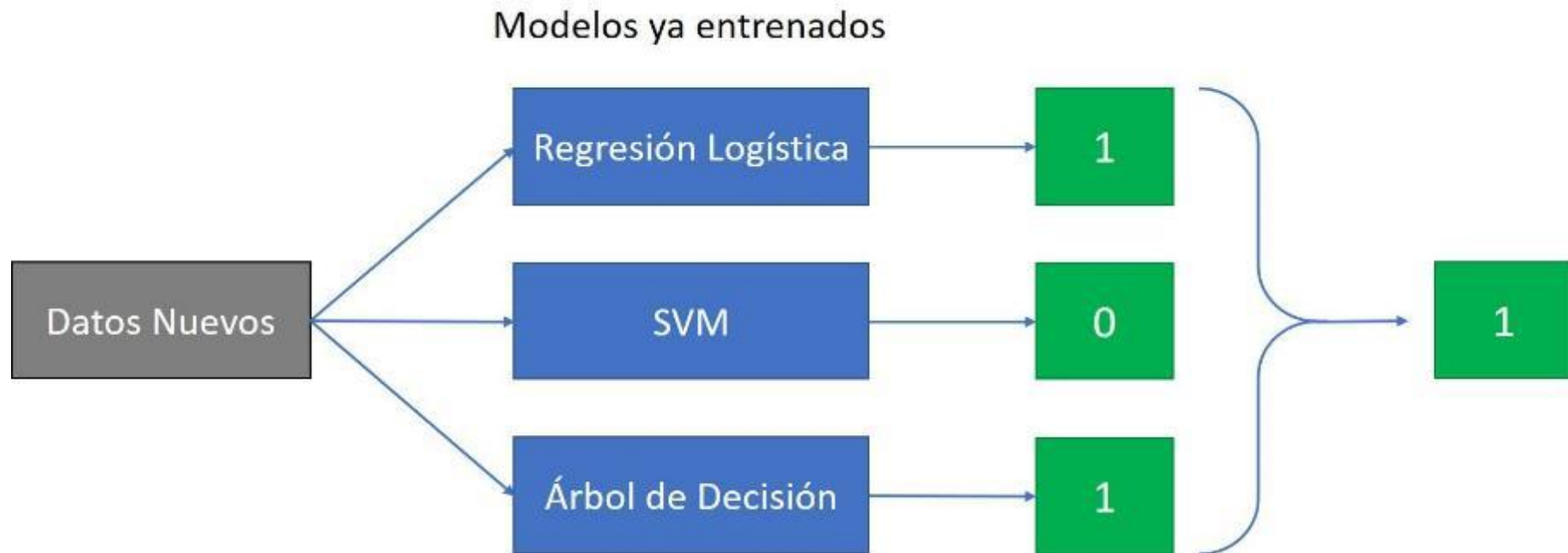
Voting: utiliza modelos de diferentes familias con el mismo conjunto de datos para obtener un meta clasificador.

Bagging: entrena la misma familia de modelos con conjuntos de datos diferentes esperando que cada uno se pueda especializar en diferentes datos.



Aprendizaje supervisado

Voting classifier - Ejemplo



Aprendizaje supervisado

Redes Neuronales

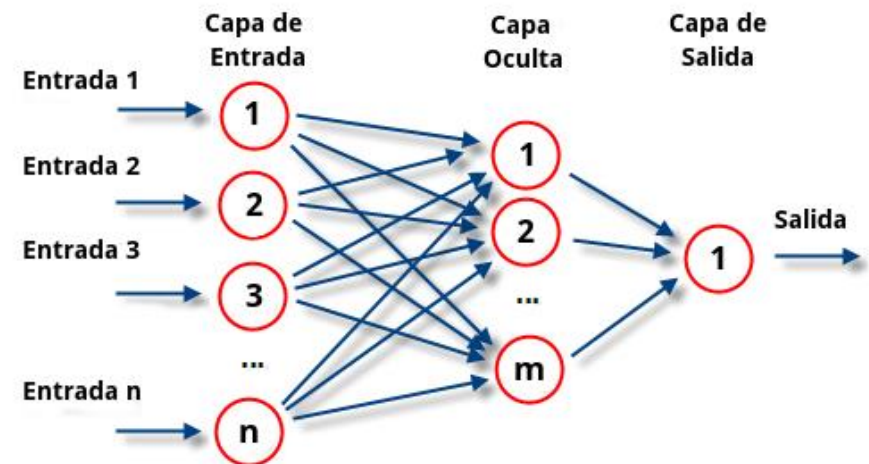
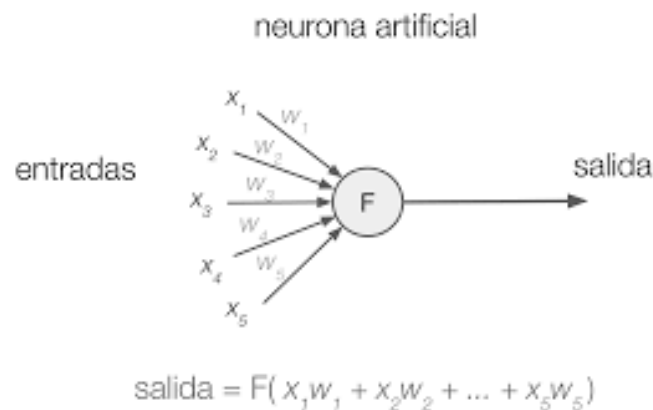
Aprendizaje supervisado

Redes Neuronales

Las redes neuronales forman parte del campo de la Inteligencia Artificial. Se inventaron en los años 60, pero no se usaron hasta más adelante debido a la capacidad limitada que tenían los ordenadores.

Durante los años 80, Geoffrey Hinton, investigó sobre el concepto Deep Learning investigando el cerebro humano e inspirándose en él para intentar reproducir de forma informática su comportamiento.

Por tanto, las neuronas artificiales se modelan imitando el comportamiento de una neurona cerebral.



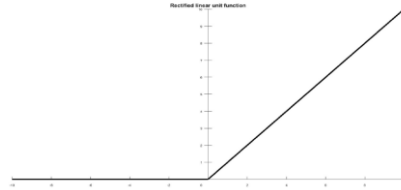
Aprendizaje supervisado

Redes Neuronales – funciones de activación

Las funciones de activación son la manera de transmitir la información por las conexiones de salida ([ejemplo](#)).

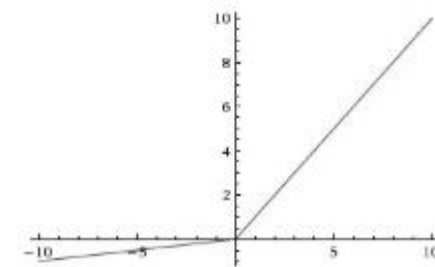
RELU

$$f(x) = \max(0, x)$$



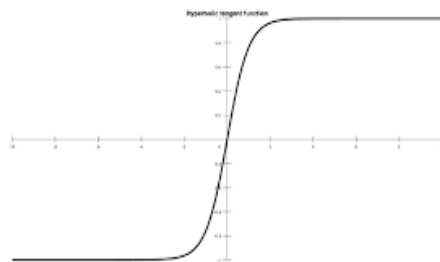
Leaky RELU

$$f(x) = \begin{cases} \alpha x, & x > 0 \\ x, & x \leq 0 \end{cases}$$



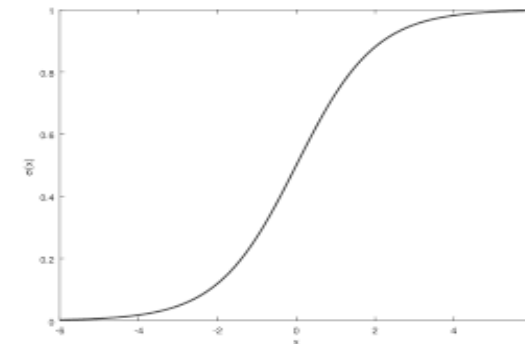
Tangente hiperbólica

$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$



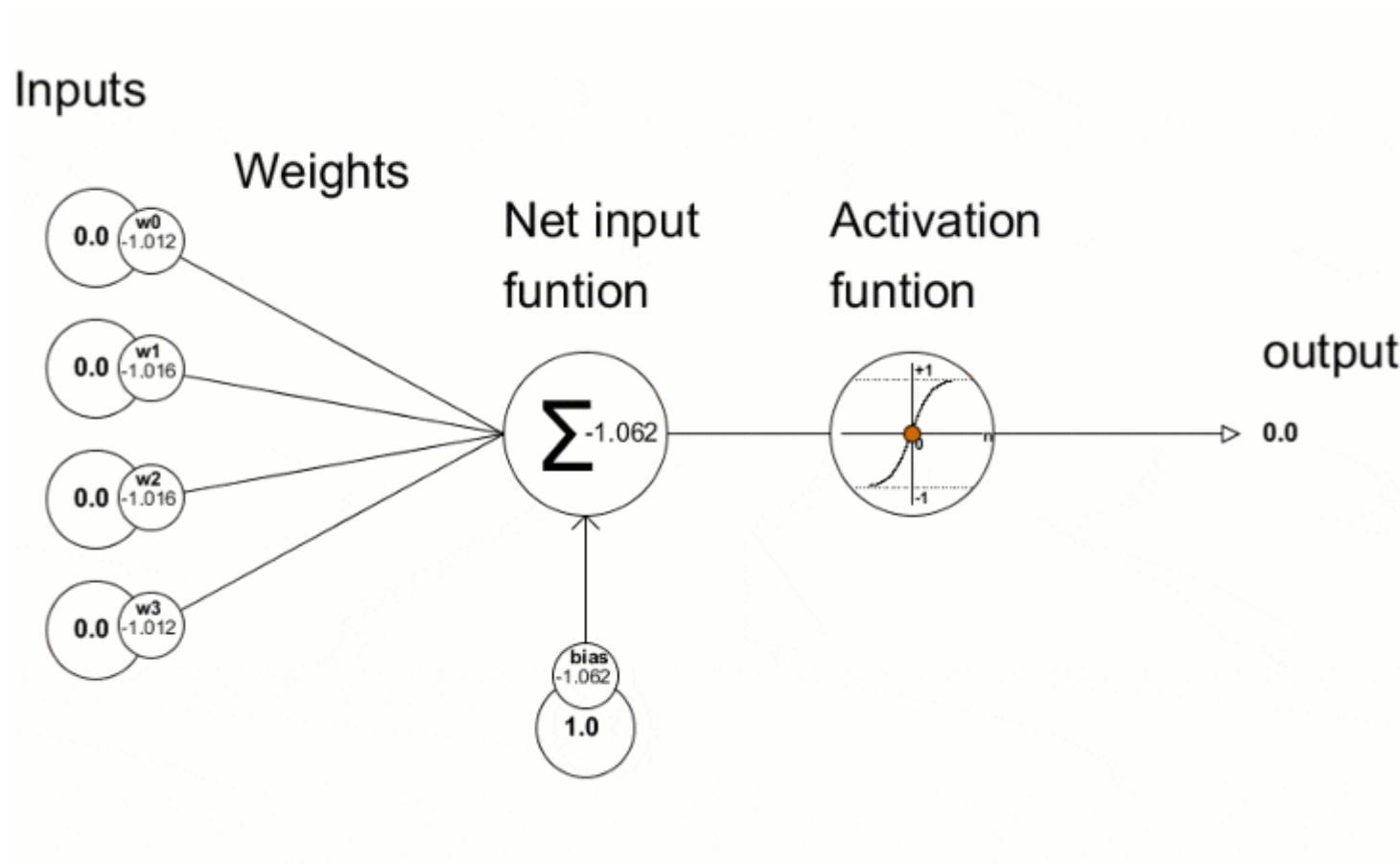
Sigmoide

$$f(x) = \frac{1}{1 + e^{-x}}$$



Aprendizaje supervisado

Redes Neuronales



Aprendizaje supervisado

Redes Neuronales en Python

SCIKIT-LEARN

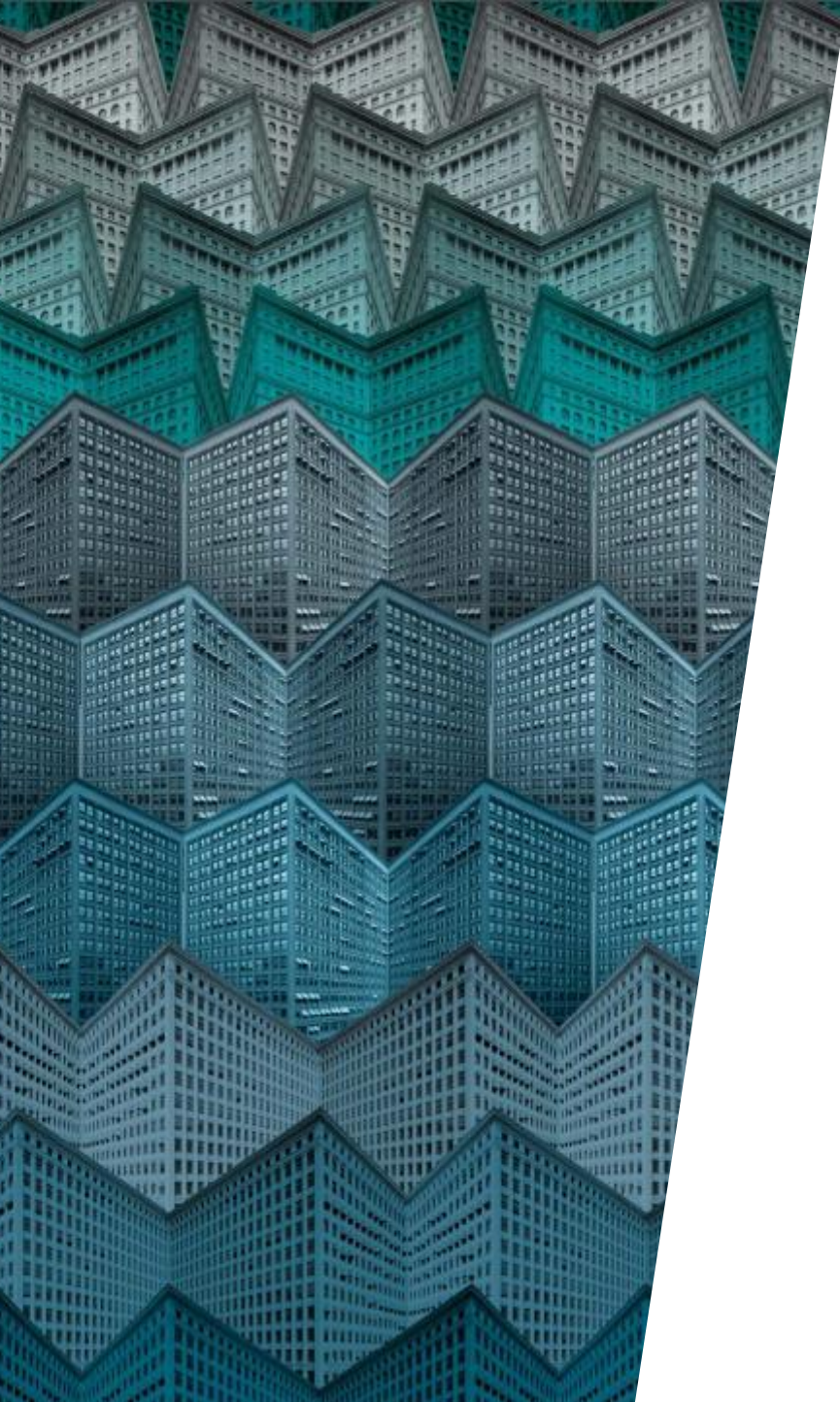


KERAS



TENSORFLOW

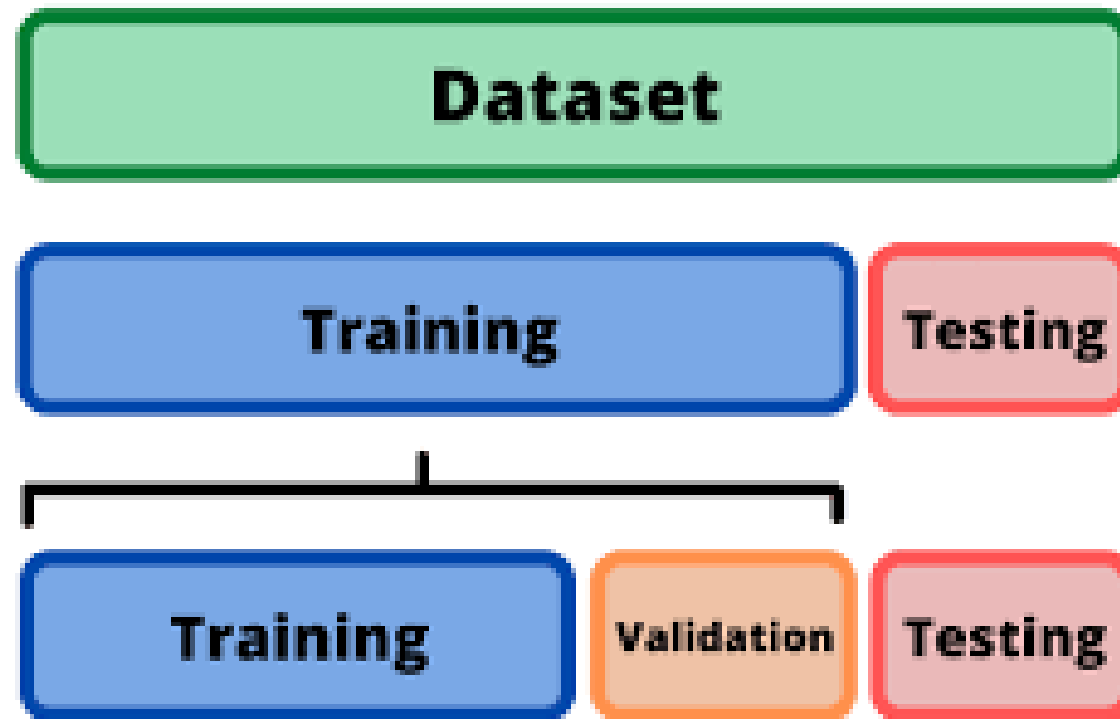




Metodología

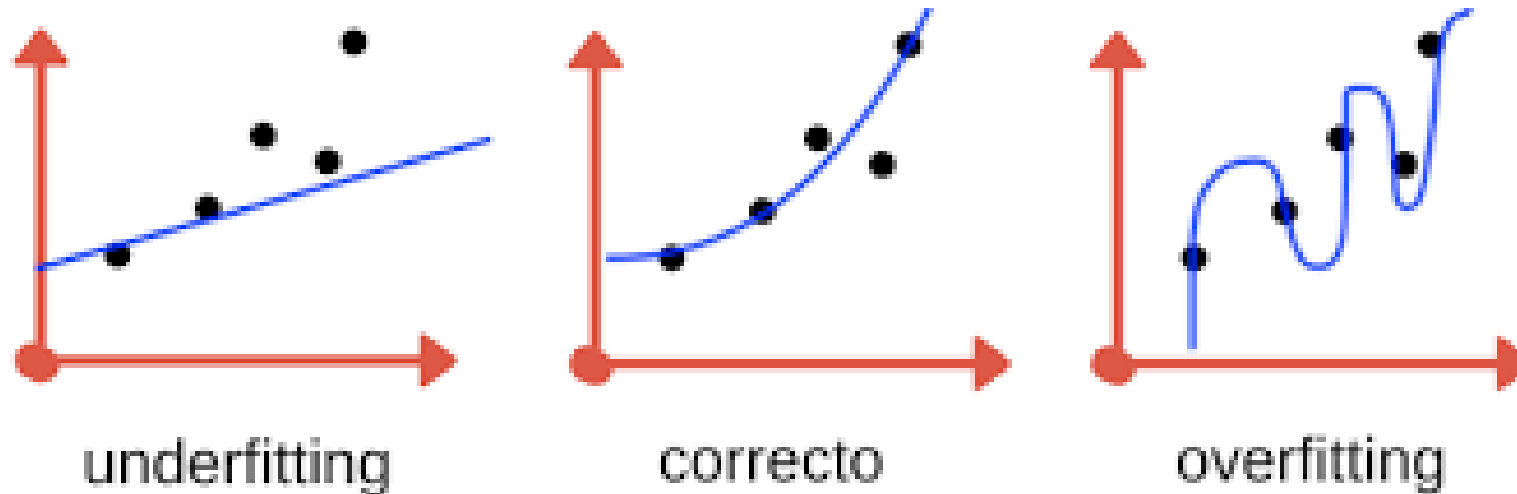
Metodología

Separación train-test-validación



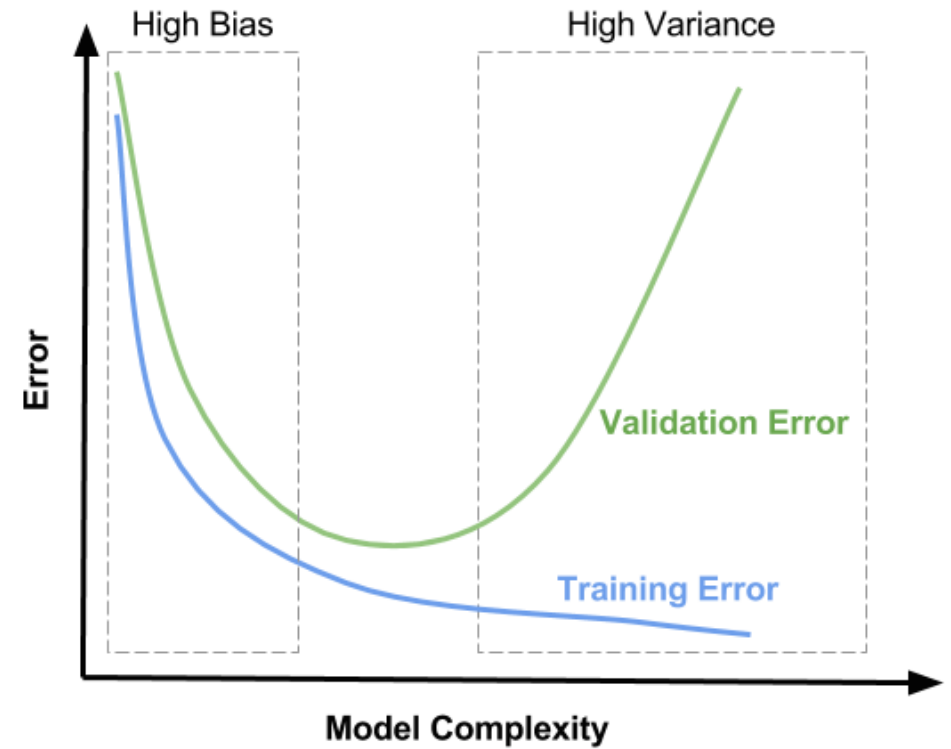
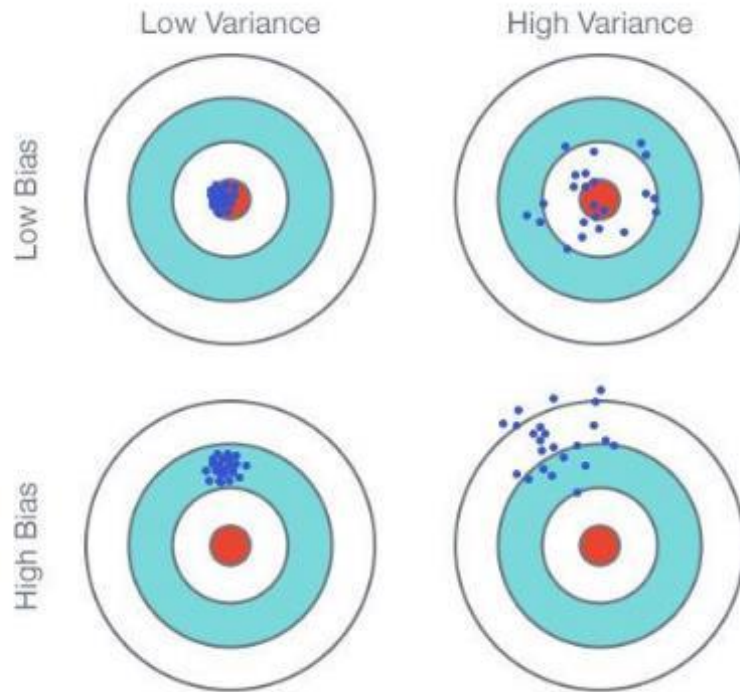
Metodología

Overfitting vs underfitting



Metodología

Sesgo y varianza

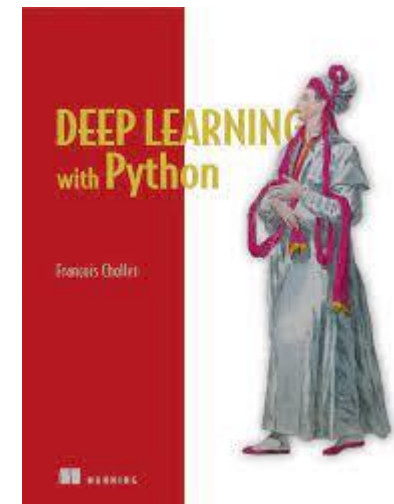
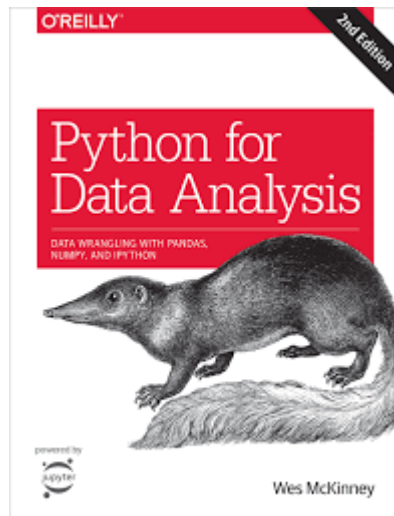




Referencias

Referencias

- Documentación oficial de Python: <https://docs.python.org/3/>
- Python for Data Analysis.
- Hands-on Machine Learning with Scikit-Learn, Keras & TensorFlow.
- Deep Learning with Python





Afi Escuela

© 2022 Afi Escuela. Todos los derechos reservados.