



Introducción a Data Science

Luciano Moliterno Gonzalez
luciano.moliterno.97@gmail.com

26 de febrero de 2025



Índice

1. Introducción
2. Data Science vs Big Data
3. ¿Qué es Python?
4. Proceso de Extracción
del Conocimiento
5. Preprocesamiento de datos
6. Modelos de Machine Learning
7. Metodología
8. Referencias



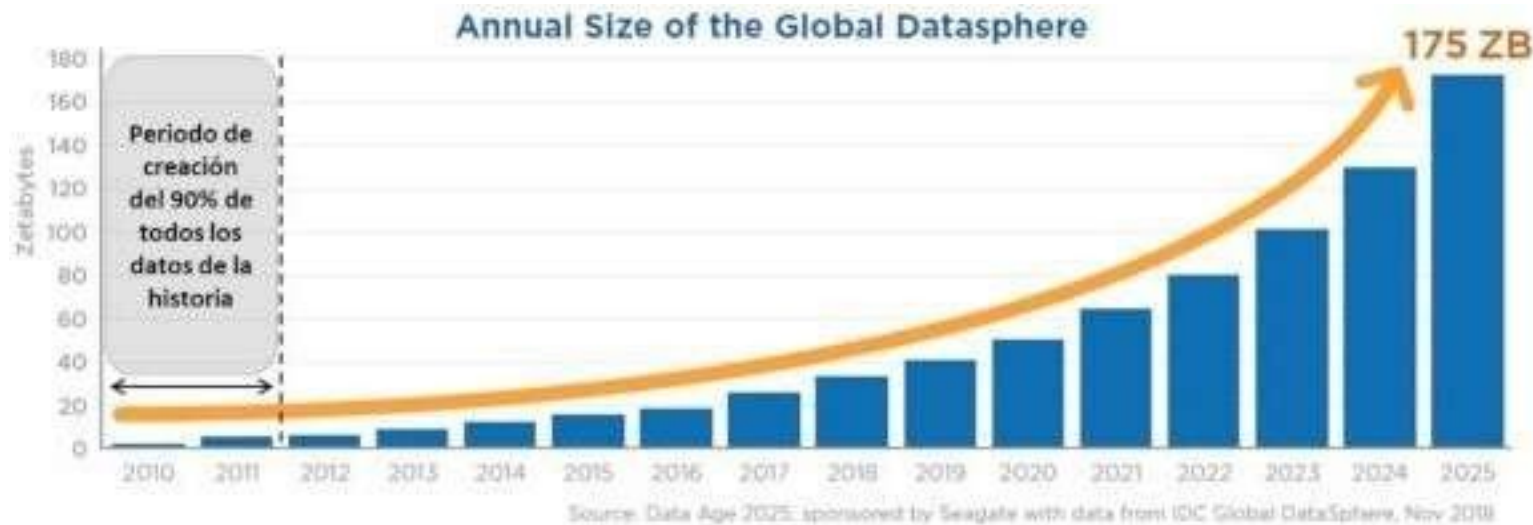
Introducción

Introducción

Algunos datos...

El 90% de los datos existentes se han creado en los últimos años (IBM).

- Existen 4,6 billones de teléfonos móviles en el mundo.
- Facebook procesa 10 terabytes de datos cada día.
- Google procesa más de 25 petabytes de datos en un solo día.
- Twitter procesa más de 7 terabytes de datos cada día.



[Gantz, J; Rydning, J; Reinsel, D. \(2019\) "The digitization of the World. From Edge to Core"](#)



Data Science y Big Data

Big Data

Big Data son un conjunto de tecnologías orientadas al almacenamiento y procesamiento de datos. Datos que son tan grandes, rápidos o complejos que es **difícil o imposible procesarlos con los métodos tradicionales**.

3 Vs Douglas Laney: Volumen, Variedad y Velocidad.

Estas tecnologías que permiten procesar grandes volúmenes de datos, de distintas tipologías (imágenes, textos, audios, ...) y que se generan a distinta velocidad. Siempre con la intención de generar valor a partir de todos estos datos.



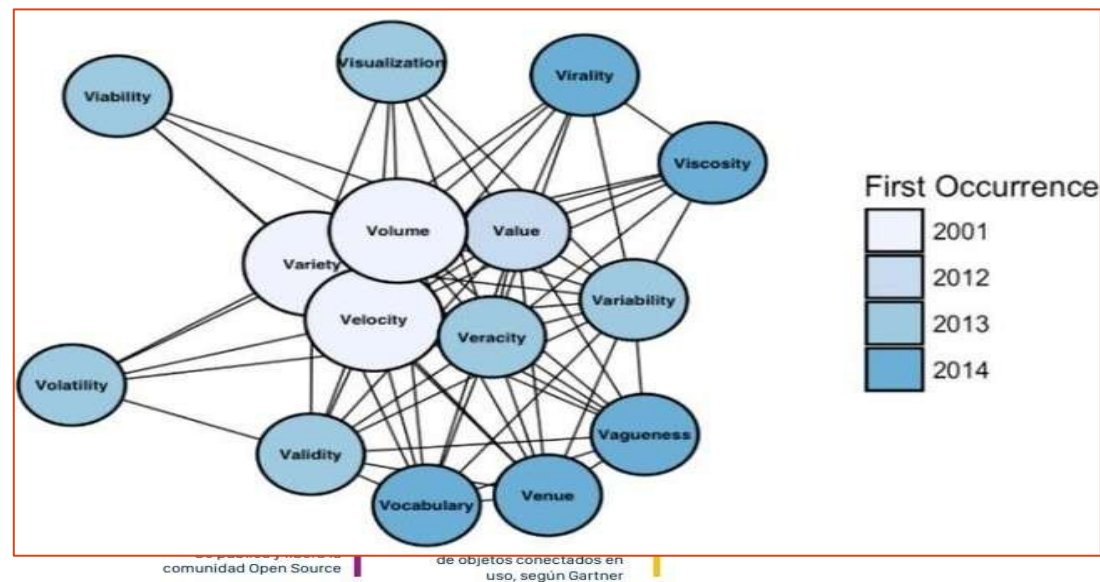
Laney, D. (2001) "3D data management: Controlling data volumen, variety and velocity".
Lahoz, J. (2019): "Big Data, Concepto y Evolución".

Big Data

Big Data son un conjunto de tecnologías orientadas al almacenamiento y procesamiento de datos. Datos que son tan grandes, rápidos o complejos que es **difícil o imposible procesarlos con los métodos tradicionales**.

3 Vs Douglas Laney: Volumen, Variedad y Velocidad.

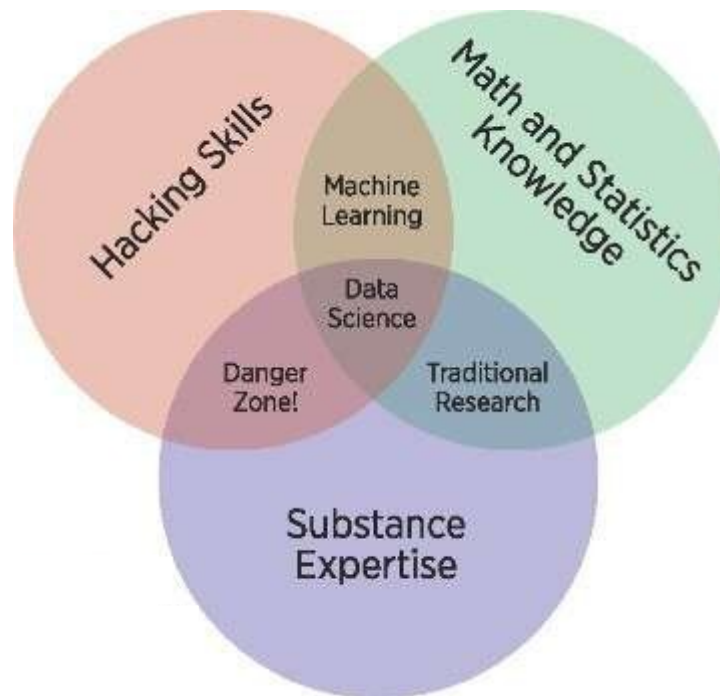
Estas tecnologías que permiten procesar grandes volúmenes de datos, de distintas tipologías (imágenes, textos, audios, ...) y que se generan a distinta velocidad. Siempre con la intención de generar valor a partir de todos estos datos.



Data Science

Data Science es una disciplina científica centrada en el análisis de grandes fuentes de datos para **extraer información**, comprender la realidad y **descubrir patrones** para tomar decisiones.

El término Data Science ha estado presente en los últimos 30 años; sin embargo, no es hasta la **década de los 70** cuando empezó a usarse para métodos de **preprocesamiento de datos**. En **2001**, la ciencia de datos se separa del Big Data y se proclama como disciplina independiente.





¿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**.
 - 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?

¿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 módulos 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?

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.
-

Ejemplo en Python

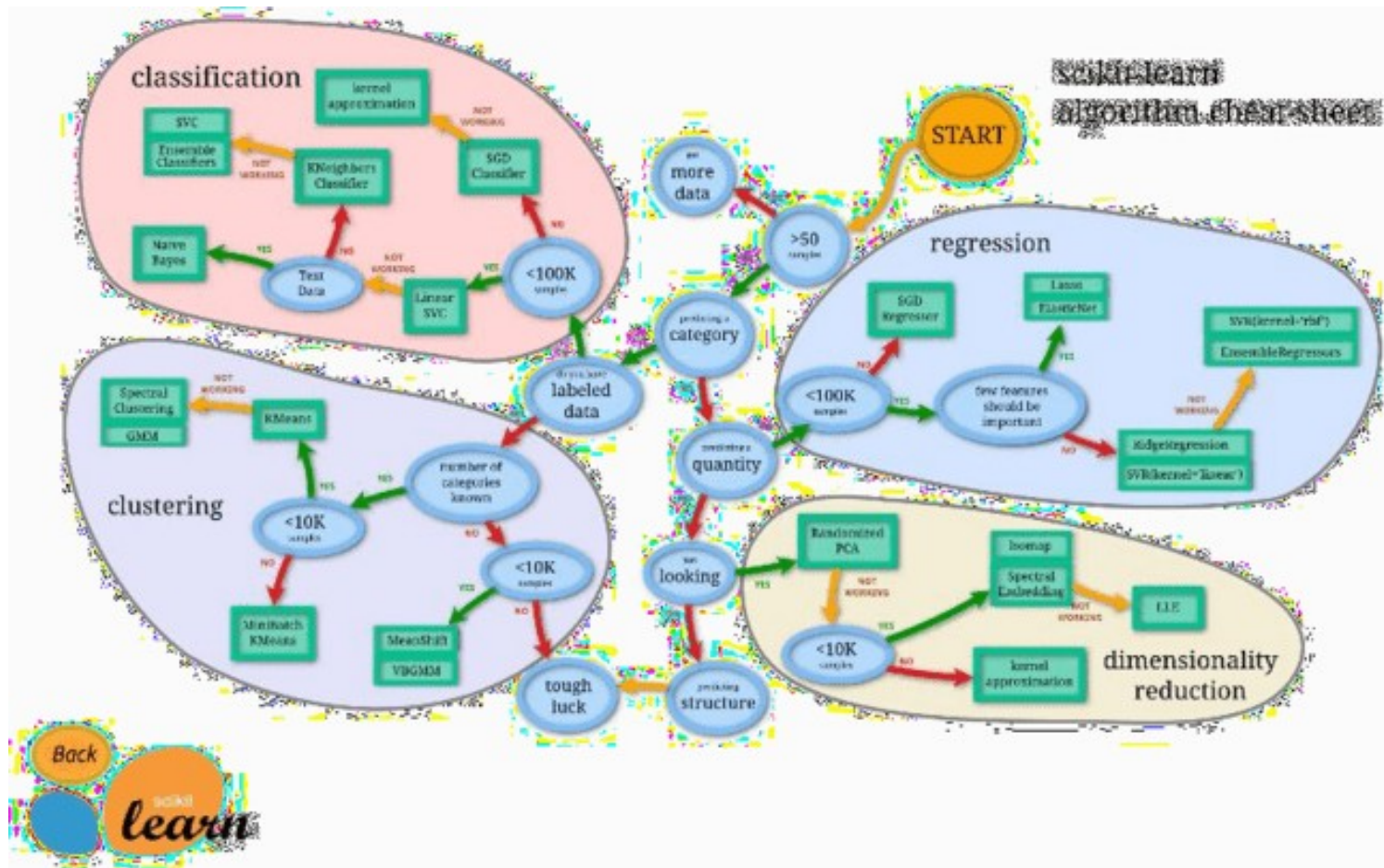


BasicsPython.ipnyb

¿Qué es Python?

Scikit-learn

Librería de Python utilizada para aprendizaje automático.



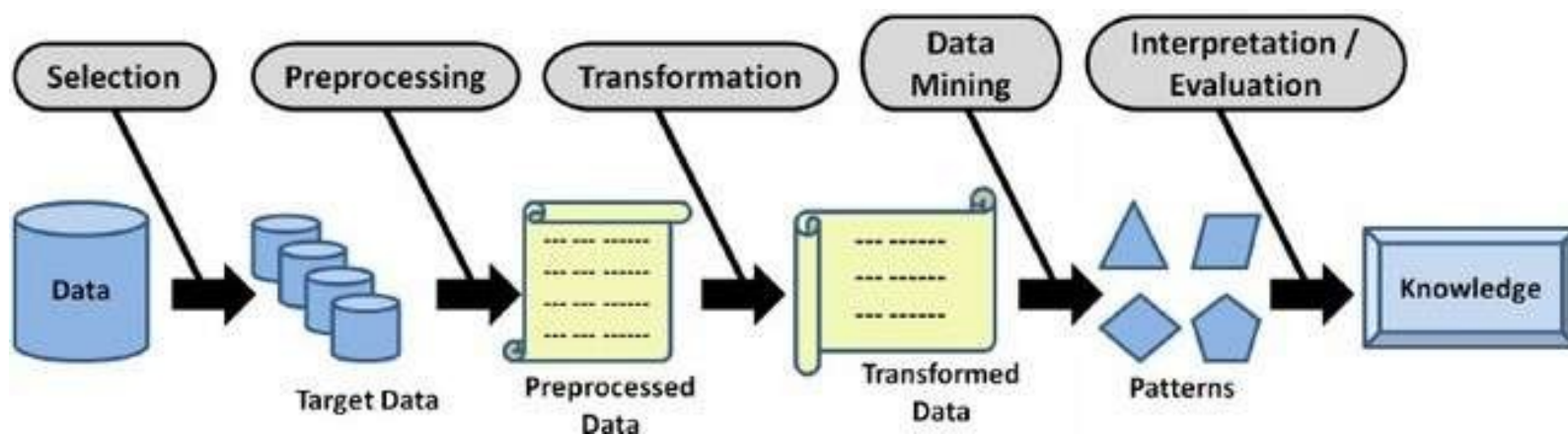


Proceso de Extracción del Conocimiento

Proceso de Extracción del Conocimiento

El **Proceso de Extracción del Conocimiento** (Knowledge Discovery in Databases, KDD) es:

- El proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles e inteligibles en datos.
- Un análisis exploratorio más o menos automático de bases de datos grandes.

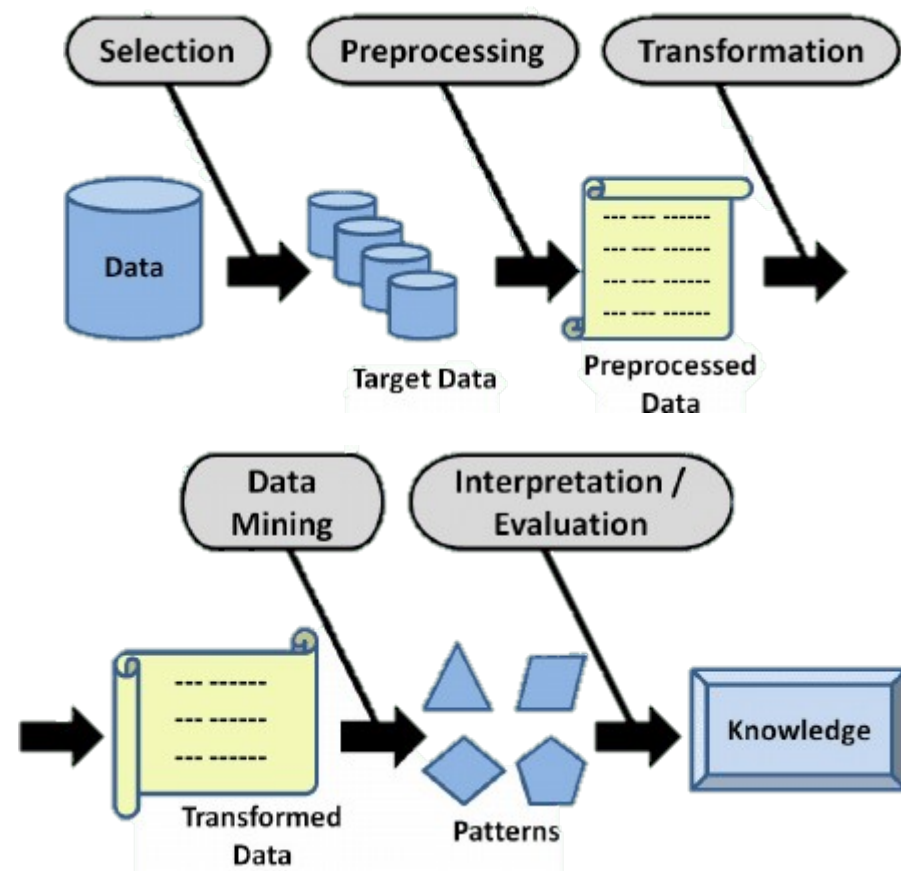


Proceso de Extracción del Conocimiento

Paso 1.

COMPRENDER EL PROBLEMA Y ESTABLECER OBJETIVOS.

Es fundamental tener claros los límites y objetivos que pretendemos. En este paso, se reúne toda la información más importante a utilizar .

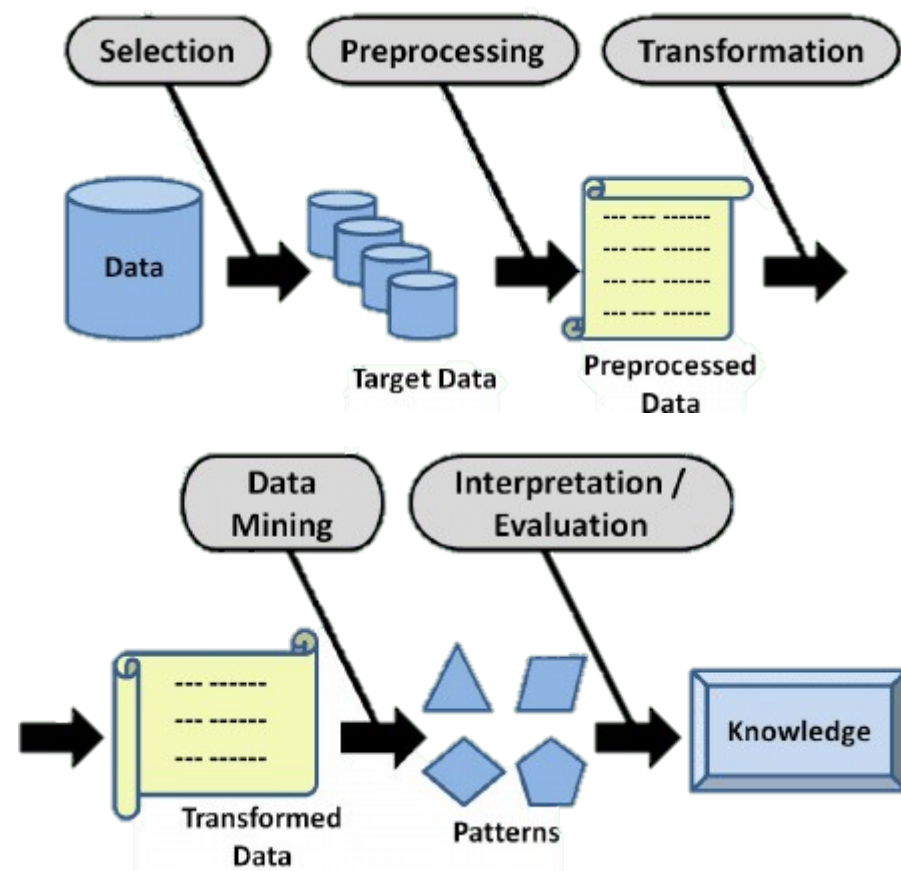


Proceso de Extracción del Conocimiento

Paso 2.

CREAR UN SET DE DATOS OBJETIVO.

Una vez establecido el problema, debemos determinar cuál es la variable objetivo. Los datos pueden existir en bases de datos, documentos, imágenes, etc. Debemos homogeneizar los formatos para poder procesarlos y analizarlos.



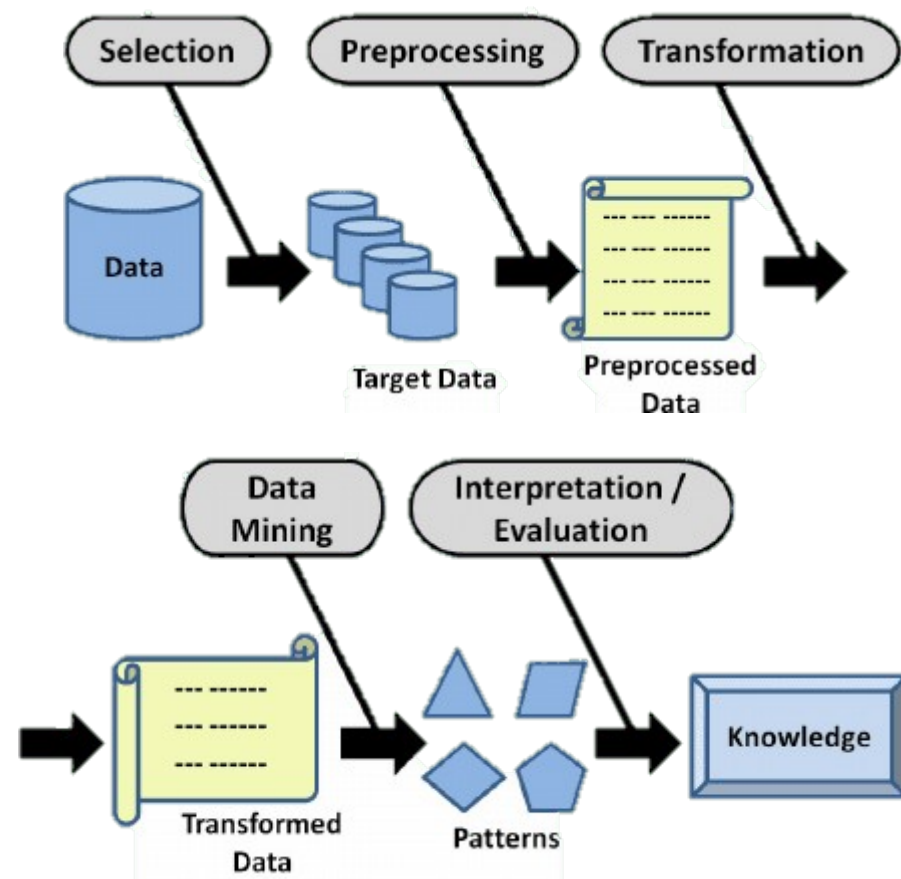
Proceso de Extracción del Conocimiento

Paso 3.

PREPROCESAMIENTO Y LIMPIEZA DE DATOS.

- Eliminación de ruido y datos aislados o *outliers*.
- Eliminar inconsistencias y duplicados.
- Imputar información faltante.

El preprocesamiento y limpieza tiene como objetivo mejorar la calidad de los datos y los resultados de la minería.



Proceso de Extracción del Conocimiento

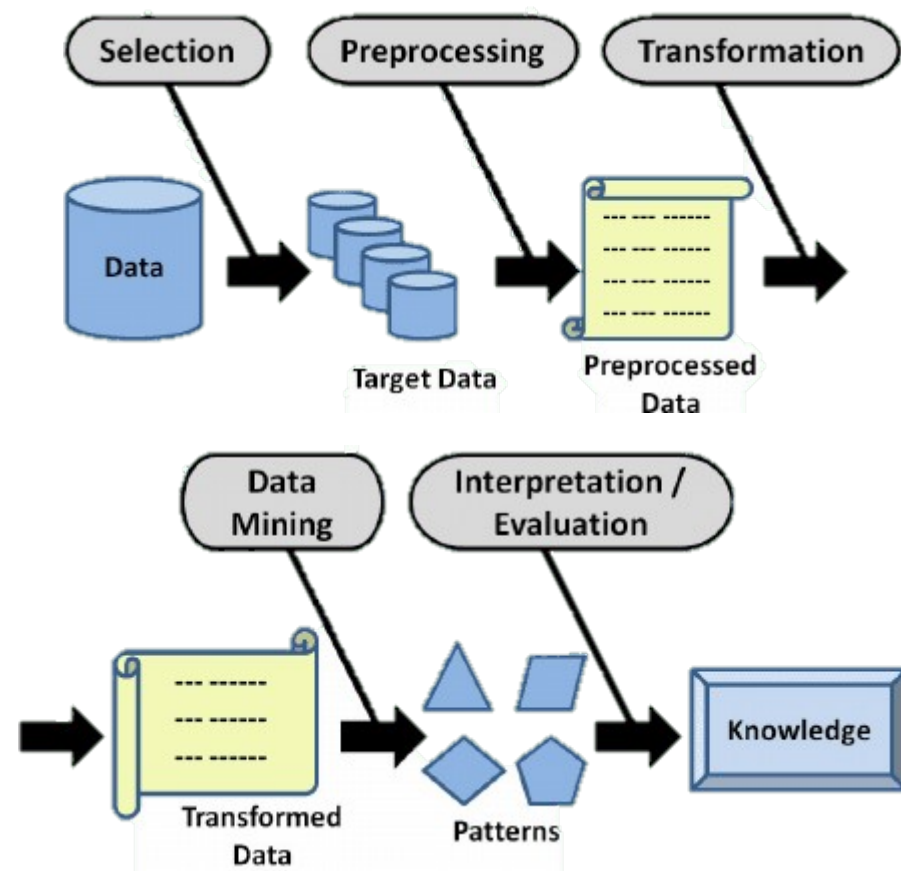
Paso 4.

MINERÍA DE DATOS

Haciendo uso de algoritmos, extraemos nueva información de nuestro set de datos.

Pasos de la minería de datos:

- Seleccionar la tarea
- Seleccionar el algoritmo/algoritmos a utilizar
- Usar nuestros algoritmos

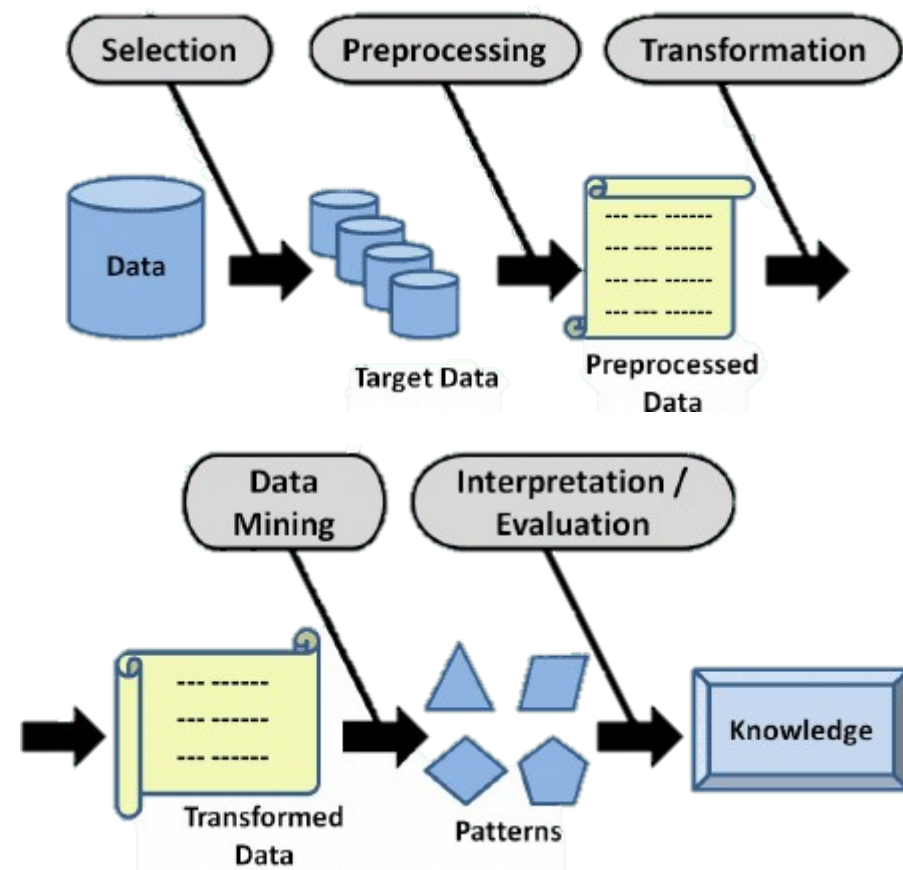


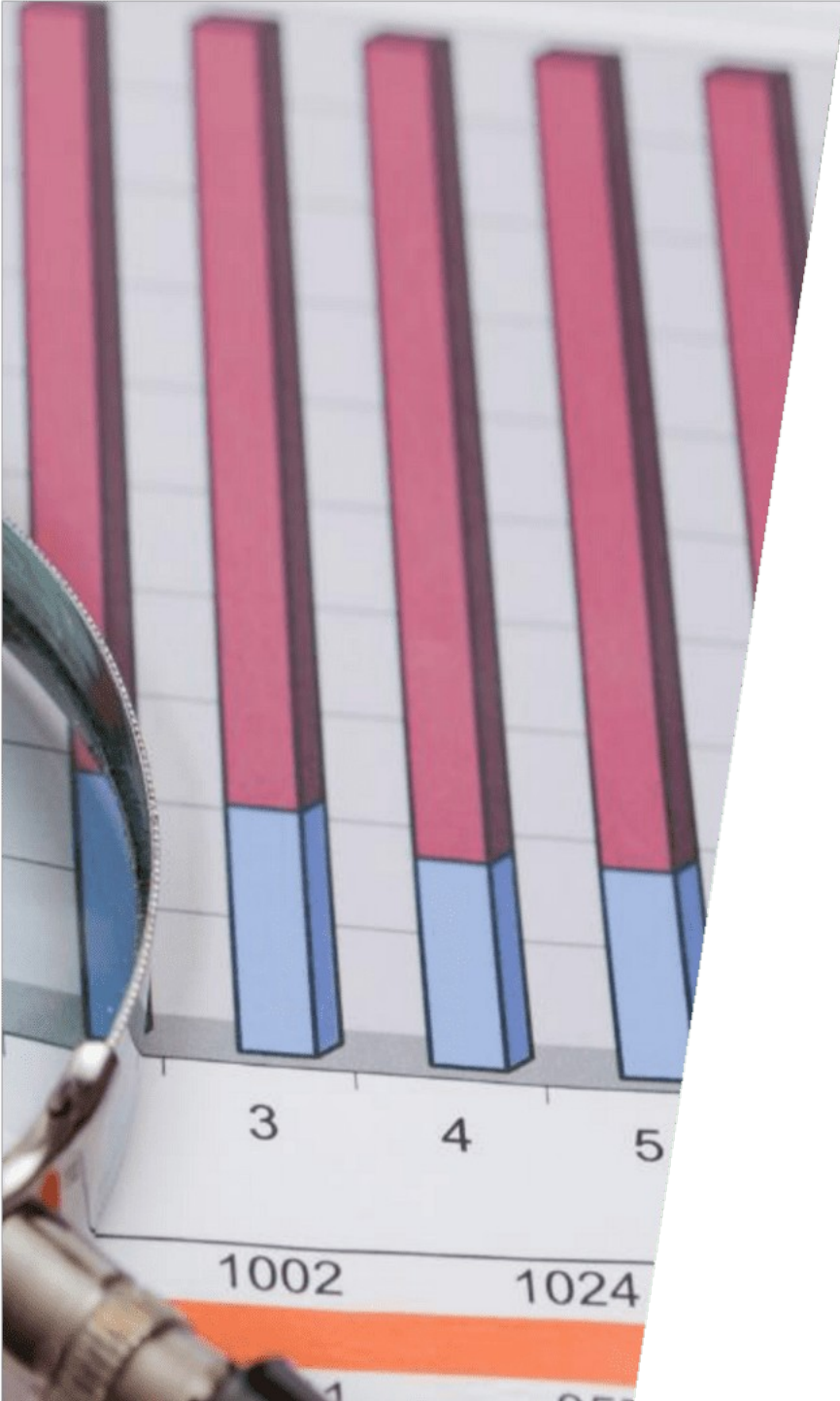
Proceso de Extracción del Conocimiento

Paso 5.

INTERPRETACIÓN DE LOS PATRONES

Esta etapa consiste en la interpretación de los resultados obtenidos; así como la exposición de manera clara de éstos para que sean entendibles. Es muy habitual el uso de técnicas de visualización.





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 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

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

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.

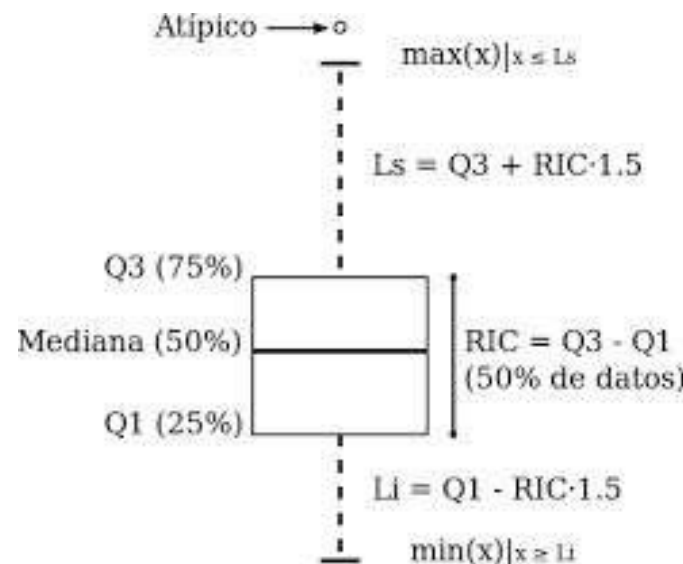
Preprocesamiento de datos

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



PreprocesadoInformacion.ipnyb



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.

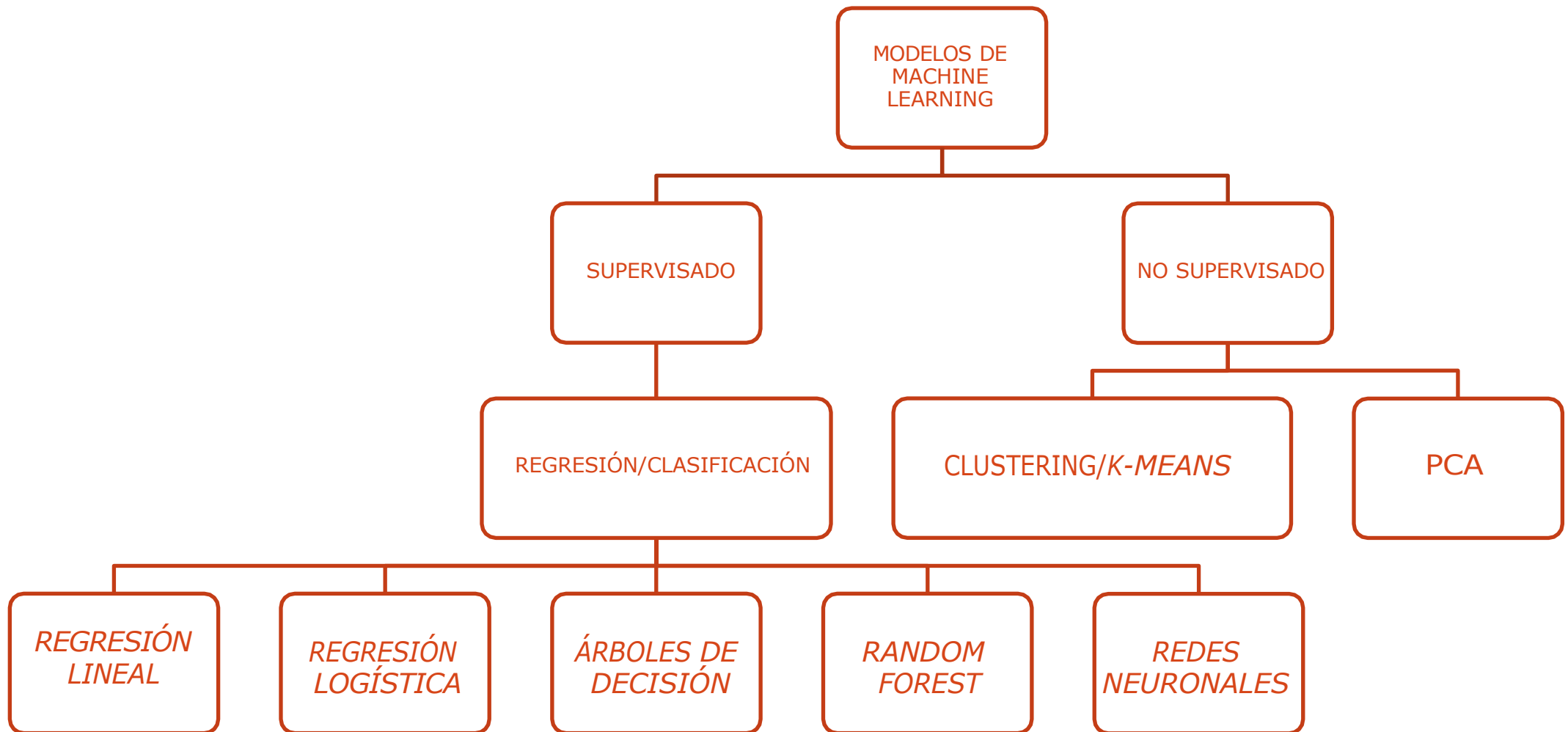
- **Regresión:** trata de predecir un número.
- **Clasificación:** trata de predecir una categoría.

APRENDIZAJE NO SUPERVISADO

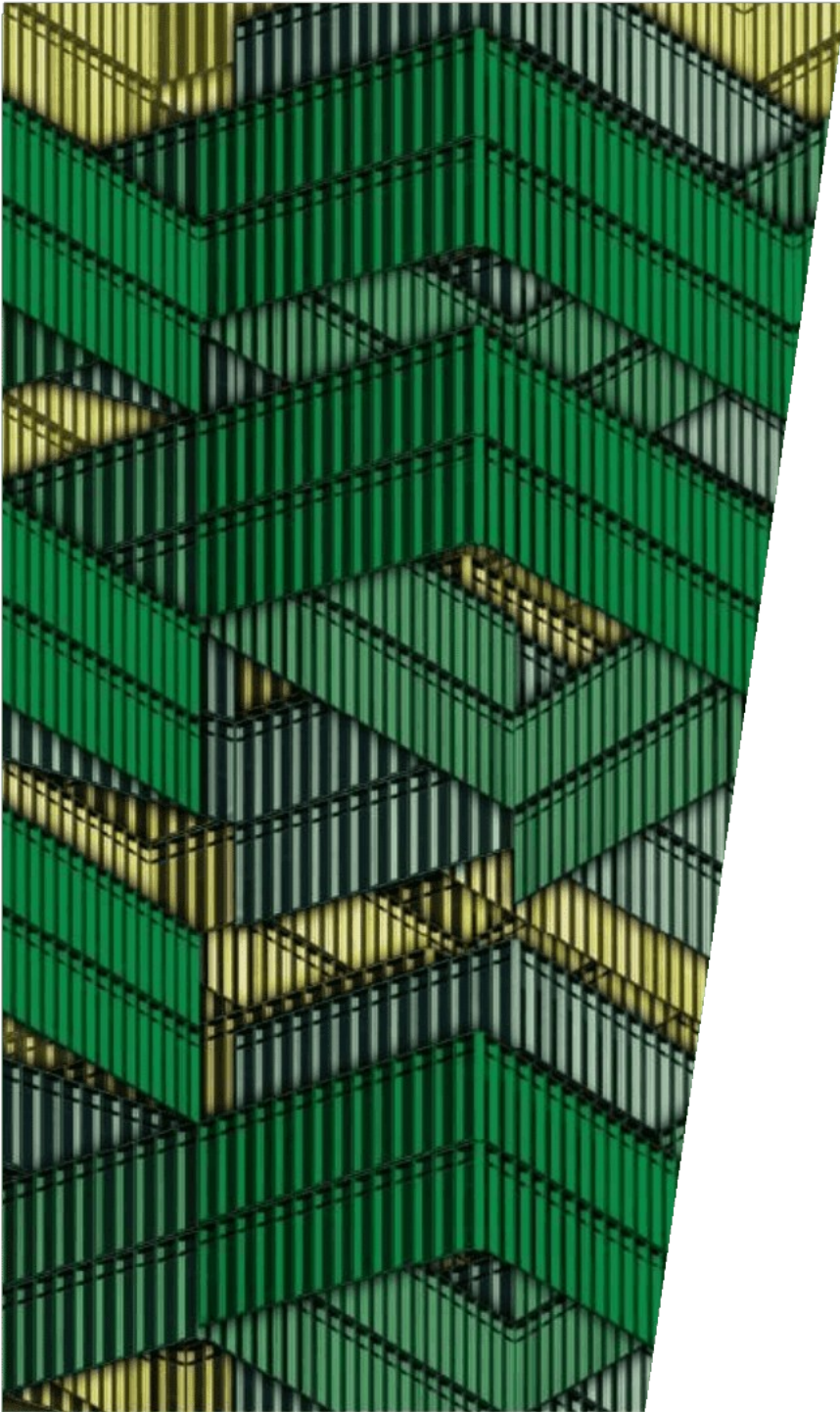
Para cada observación, tenemos un vector de medidas que no lleva asociada una respuesta.

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!

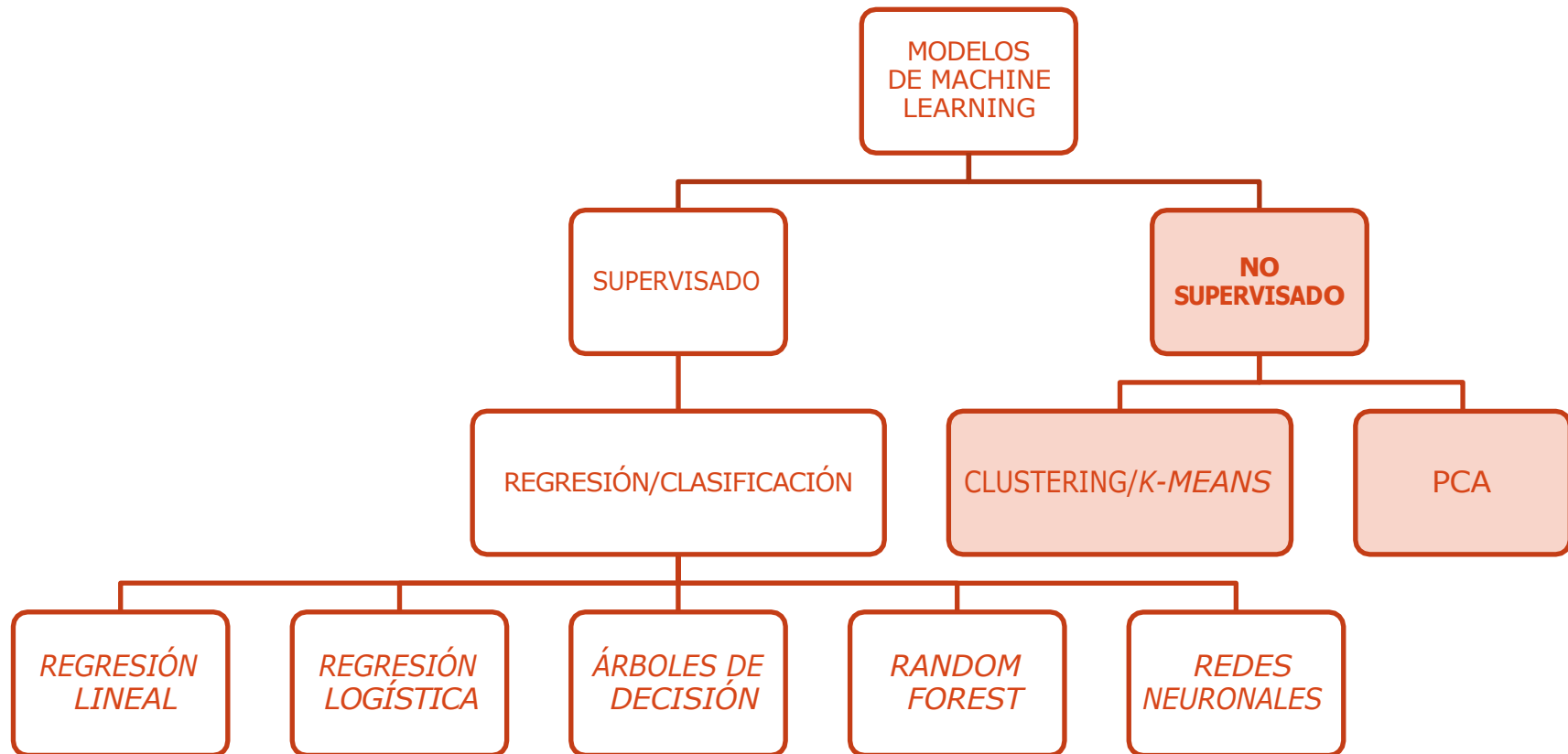


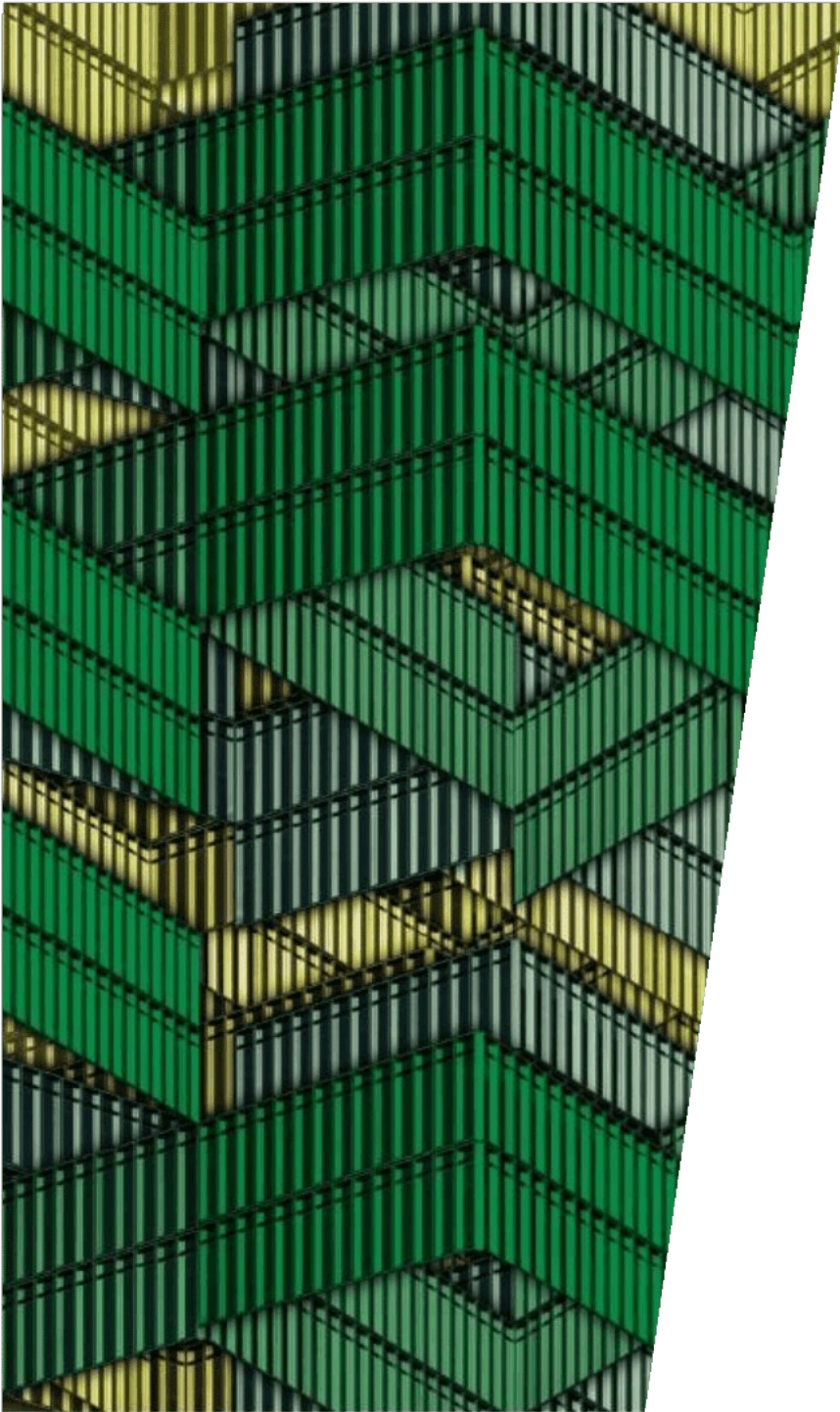
Aprendizaje no supervisado

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 principales.

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 .

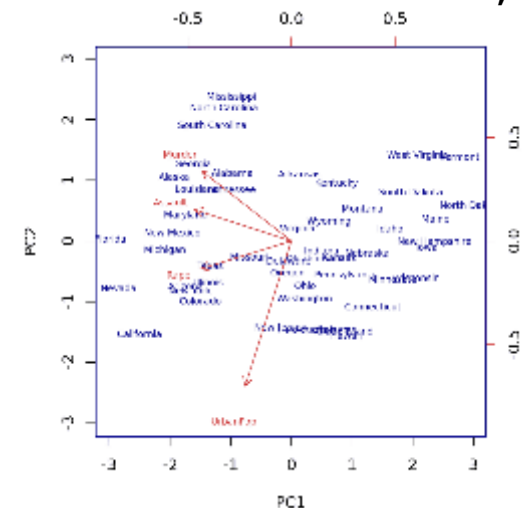
$$z_j = a_{j1}X_1 + a_{j2}X_2 + \dots + a_{jp}X_p$$
$$\text{var}(z_j) = \text{var}(a_j x) = a_j^T S a_j$$

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:

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

a_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

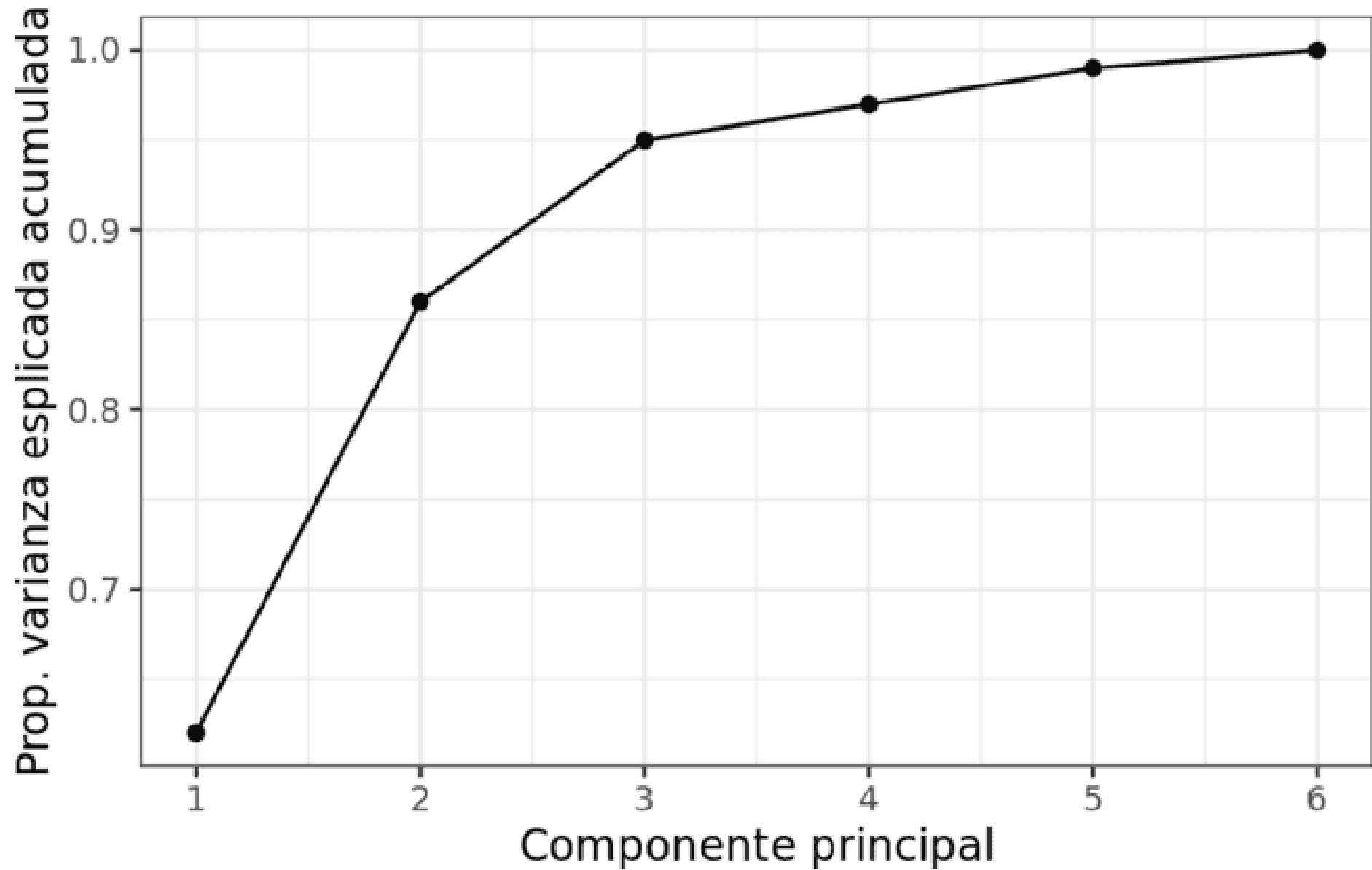
Componentes Principales en Python

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



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:

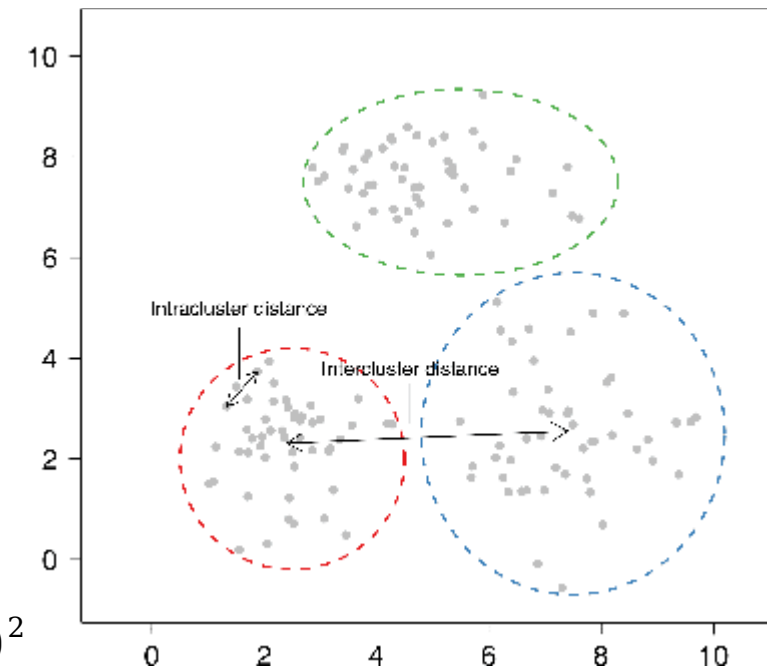
- Minimizar la varianza dentro de cada grupo.

$$WSS = \sum_{g=1}^G \sum_{j=1}^p (x_{ij} - \bar{x}_{\cdot j})^2$$

WSS: Within Cluster Sum of Square

- Maximizar la varianza entre grupos

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

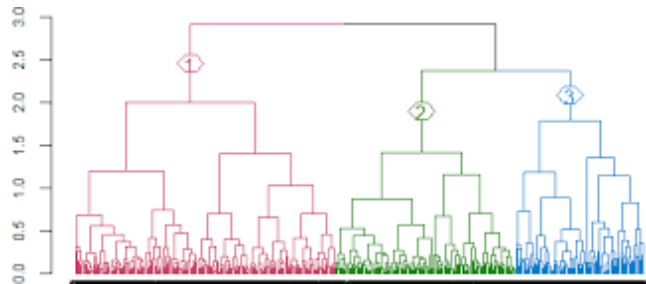


Aprendizaje no supervisado

Clustering – algoritmos de clasificación

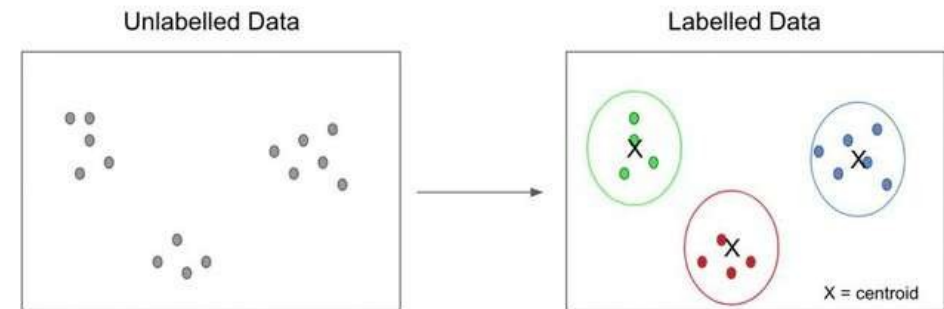
Métodos jerárquicos

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



Métodos de partición

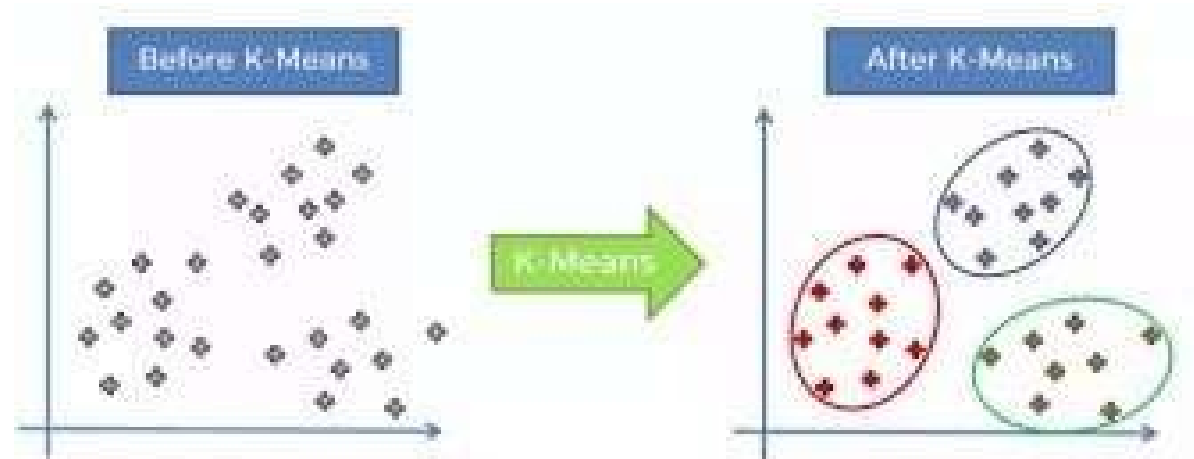
Dividen las observaciones en un número de grupos pre-especificado.



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



Clustering.ipnyb

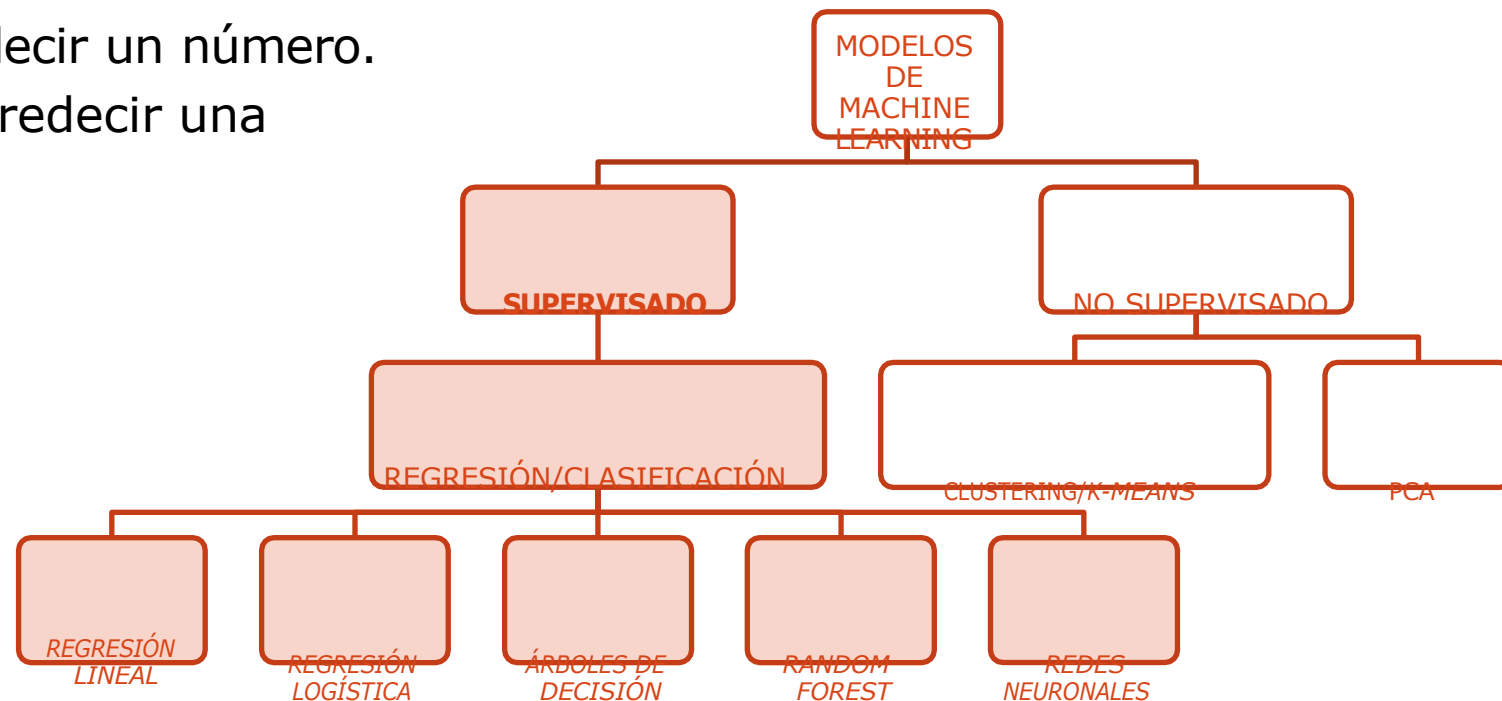
Aprendizaje supervisado

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

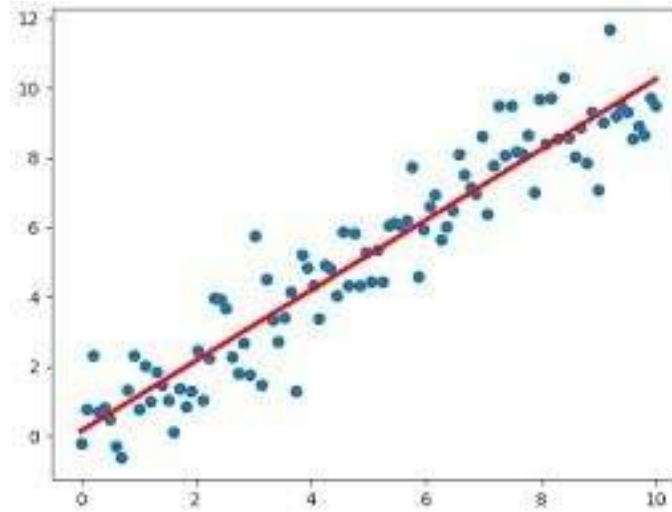
El modelo de **regresión lineal** es una de las formas más sencillas de estimar nuestra variable Y a partir de los predictores. La regresión lineal resuelve un problema de **regresión**.

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

A partir de los datos disponibles estimaremos los **coeficientes** función h :

◈ β_0 y β_1 , obteniendo como
◈ β^0
◈

$$Y \approx Y^{\hat{}} = \beta_0 + \beta_1 X$$



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:

$$\beta_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

$$\beta_0 = \frac{-}{\beta_1} x_0$$

Aprendizaje supervisado

Regresión Lineal

Para medir la precisión entre los coeficientes una relación lineal entre X y Y verificando:

1. $Y \in \mathbb{R}$, debemos asumir la existencia de β^0

$$Y = \beta_0 + \beta_1 X + \varepsilon, \quad (0, \infty), \quad \varepsilon \perp X$$

$$\varepsilon \sim N$$

Nos referimos a la expresión anterior como

$$SE(\beta_0)^2 = \left[\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right] \sigma^2, \quad SE(\beta_1)^2 = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

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:

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

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**:

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

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

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

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

Aprendizaje supervisado

Regresión Lineal 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



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**, el modelo es **multinomial**.

El modelo de regresión logística es:

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 .



El modelo de regresión logística, será un modelo

Aprendizaje supervisado

Regresión Logística

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

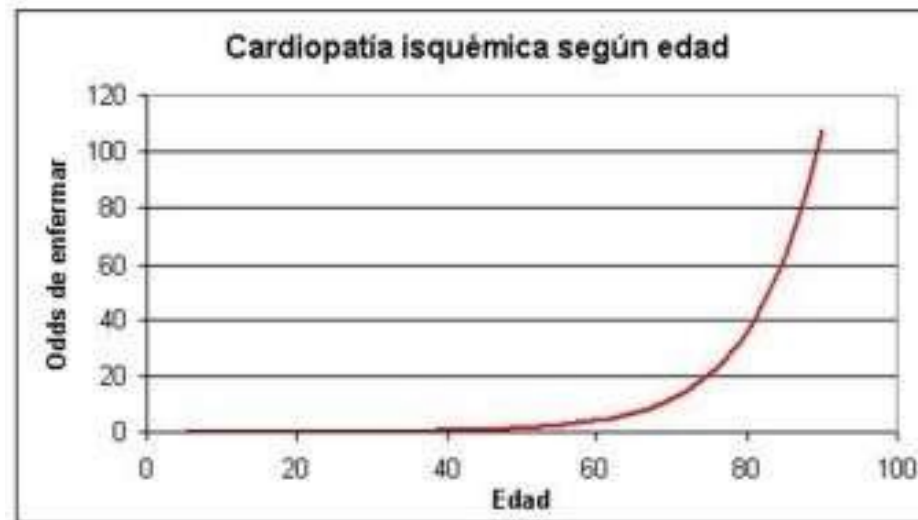
$$\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} = \exp(b_0 + \sum_{i=1}^n b_i x_i)$$

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:

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

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

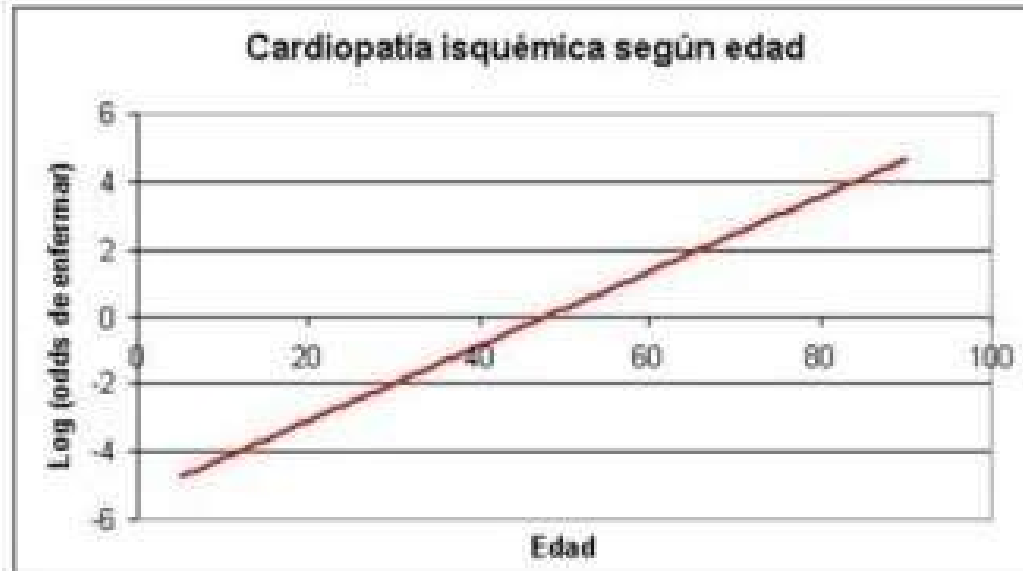
$$\log \left(\frac{\Pr(y = 1|x)}{1 - \Pr(y = 1|x)} \right) = b_0 + \sum_{i=1}^n b_i x_i \quad \log \left(\frac{p_i}{1 - p_i} \right) = \beta_0 + \sum_{i=1}^n \beta_i x_i$$

Aprendizaje supervisado

Regresión Logística

Dividiendo la expresión
expresión es más ma

Aplicando el logaritmo
de la razón de propo



los Odds. La

responde al logaritmo

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

Aprendizaje supervisado

Regresión Logística en Python

Para calcular un modelo de regresión lineal 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 lineal 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 lineal 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



RegresionLogistica.ipnyb

The background image is a collage of financial data. It includes a bar chart with blue bars of varying heights, a table with columns for 'Company A', 'Company B', and 'Company C' showing various financial figures, and a blue box containing text labels like 'ANNUAL AMOUNT', 'TERM OF LOAN IN MONTH', 'FIRST PAYMENT DATE', 'MONTHLY PAYMENT', 'TOTAL INTEREST', and 'TOTAL PAYMENTS'.

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:

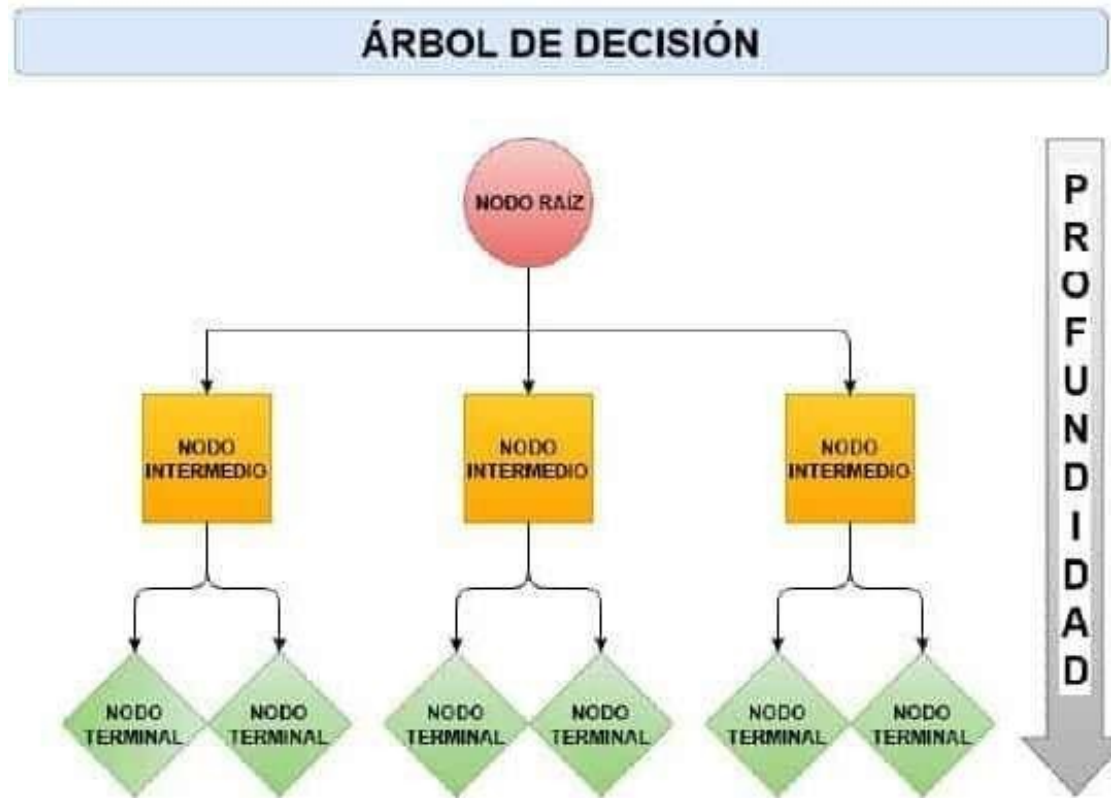
- Transparencia: fácil de interpretar.
- Portabilidad: las pautas que se extraen del camino a una hoja del árbol se expresan fácilmente en distintos formatos.
- Modelización: pueden utilizar tanto variables continuas como categóricas.

Desventajas:

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

Aprendizaje supervisado

Árboles de decisión - terminología

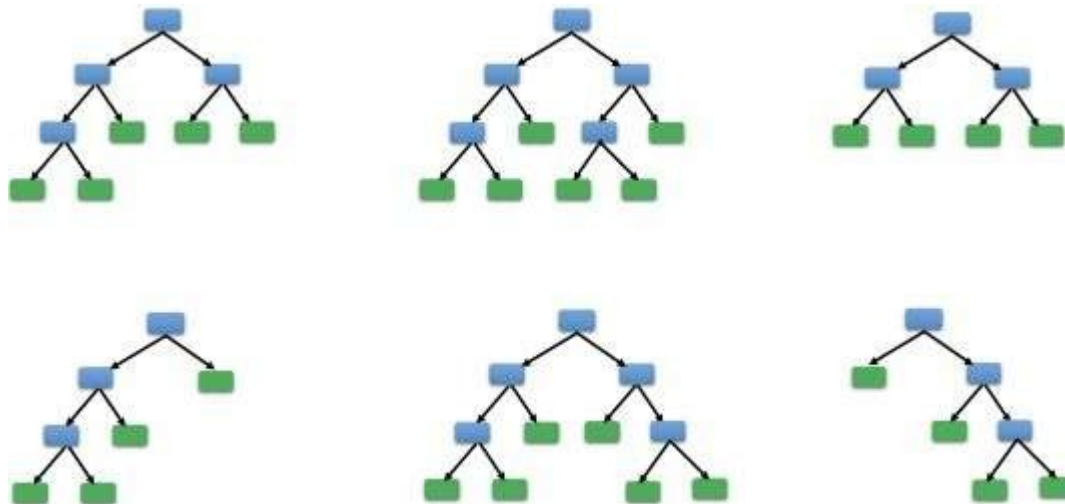


Aprendizaje supervisado

Árboles de decisión

¿Cómo mejorar los resultados en árboles de decisión?

- **Prepoda:** decidimos cuándo queremos parar de construir el árbol.
- **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



ArbolesDecision.ipnyb



Aprendizaje supervisado Redes Neuronales

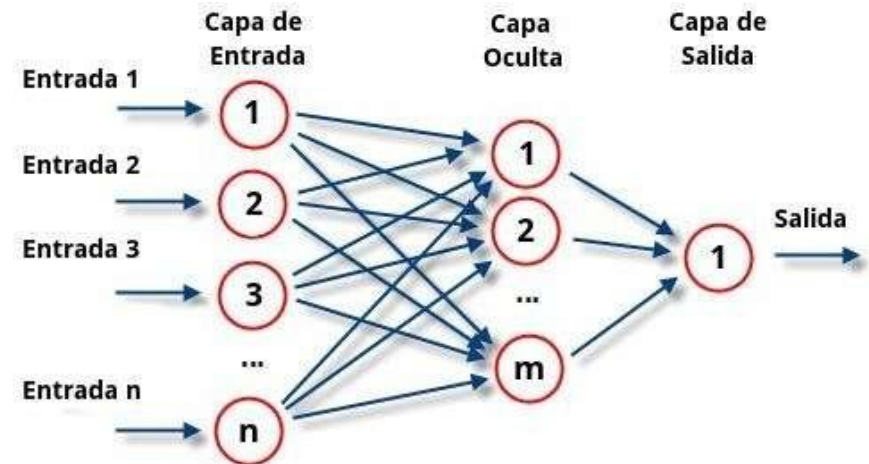
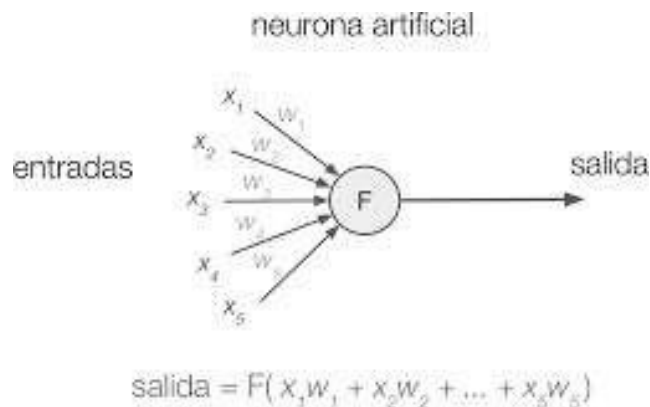
Aprendizaje supervisado

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.



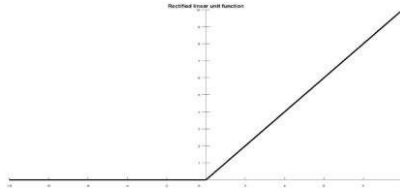
Aprendizaje supervisado

Redes Neuronales – funciones de activación

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

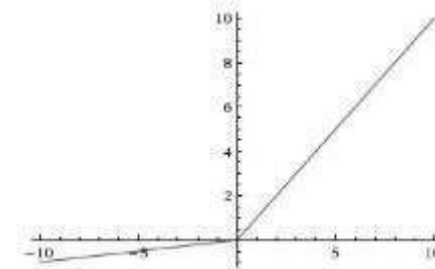
RELU

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



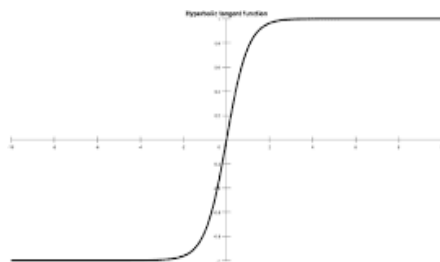
Leaky RELU

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



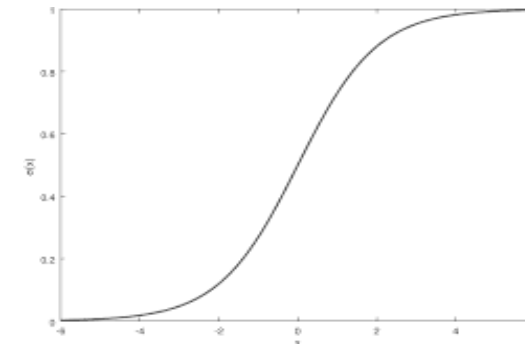
Tangente hiperbólica

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



Sigmoide

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



Aprendizaje supervisado

Redes Neuronales en Python

SCIKIT-LEARN

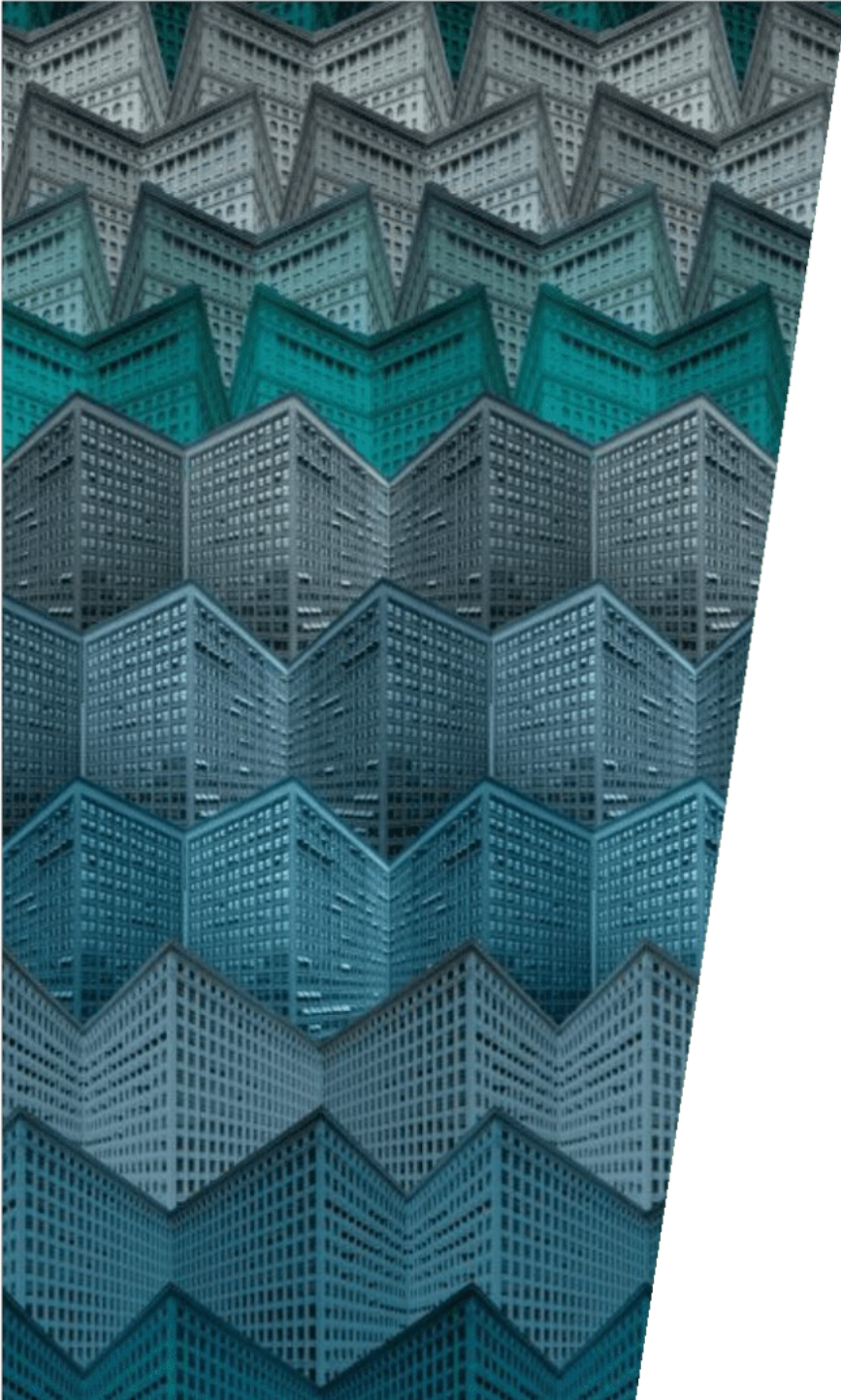


KERAS



TENSORFLOW

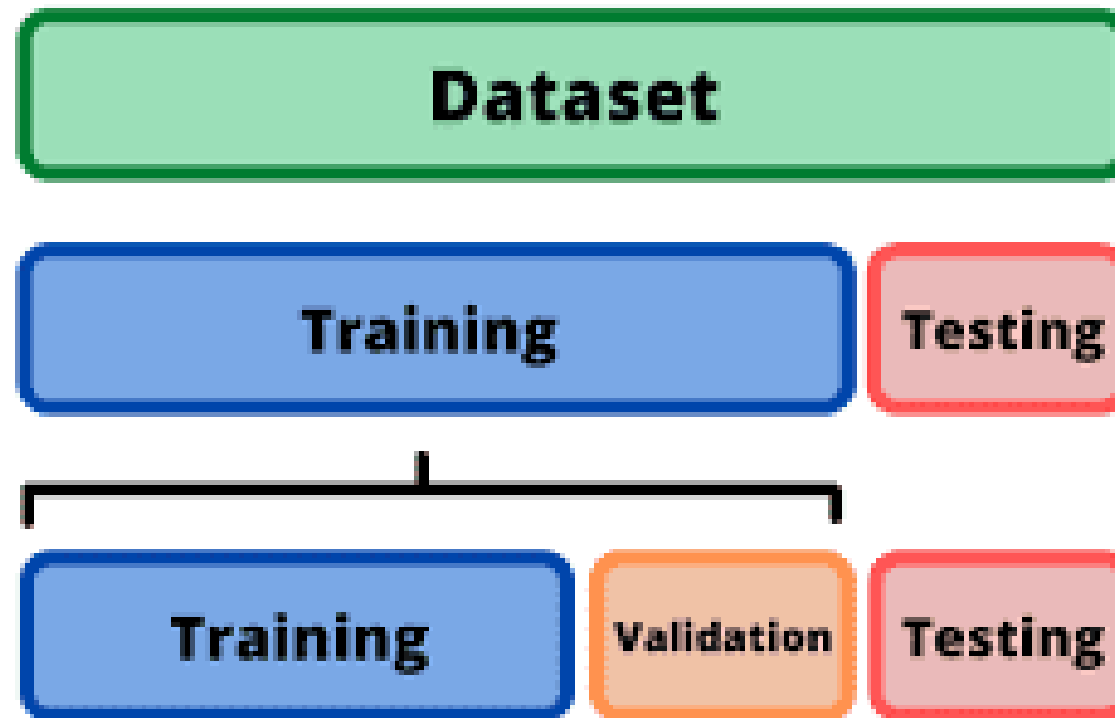




Metodología

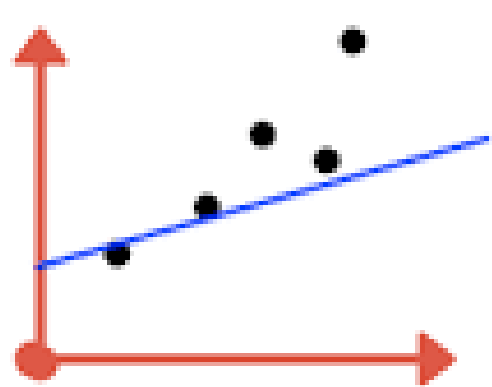
Metodología

Separación train-test-validación

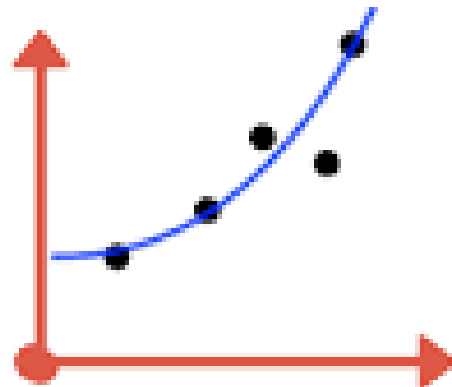


Metodología

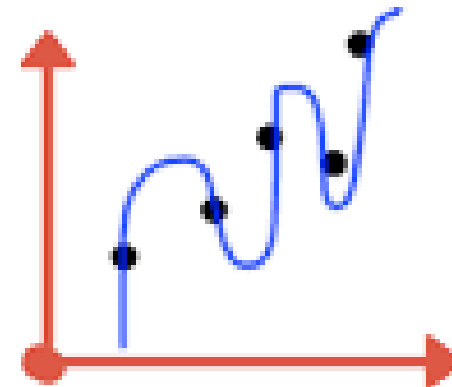
Overfitting vs underfitting



underfitting



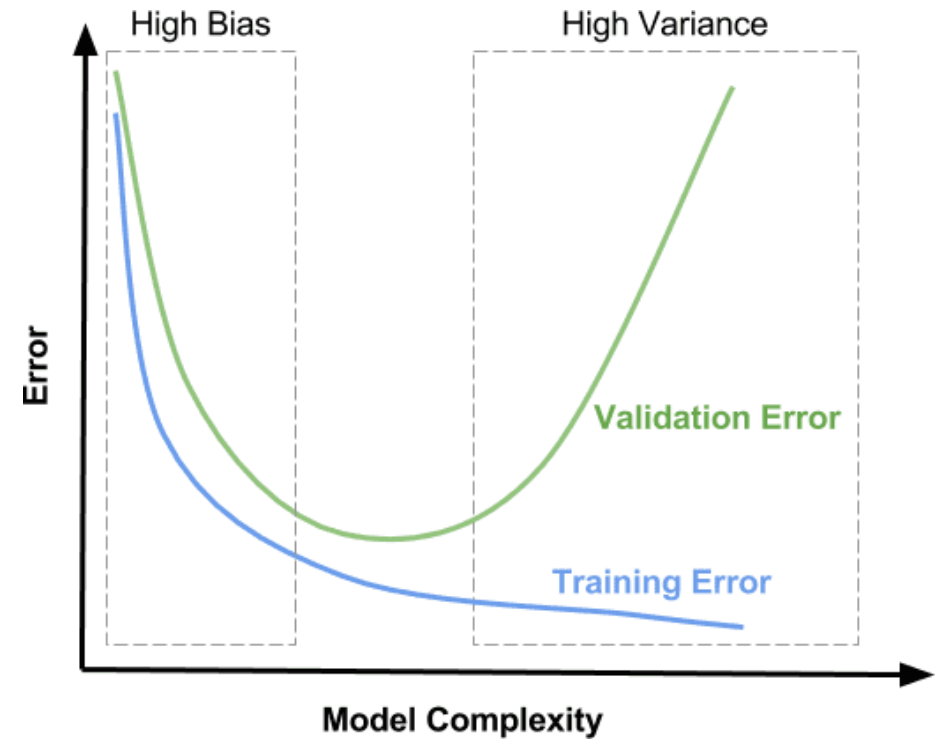
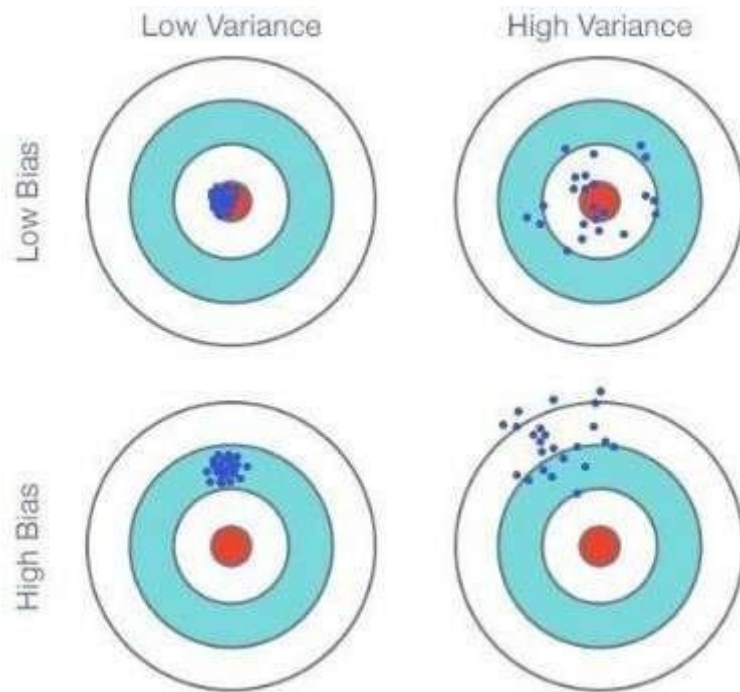
correcto



overfitting

Metodología

Sesgo y varianza

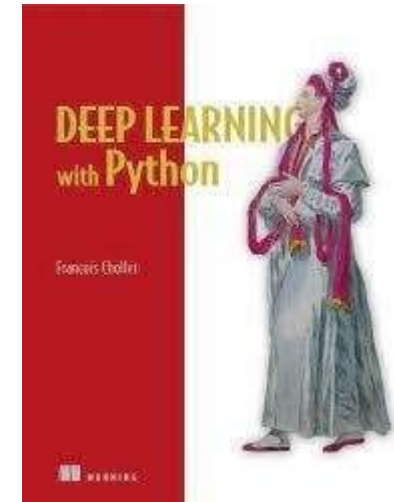
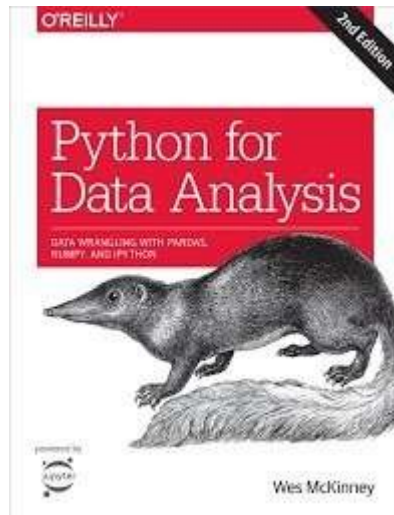





Referencias

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





Data Science

Luciano Moliiterno