Universidad Autónoma Chapingo

Enseñar la explotación de la tierra, no la del hombre

Universidad Autónoma Chapingo

Centro de Educación Continua

Ciencia de datos e inteligencia artificial en agricultura

Herramientas de ciencia de datos y machine learning

M.I Canek Mota Delfin

Temario

1. Estructura de datos y manejo de librería de Pandas
2. Análisis, visualización de datos y estadística básica
3. Técnicas de regresión y PCA
4. Clasificación/Regresión Machine Learning

Evaluación

Ejercicios en clases y participación en clases

Los datos son el nuevo oro del siglo

En la era digital en la que vivimos, las empresas tienen acceso a una cantidad increíble de datos y su ventaja competitiva reside en la capacidad de extraer información a partir de ello.

Puedes revisar en Google cuanta información tiene de ti.

¿Qué son los datos?

Es una representación simbólica de un atributo o variable que puede ser cuantitativa o cualitativa.

Ejemplo: Los números en una lista, las palabras en un texto, o los símbolos en una ecuación.

Los datos se refieren a la información existente o al conocimiento representado de alguna forma que permita su uso y análisis

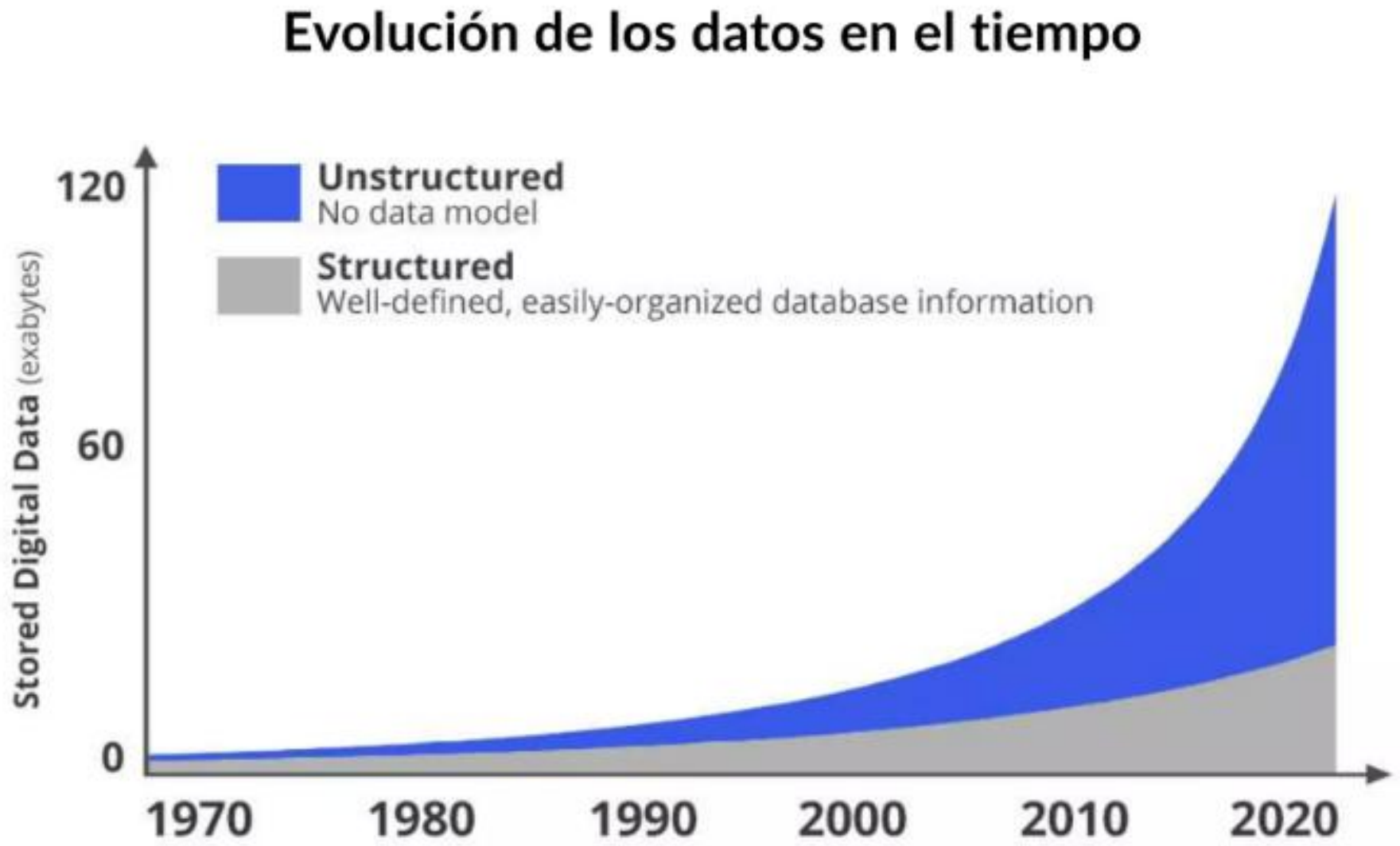
Tipos de datos

Estructurados

	A	B	C	D	E	F
1	FECHA	SECTOR	TIPO CONTRATO	OPCION	VENDEDOR	IMPORTE
2	02/01/2011	TV-HIFI	1 AÑO	TOTAL	SANDRA	62
3	02/01/2011	TV-HIFI	1 AÑO	TOTAL	SANDRA	62
4	02/01/2011	P-EM	1 AÑO	TOTAL	MARIA	62
5	03/01/2011	P-EM	2 AÑOS	TOTAL	SANDRA	112
6	03/01/2011	TV-HIFI	1 AÑO	PIEZAS	LUIS	52
7	03/01/2011	G-EM	1 AÑO	TOTAL	PEDRO	62
8	03/01/2011	G-EM	3 AÑOS	TOTAL	MARIA	156
9	03/01/2011	TV-HIFI	1 AÑO	TOTAL	SANDRA	62
10	03/01/2011	P-EM	1 AÑO	TOTAL	LUIS	62
11	03/01/2011	TV-HIFI	2 AÑOS	TOTAL	SANDRA	112
12	06/01/2011	G-EM	2 AÑOS	TOTAL	SOFIA	112
13	06/01/2011	TV-HIFI	1 AÑO	PIEZAS	LUIS	52
14	06/01/2011	TV-HIFI	3 AÑOS	PIEZAS	SANDRA	130

No Estructurados

Tipos de datos



¿Qué es la información?

Es un conjunto organizado de datos que han sido procesados de manera que constituyen un mensaje significativo, cambiando el estado de conocimiento de quien lo recibe.

Ejemplo: Un informe generado a partir de las transacciones diarias de una empresa, o un análisis de mercado basado en encuestas.

La información es conocimiento explícito extraído como resultado de la interacción o las percepciones del entorno.

Ciencia de datos

La ciencia de datos es un campo interdisciplinario que utiliza métodos matemáticos, procesos y algoritmos para extraer conocimientos e ideas a partir de los datos.

Esta disciplina combina varios campos como la estadística, la informática y el aprendizaje automático para analizar grandes volúmenes de datos para obtener información valiosa.

Ciencia de datos

La ciencia de datos es un campo interdisciplinario que utiliza métodos matemáticos, procesos y algoritmos para extraer conocimientos e ideas a partir de los datos.

Esta disciplina combina varios campos como la estadística, la informática y el aprendizaje automático para analizar grandes volúmenes de datos para obtener información valiosa.

Qué y no se puede hacer

La ciencia de datos es una herramienta poderosa con el potencial de proporcionar ideas valiosas y respuestas a través del análisis de datos. Sin embargo, también tiene sus limitaciones.

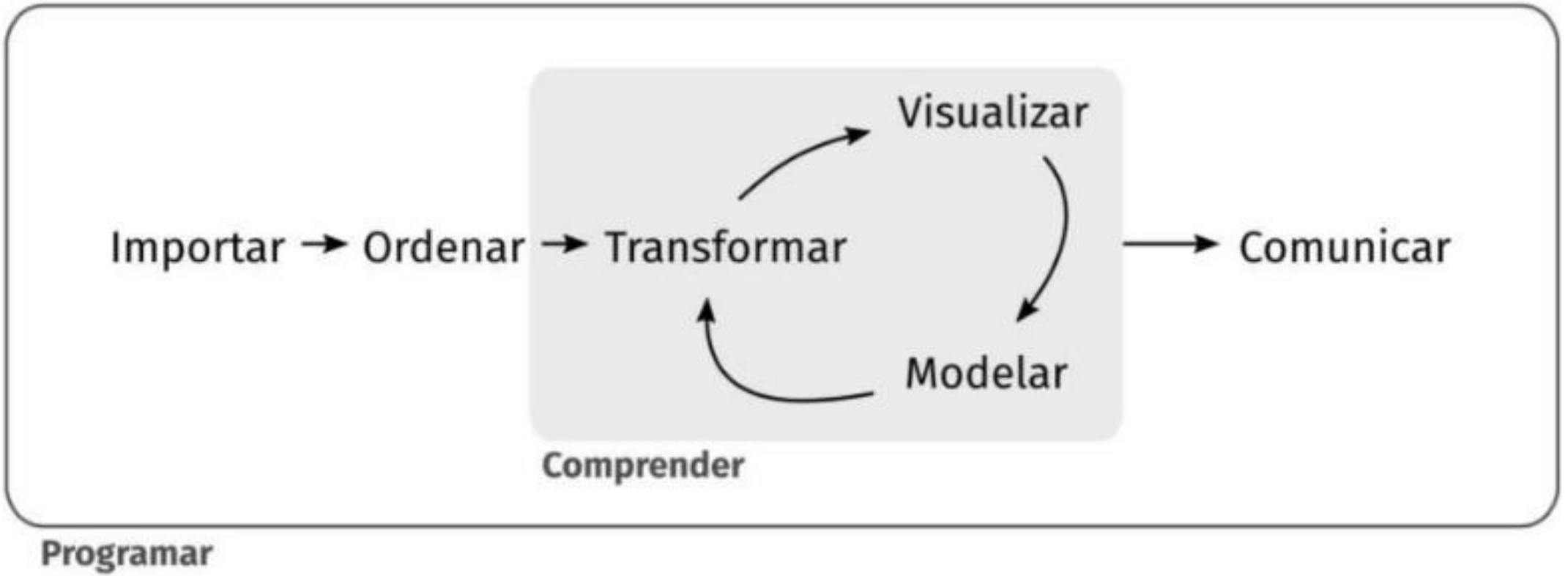
Lo que se puede hacer:

1. Descubrir patrones ocultos
2. Identificar tendencias
3. Realizar predicciones
4. Tomar desiciones informadas

Lo que no se puede hacer:

1. Proporcionar respuestas definitivas
2. Predecir el future con Certeza absoluta
3. Eliminar sesgos y limitaciones inherentes a los datos

Proceso de análisis de datos



La preparación de los datos representa alrededor del 80% del trabajo de los científicos de datos

Proceso de análisis de datos

La ciencia de datos se basa en varios pasos clave para transformar datos en conocimiento útil:

1. Recolección de datos
2. Limpieza de datos
3. Análisis exploratorio
4. Modelado
5. Evaluación / “Validación”
6. Comunicación

Que hace un científico de datos

- Formular preguntas para comprender un problema
- Recoger un conjunto de datos para resolver un problema determinado
- Tratar los datos (Limpieza, Fusión, etc) para que sean utilizables y presentables
- Visualización
- Decidir métricas, indicadores o criterios de éxito o fracaso
- Elegir modelos o algoritmos adecuados para resolver un problema
- Entrenar un modelo o proponer un modelo
- Presentar resultados, hacer recomendaciones

Habilidades para desarrollar

Nivel operacional: Habilidades técnicas

Ninguna es más importante que la otra, siempre debe haber balance

Estadística/ Matemática

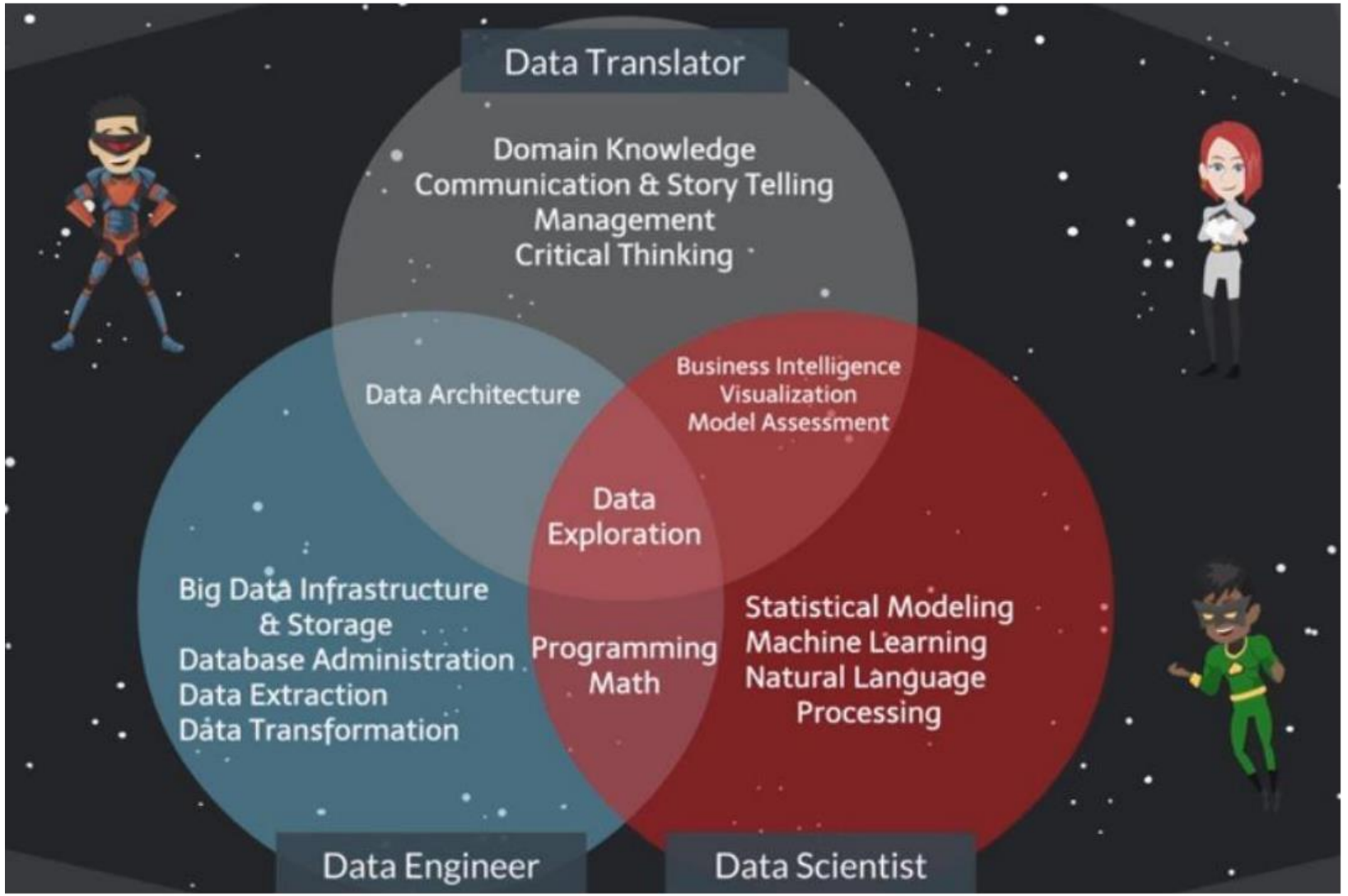
- Validación de modelos
- Generación de conclusiones

Habilidades de un hacker

- Aprovechar al máximo la tecnología

Experiencia

Roles operacionales



EDA

Exploratory Data Analysis o Análisis Exploratorio de datos

Es una fase crucial en el proceso de análisis de datos que se centra en resumir sus características principales, a menudo con métodos visuales

1. **Entender la distribución de los datos:** Conocer cómo están distribuidos los datos, identificar patrones, y detectar cualquier anormalidad o valor atípico.
2. **Descubrir relaciones entre variables:** Analizar las posibles relaciones y correlaciones entre diferentes variables.
3. **Verificar suposiciones iniciales:** Evaluar si las suposiciones hechas sobre datos son válidas.

Técnicas:

1. Estadística descriptiva
2. Visualización de datos
3. Detección de valores atípicos
4. Análisis Bivariado y Multivariado

Conociendo algunas herramientas

•**Pandas**: Fundamental para la manipulación y análisis de datos estructurados. Facilita la carga, manipulación y limpieza de datos.

•**NumPy**: Utilizada para operaciones matemáticas y cálculos eficientes sobre grandes arreglos y matrices de datos.

•**Matplotlib**: Librería básica para crear visualizaciones estáticas en Python.

•**Seaborn**: Basada en Matplotlib, proporciona una interfaz más amigable y atractiva para crear gráficos estadísticos.

•**Plotly**: Permite crear gráficos interactivos que pueden ser muy útiles en el análisis de datos.

•**SciPy**: Complementa a NumPy con más funciones matemáticas, científicas y de ingeniería.

•**Statsmodels**: Proporciona herramientas para la estimación de modelos estadísticos y realización de pruebas estadísticas.

Practica

De donde obtenemos datos

Kaggle

UCI Machine Learning Repository

Datos abiertos de México

Scikit-Learn

State of machine learning and data sciencie 2021

https://storage.googleapis.com/kaggle-media/surveys/Kaggle's%20State%20of%20Machine%20Learning%20and%20Data%20Science%202021.pdf

Trabajar con datos faltantes

Valores faltantes

Borrar toda la fila o columna

Sustituirla por la media de la variable

Borrar la fila o columna cuando hay un % de datos faltantes

En variables no numéricas sustituir el valor faltante por el dato más frecuente

Datos Categóricos

Nominales

Grupos sin orden

Ordinales

Grupos con orden

Datos Categóricos

Mala

Regular

Buena

Excelente

Muy insatisfecho

Insatisfecho

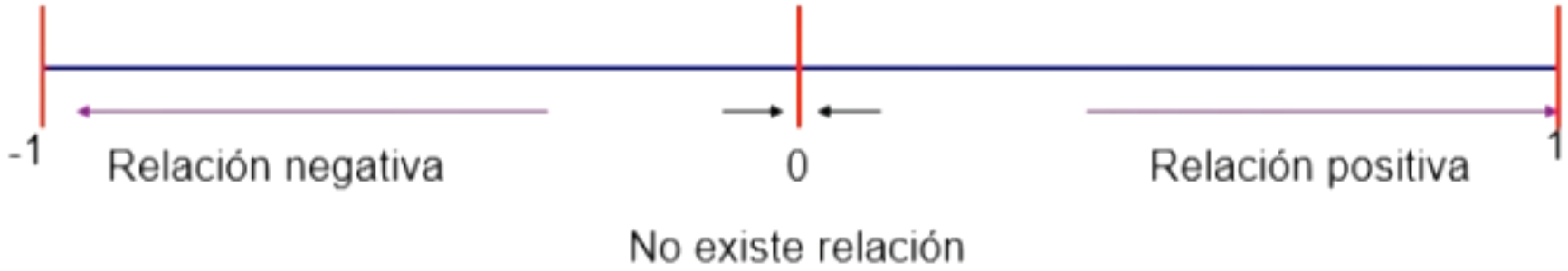
Neutral

Satisfecho

Muy satisfecho

Correlación entre variables

El análisis de correlación pe3rmite medir la intensidad de la relación que puede existir entre dos variables



$$-1 \leq r_{xy} \leq 1.$$

$$r_{xy} = \frac{s_{xy}}{s_x s_y} = \frac{s_{xy}}{\sqrt{s_x^2 s_y^2}}$$

Correlación entre variables

Valores de r	Tipo y grado de correlación
-1	Negativa perfecta
$-1 < r \leq -0.8$	Negativa fuerte
$-0.8 < r < -0.5$	Negativa moderada
$-0.5 \leq r < 0$	Negativa débil
0	No existe
$0 < r \leq 0.5$	Positiva débil
$0.5 < r < 0.8$	Positiva moderada
$0.8 \leq r < 1$	Positiva fuerte
1	Positiva perfecta

Hipótesis Nula (H_o) : No existe relación entre las variables

$$H_o: \rho = 0$$

Hipótesis Alterna (H_1) : Existe relación entre las variables

$$H_1: \rho \neq 0$$

Correlación entre variables

Cuando existen muchas dimensiones en los datos queremos saber cómo se relacionan las variables.

Usar graficas de tipo Scatter es un camino muy rápido para visualizar la correlación entre variables de manera muy rápida

Solo se detecta la parte lineal de las relaciones entre variables

Trabajar con datos faltantes

Borrar toda la fila o columna

Sustituirla por la media de la variable

Borrar la fila o columna cuando hay un bajo % de datos faltantes

En variables no numéricas sustituir el valor faltante por el dato más frecuente

Trabajar con datos atípicos

Valor que estas 3 desviaciones estándar de la media, y se tienen algunas opciones:

1. Borrar la línea cuando exista un determinado % de datos faltantes
2. Sustituirlo por la media
3. Aplicar filtros de ruido

Resumen

- Series de datos y DataFrames
- Manipulación de datos: Selección, filtrado y ordenamiento
- Limpieza de datos: corrección de errores
- Imputación de valores faltantes
- Detección de autliers
- Matplotlib
- Seaborn
- Estadisticas basicas y distribuciones de probabilidad

Series de tiempo

Un conjunto de observaciones tomadas
a través del tiempo

Descriptivo:

Permite entender las características de la serie de tiempo

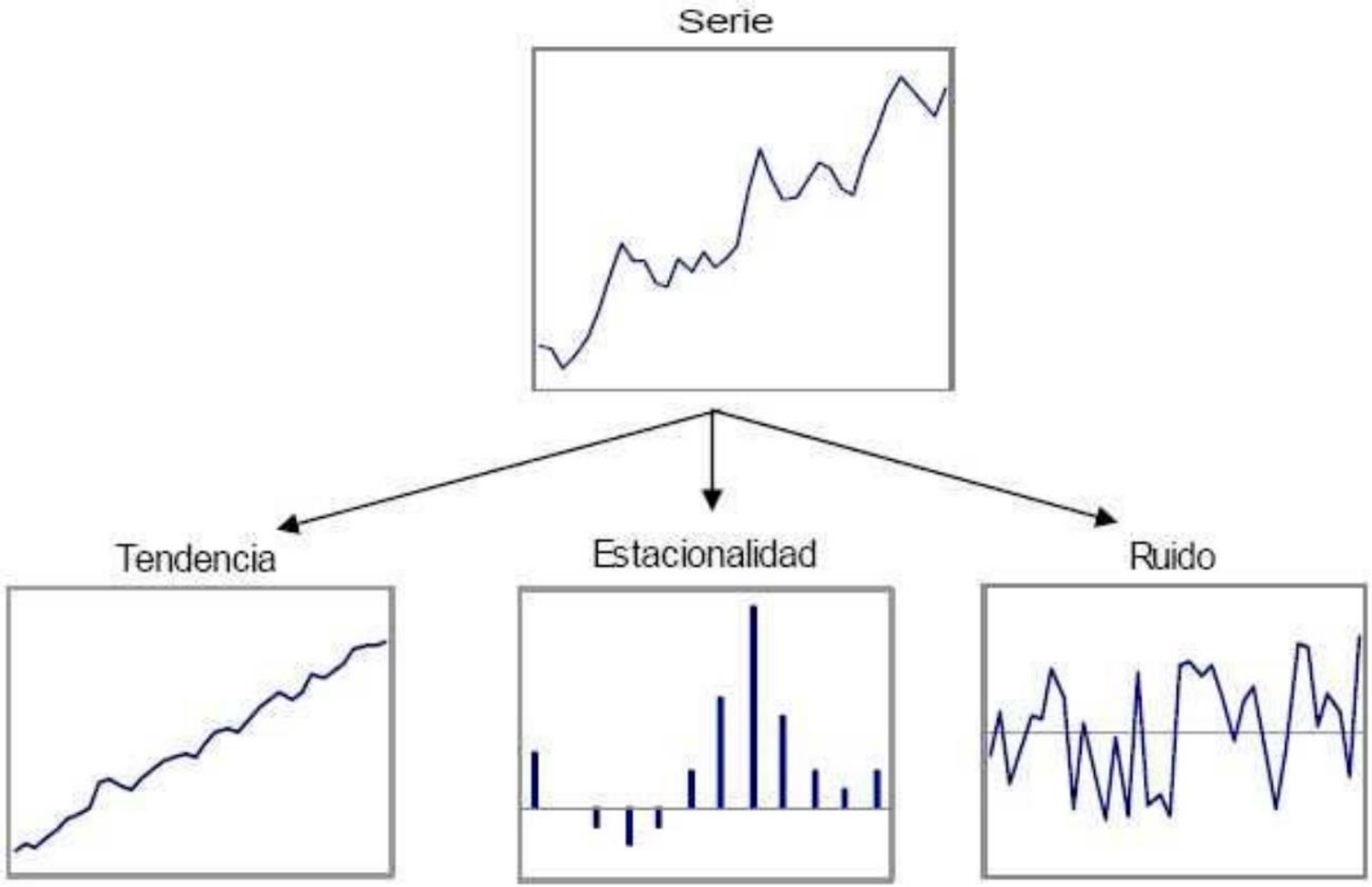
Explicativo/inferencial

Explicar las razones del comportamiento de la serie, relación causa efecto

Predictivo

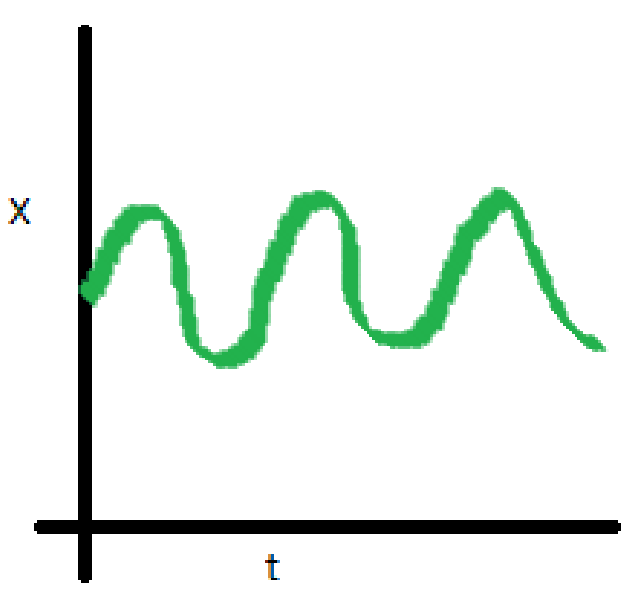
Se utiliza el histórico para predecir en el tiempo

Series de tiempo

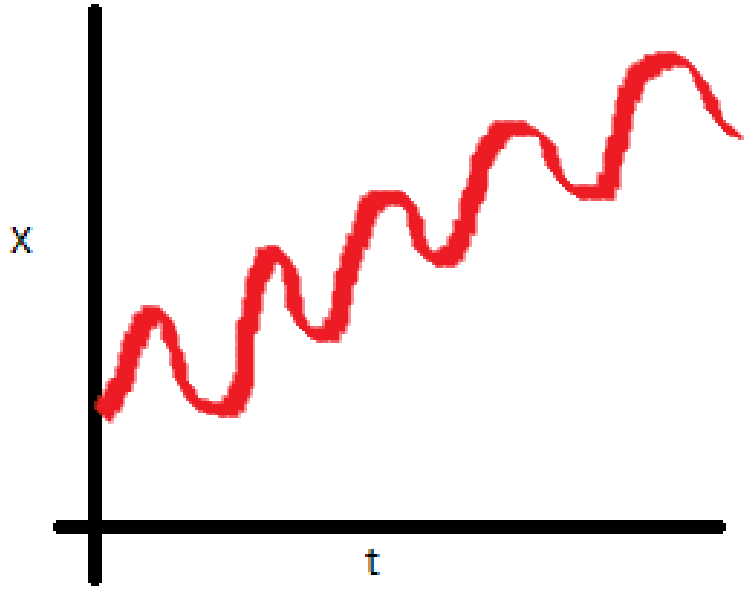


Series de tiempo

ESTACIONARIA



Stationary series



Non-Stationary series

Escalar, Normalizar y Estandarizar

Escalado

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

Normalizar

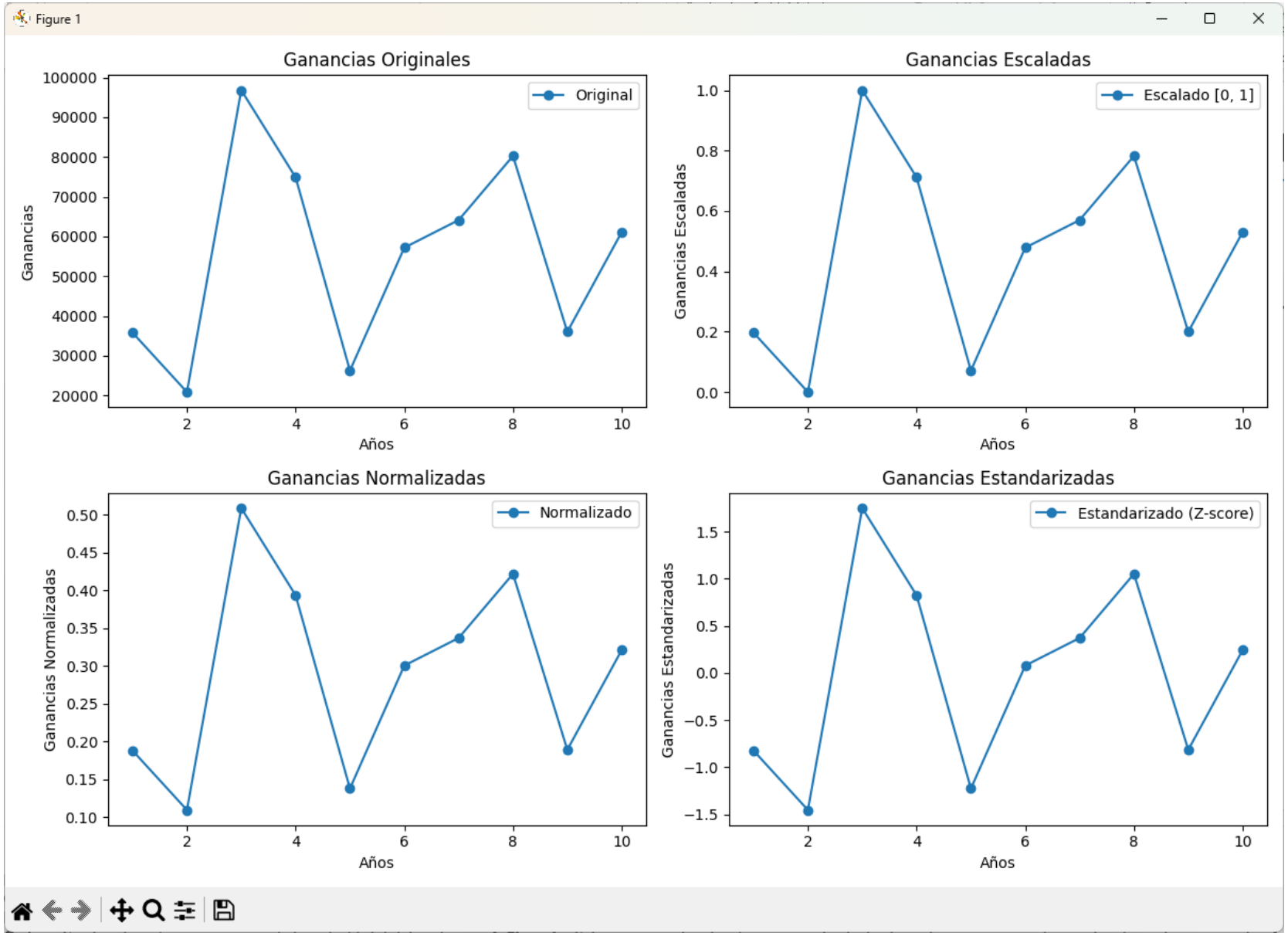
$$x = \frac{x}{||x||}$$

Estandarizar

$$x = \frac{x - \mu}{\sigma}$$

Cada técnica tiene sus aplicaciones y se elige en función del algoritmo y del contexto del problema que se esté resolviendo.

Escalar, Normalizar y Estandarizar



Redes neuronales recurrentes

Las redes neuronales recurrentes están diseñadas para trabajar con datos secuenciales

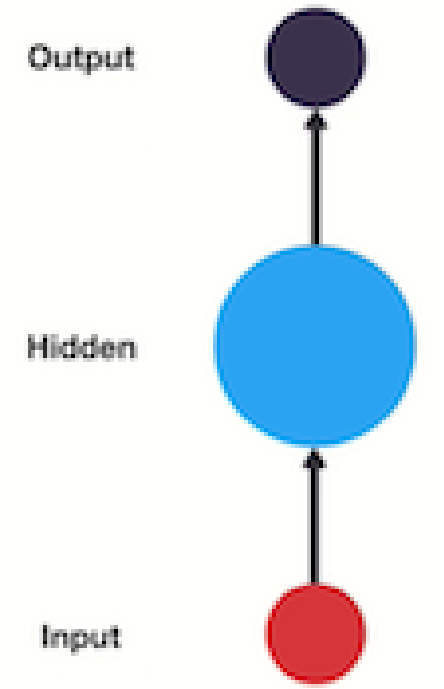
Series temporales

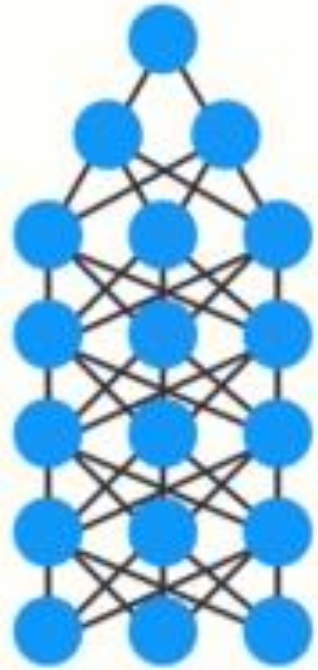
RNN utiliza la información anterior en la secuencia para producir la salida actual.

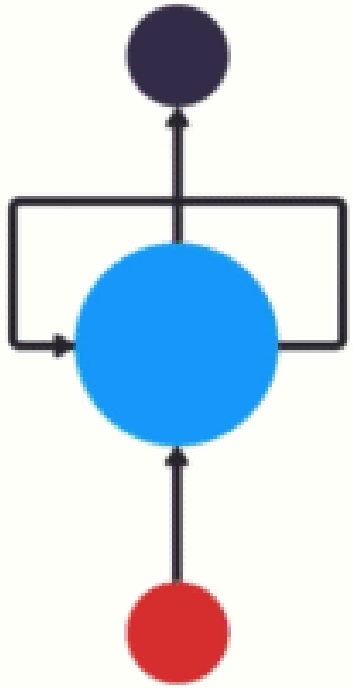
Redes neuronales recurrentes

La RNN tiene toda la información de la secuencia de entrada

Redes neuronales recurrentes



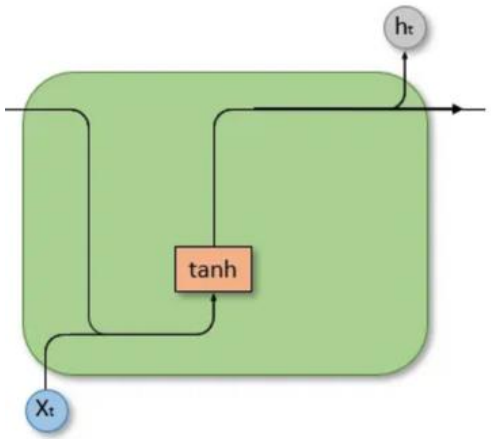




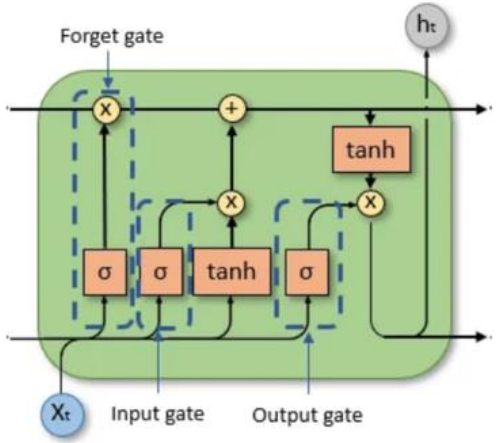
Redes neuronales recurrentes

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

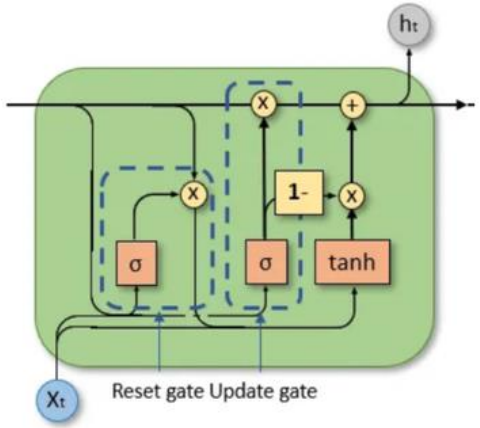
RNN



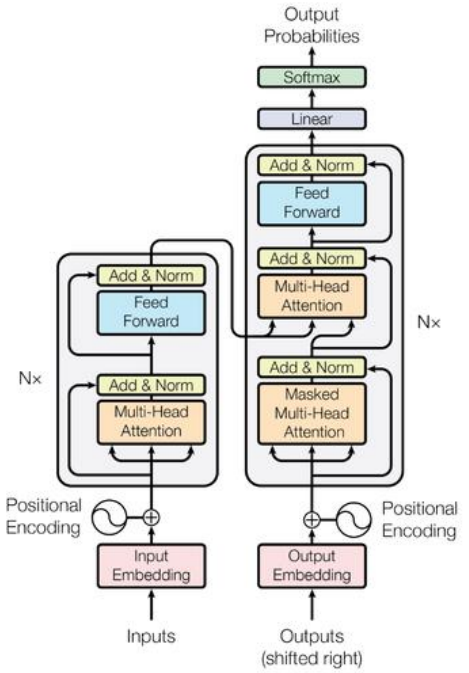
LSTM



GRU

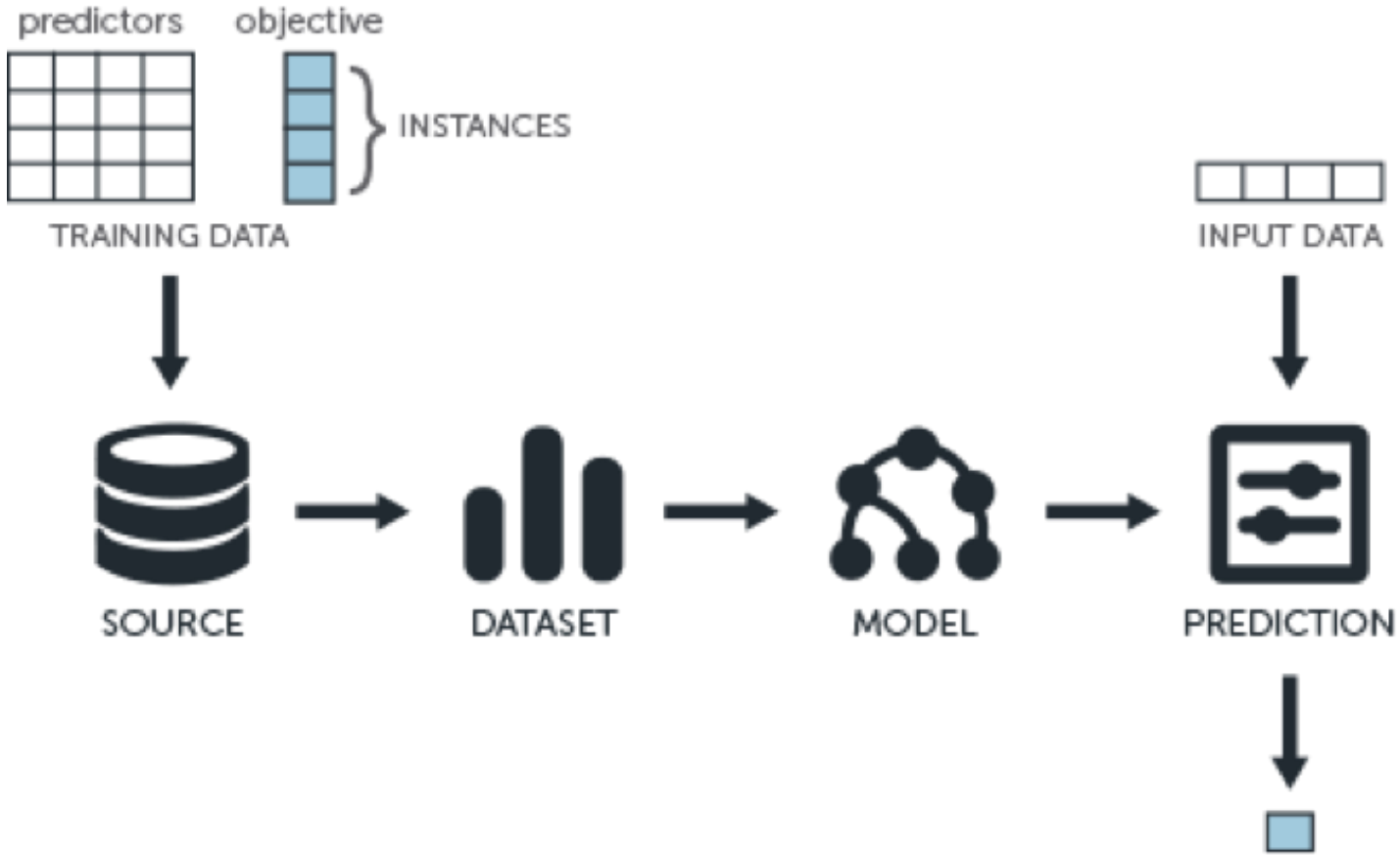


Transformers



Técnicas de Regresión y PCA

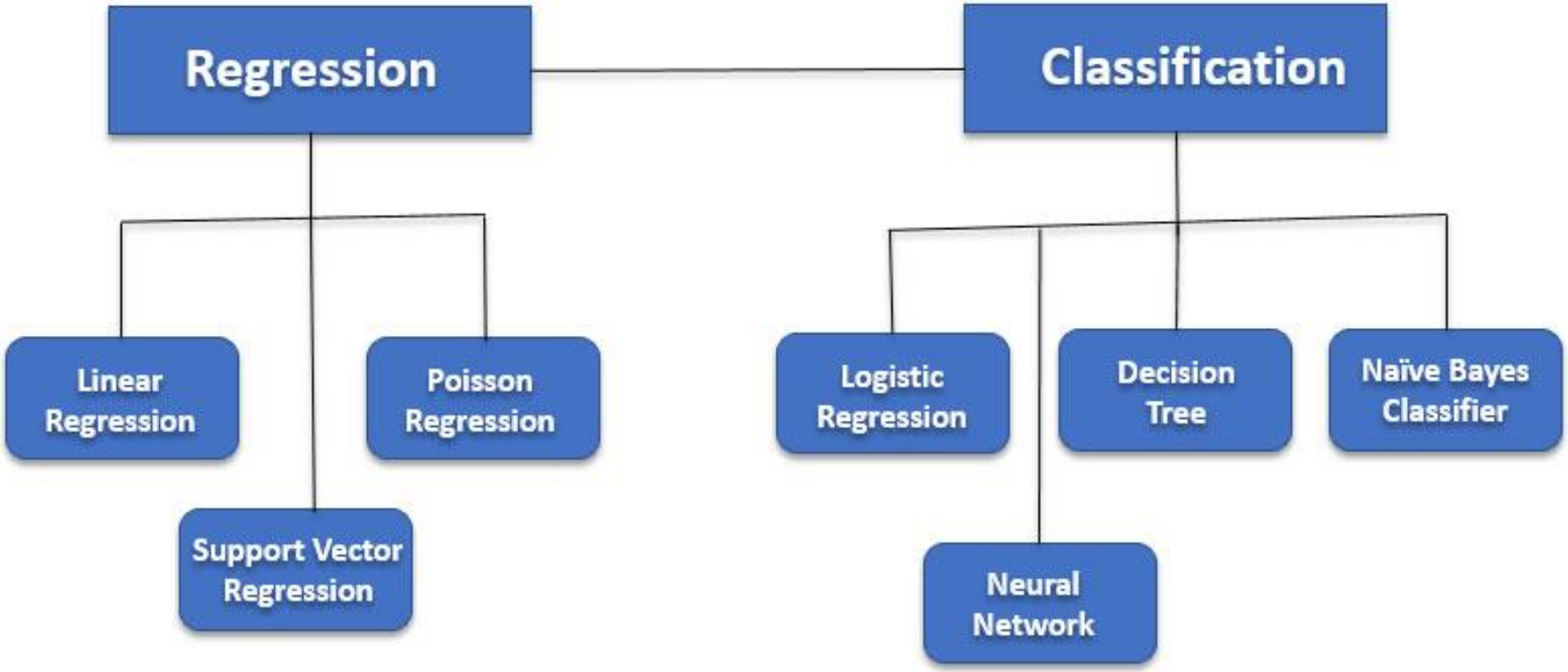
Regresión lineal y no lineal: Regresión y predicción



Determinar si una variable X (o más probablemente, un conjunto de variables $X_1,\ldots,X_p$) está asociada a una variable Y. Si existe tal asociación, se busca entender cuál es esa relación y si podemos utilizarla para predecir Y.

Aprendizaje supervisado

El proceso de entrenar un modelo sobre datos donde se conoce el resultado, para su posterior aplicación a datos en los que no se conoce el resultado, se denomina **aprendizaje supervisado**. Los algoritmos se pueden dividir en casos de predicción y clasificación:



Regresión lineal

La regresión lineal es el enfoque más simple para el aprendizaje supervisado y útil para predecir una respuesta cuantitativa.

Los métodos más usados son:

- Regresión Lineal Simple (SLR)
- Regresión Lineal Múltiple (MLR)
- Regresión de Componentes Principales (PCR)
- Regresión de Mínimos Cuadrados Parciales (PLSR)

Regresión lineal simple

La regresión lineal simple supone que existe aproximadamente una relación lineal entre X e Y, matemáticamente se escribe cómo:

$$Y = b_0 + b_1 X$$

Donde Y es el objetivo y X el vector de características, b_0 y b_1 son dos constantes desconocidas que representan el **intercepto** (o constante) y **pendiente** en el modelo lineal, respectivamente. b_0 y b_1 se llama **coeficientes** o **parámetros** del modelo.

El objetivo de la calibración es encontrar estimaciones de b_0 y b_1 utilizando los datos de entrenamiento.

Estimación de coeficientes

Sea $(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)$ un conjunto de n pares de observaciones, nuestro objetivo será encontrar una intersección y pendiente tal que la línea resultante esté lo más cerca posible de las n observaciones. La manera más común de medir la cercanía implica minimizar el criterio de **mínimos cuadrados.**

Sea $\hat{y}_i = \hat{b}_0 + \hat{b}_1 x_i$ la predicción de Y basada en el i-ésimo valor de X. Entonces $e_i = y_i - \hat{y}_i$ representa el i-ésimo residual, por lo tanto, la suma de residuales cuadrados (RSS) es:

$$RSS = e_1^2 + e_2^2 + \cdots + e_n^2$$

O lo que es lo mismo:

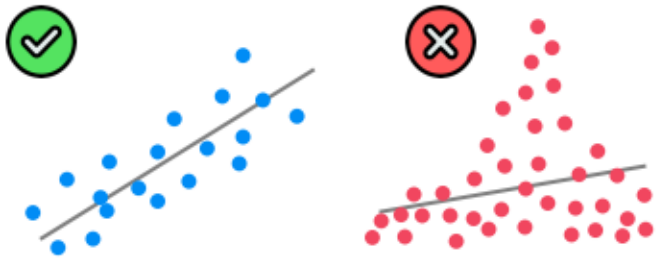
$$RSS = \left(y_1 - \hat{b}_0 - \hat{b}_1 x_1\right)^2 + \left(y_2 - \hat{b}_0 - \hat{b}_1 x_2\right)^2 + \cdots + \left(y_n - \hat{b}_0 - \hat{b}_1 x_n\right)^2$$

Con algunos cálculos se puede demostrar que los valores de los coeficientes que minimizan RSS son:

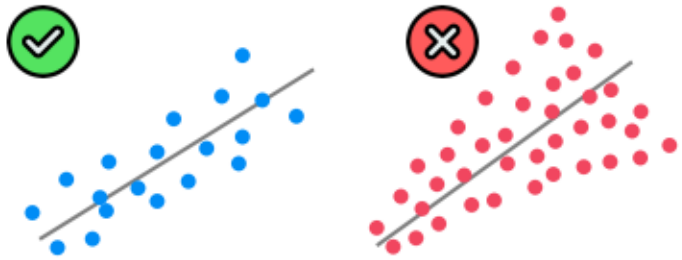
$$\hat{b}_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \hat{b}_0 = \bar{y} - \hat{b}_1 \bar{x}$$

Supuestos

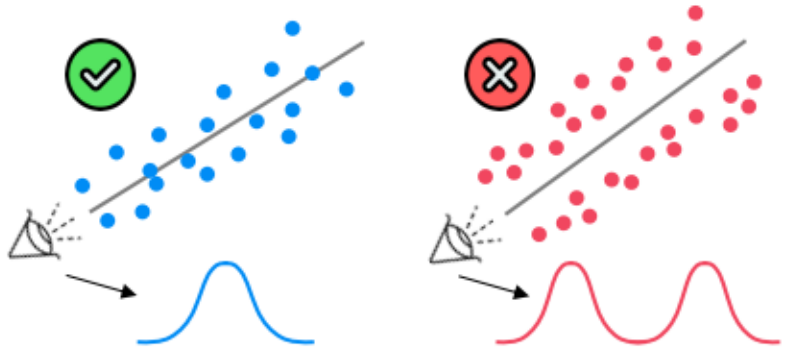
1. Linearity
(Linear relationship between Y and each X)



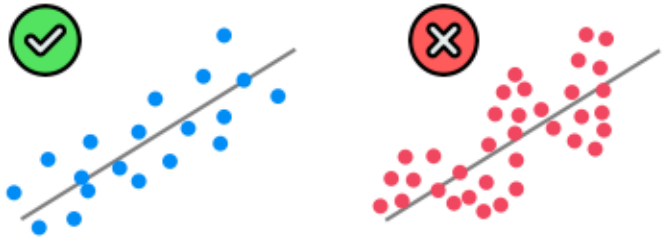
2. Homoscedasticity
(Equal variance)



3. Multivariate Normality
(Normality of error distribution)



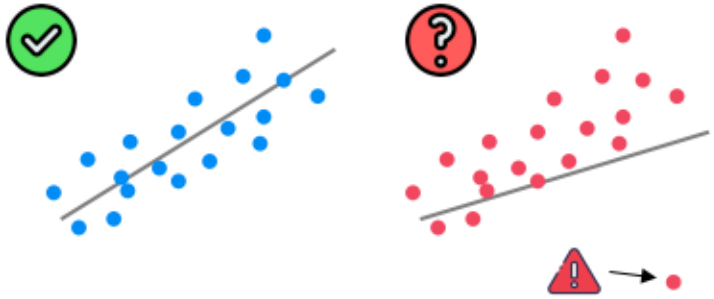
4. Independence
(of observations. Includes “no autocorrelation”)



5. Lack of Multicollinearity
(Predictors are not correlated with each other)

$X_1 \nsim X_2$ $\quad X_1 \sim X_2$

6. The Outlier Check
(This is not an assumption, but an “extra”)

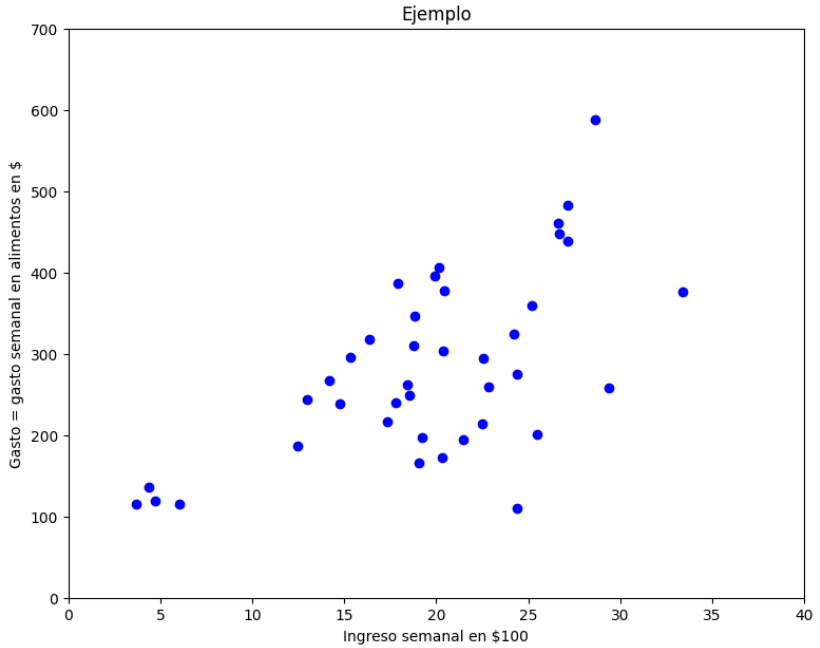


Ejemplo

Consideremos el conjunto de datos de la imagen que muestra niveles de ingresos semanales y niveles de gasto semanal en alimentos. Podemos representar X como el ingreso y Y el gasto en alimentos:

$$gasto = b_0 + b_1 * ingreso$$

Usamos las expresiones de Ec.1 para calcular los coeficientes y verificamos los resultados usando `LinearRegression` de `sklearn.linear_model`:



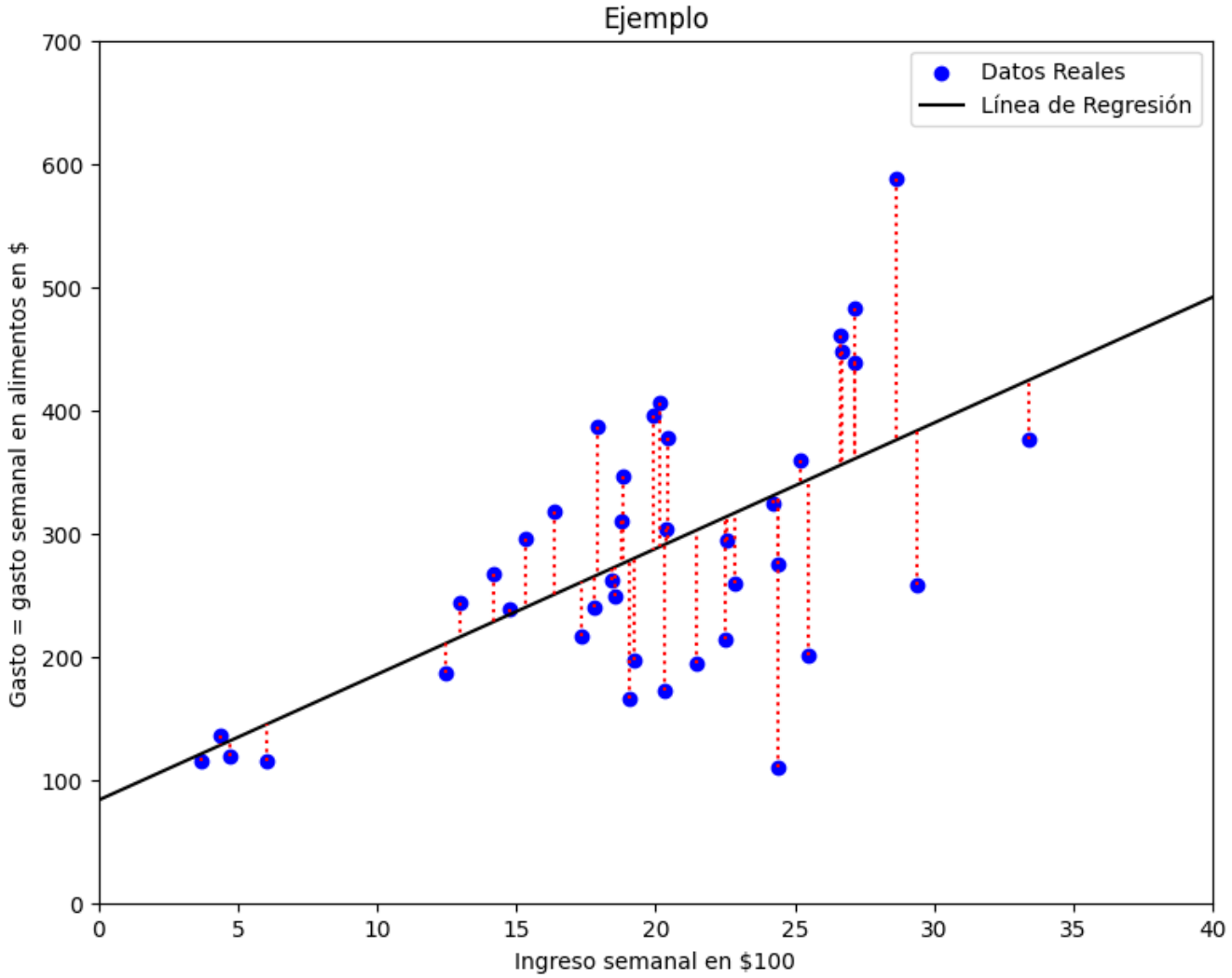

```python
import numpy as np
from sklearn.linear_model import LinearRegression
#Datos
X = ingreso
Y = gasto
# Calcular las medias de X e Y
mean_X = np.mean(X)
mean_Y = np.mean(Y)
# Calcular b1 (pendiente)
numerator = np.sum((X - mean_X) * (Y - mean_Y))
denominator = np.sum((X - mean_X) ** 2)
b1 = numerator / denominator
# Calcular b0 (intercepto)
b0 = mean_Y - (b1 * mean_X)
print(f'Coeficiente de pendiente (b1): {b1}')
print(f'Intercepto (b0): {b0}')
# Usar LinearRegression
X_reshaped = X.reshape(-1, 1)
model = LinearRegression().fit(X_reshaped, Y)
b1_sklearn = model.coef_[0]
b0_sklearn = model.intercept_
print(f'Coeficiente de pendiente con scikit-learn (b1):
{b1_sklearn}')
print(f'Intercepto con scikit-learn (b0): {b0_sklearn}')
```

Ejemplo

La gráfica muestra los mínimos cuadrados ajustados para la regresión del gasto en alimentos en función del ingreso semanal.
Cada segmento rojo representa un residual.

Coeficiente de pendiente (b1): 10.209
Intercepto (b0): 83.416

Coeficiente de pendiente con scikit-learn (b1): 10.209
Intercepto con scikit-learn (b0): 83.416



Regresión Lineal Múltiple

En la práctica a menudo tenemos más de un predictor por lo que un mejor enfoque es extender el modelo SLR para manejar múltiples predictores y se extiende también el concepto de ajuste por mínimos cuadrados. Matemáticamente toma la forma de:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_p X_p + \epsilon$$

El modelo de MLR se representa más fácilmente en forma matricial. Dada una muestra de n observaciones de la variable objetivo $(y_1, \ldots, y_n)$ y de las variables predictoras $(x_{11}, \ldots, x_{n1})$, $(x_{12}, \ldots, x_{n2})$,..., $(x_{1p}, \ldots, x_{np})$, el modelo se puede escribir como:

$$Y = \begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \cdots & \cdots & \cdots & \cdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \cdots \\ \beta_p \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ \cdots \\ e_n \end{pmatrix} = X\beta + \epsilon$$

Estimación de coeficientes

La estimación de coeficientes se basa en los mismos principios que RLS. **Sin embargo, para casos donde las variables predictoras no vengan medidas en la misma escala de medida**, el modelo obtenido (es decir, los coeficientes del modelo) exhibe una dependencia de dicha escala que puede provocar que una variable con una variabilidad pequeña aparezca con un coeficiente grande en el modelo estimado.

Para evitar esa dependencia se suele trabajar con las **variables estandarizadas**, es decir, corregidas por su media y desviación típica para eliminar los efectos de escala.

El conjunto de coeficientes β que minimiza la suma de errores cuadrados en el conjunto de entrenamiento se puede calcular como:

$$\widehat{\boldsymbol{\beta}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}\left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{Y}\right)$$

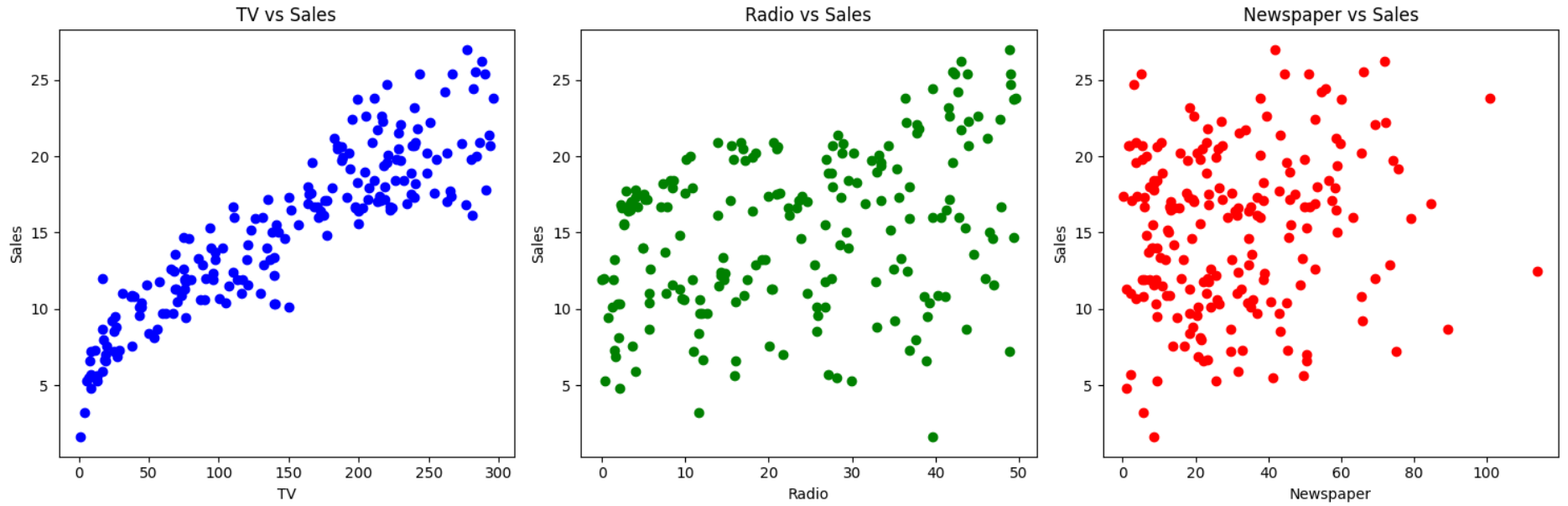
Para una nueva muestra, la variable objetivo se obtiene con el modelo lineal ajustado como:

$$\widehat{\boldsymbol{y}} = \boldsymbol{X}\widehat{\boldsymbol{\beta}}$$

Ejemplo

Consideremos un conjunto de datos que consta de ventas (en miles de unidades) de un producto en 200 mercados diferentes, junto con presupuestos de publicidad (en miles de dólares) para el producto en tres medios: TV, radio y periódico. Con un modelo de MLR podríamos utilizar los presupuestos de publicidad para predecir ventas.

Datos: https://www.kaggle.com/datasets/ashydv/advertising-dataset/data



Ejemplo

Cargamos lo datos de presupuesto para publicidad en la matriz X y ventas en la matriz Y. Ajustamos un modelo manualmente y lo comparamos con `LinearRegression.`

Ajuste con `LinearRegression`

```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
model = lm.fit(X,y)

# Mostrar los coeficientes
print(f'Coeficiente beta_1 (pendiente):
{model.coef_}')
print(f'Coeficiente beta_0 (intercepto):
{model.intercept_}’)
```

```
Coeficiente beta_1 (pendiente): [0.05444578
0.10700123 0.00033566]
Coeficiente beta_0 (intercepto): 4.625124078808653
```

Ajuste manual

```
# Añadir una columna de unos a X para el término de
intercepción
X = np.c_[np.ones(X.shape[0]), X]  # Añadir columna de
unos al principio
# Convertir a matrices numpy
X = np.array(X)
Y = np.array(y).reshape(-1, 1)
# Calcular (X^T X)^{-1} (X^T Y)
X_transpose = X.T
X_transpose_X = np.dot(X_transpose, X)
X_transpose_X_inv = np.linalg.inv(X_transpose_X)
X_transpose_Y = np.dot(X_transpose, Y)
beta = np.dot(X_transpose_X_inv, X_transpose_Y)
# Imprimir los coeficientes
print("Coeficientes beta:")
print(beta)
```

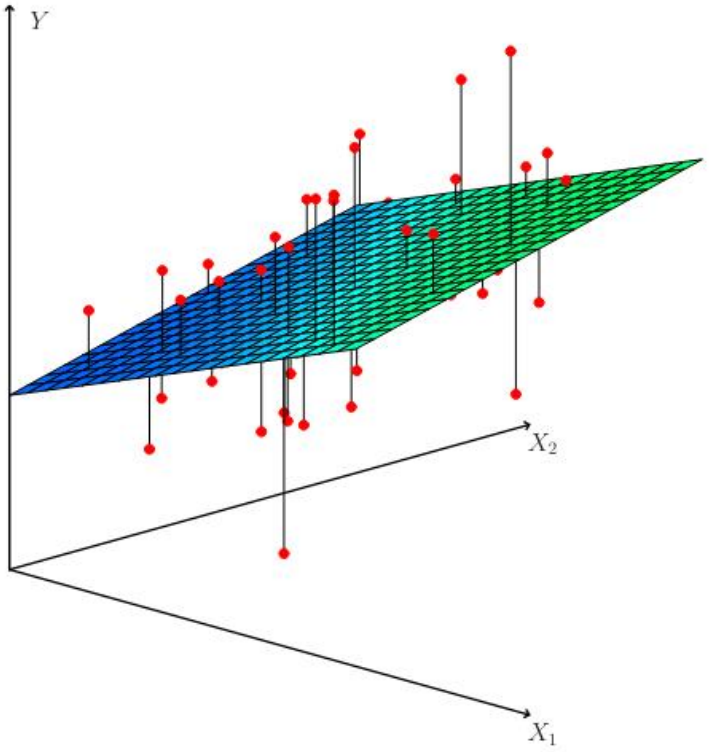
```
Coeficientes beta: [[4.62512408e+00] [5.44457803e-02]
[1.07001228e-01] [3.35657922e-04]]
```

Predicciones

Tanto en SLR como en MLR para hacer predicciones usamos `model.predict` con nuevos datos en el **mismo formato** que se usó para el **entrenamiento**.

```
new_data = [[30,10,45]]
model.predict(new_data)
array([4.89219873])
```

Con un presupuesto de 30, 10 y 45 miles de dólares en TV, radio y periódico, respetivamente, se esperarían ventas de 4892 unidades del producto.



En un modelo con dos predictores y una respuesta la línea de regresión se convierte en un plano que minimiza la suma de distancias verticales al plano (en rojo).
En un modelo con 3 predictores se complica la visualización de línea de regresión.

Multicolinealidad

La multicolinealidad es un problema que surge cuando las variables explicativas del modelo están altamente correlacionadas entre sí. Este es un problema complejo, porque en cualquier regresión las variables explicativas van a presentar algún grado de correlación.

Se dice que existe multicolinealidad cuando tenemos problemas a la hora de invertir la matriz $\boldsymbol{X^T X}$:

- Si $\left|\boldsymbol{X^T X}\right| = 0$ existe **multicolinealidad exacta.** Una variable es combinación lineal exacta de otras y tendrá que eliminarse dicho regresor.
- Si $\left|\boldsymbol{X^T X}\right| \approx 0$ existe **multicolinealidad de grado.** Alguna variable está altamente correlacionada con otra(s), pero el sistema de ecuaciones normales tiene una única solución.

Las varianzas y covarianzas estimadas de los parámetros se hacen muy grandes conforme aumenta el grado de colinealidad, es decir:

$$var\hat{(}\hat{\beta}) = \frac{Adj(X^T X)}{|X^T X|}$$

Al ser del determinante cercano a 0, se inflan las varianzas y covarianzas de los parámetros estimados, lo que implica que **la precisión de la estimación disminuye** a medida que aumenta la colinealidad.

Componentes principales

Una forma de evitar la multicolinealidad es utilizar la PCR, que captura la mayor parte de la variación presente en un conjunto de p variables correlacionadas con un número menor a de combinaciones lineales (ortogonales) no correlacionadas de las variables originales (componentes principales), ordenados por la cantidad de varianza explicada.

Con los componentes principales (PC) se utiliza regresión lineal para modelar la **relación** entre estos **componentes** y la **variable objetivo,** en lugar de usar las variables predictoras originales. Matemáticamente:

$$\boldsymbol{c = \left(T^T T\right)^{-1}\left(T^T Y\right)}$$

Donde *c* representa los coeficientes de regresión en el espacio de puntuaciones de PC y T representa la matriz de variables predictoras con las puntuaciones de los primeros a PC.

Predicción con PCR

Para hacer predicciones con un nuevo conjunto de datos X_{new}, proyectamos los datos estandarizados Z_{new} sobre los vectores propios V_k para obtener los componentes principales:

$$PC_{new} = Z_{new}V_k$$

Finalmente, utilizamos el modelo de regresión:

$$\hat{y}_{new} = PC_{new} * c$$

En Python podemos utilizar las siguientes librerías, para estandarizar datos, hacer análisis de componentes principales y combinarlo con regresión lineal, respectivamente :

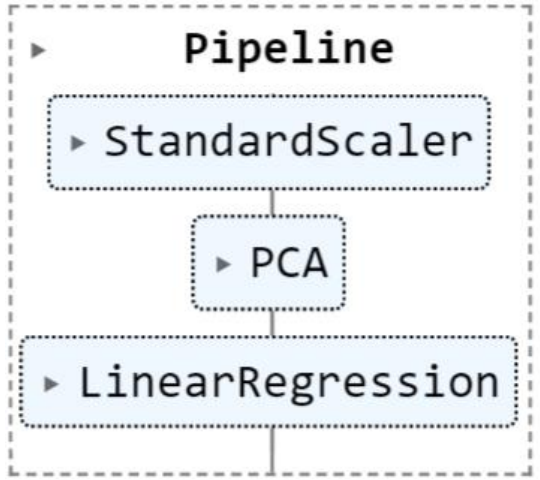
- `from sklearn.preprocessing import StandardScaler`
- `from sklearn.decomposition import PCA`
- `from sklearn.pipeline import Pipeline`

Pipeline permite ensamblar los pasos para el modelado en una única secuencia asegurando la consistencia con los datos usados, por ejemplo, realiza el escalamiento para el entrenamiento y la predicción automáticamente.

PCR con python

Por default la función PCA calcula todos los componentes principales, pero se puede calcular el número óptimo con algún método de validación.

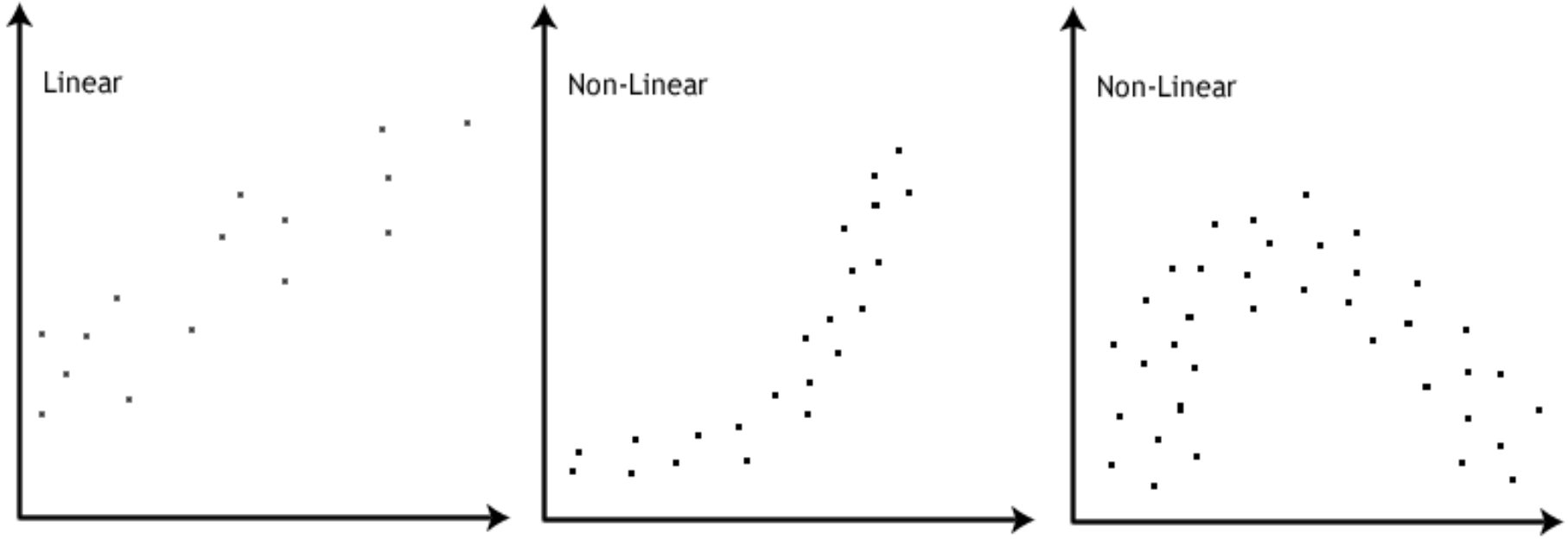
```
# Entrenamiento modelo de regresión precedido por PCA
con estandarización
pcr_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA()),
    ('regressor', LinearRegression())
])
pcr_pipeline.fit(X, y)
# Predicciones
pcr_pipeline.predict(new_data)
```



De la misma manera, podemos usar `.predict()` para realizar predicciones.

Regresión no lineal

Si el patrón subyacente en los datos muestra una curva, ya sea de crecimiento exponencial, decrecimiento, logarítmica o cualquier otra forma no lineal, ajustar un modelo de regresión no lineal puede proporcionar una representación más precisa de la relación.



Regresión no lineal

La manera más sencilla de extender directamente el modelo lineal para acomodar relaciones no lineales es usando regresión polinómica. El enfoque consiste en incluir versiones transformadas de los predictores.

$$Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \ldots + \beta_k X^k + \epsilon$$

El modelo implica predecir la variable objetivo usando una función no lineal de los predictores, **pero sigue siendo un modelo lineal.**

Podemos utilizar Python para estimar los coeficientes de regresión a fin de producir un ajuste no lineal.

Además, las funciones pueden tomar diversas formas, como exponenciales, logarítmicas, sigmoideas, etc.

Regresión no lineal en Python

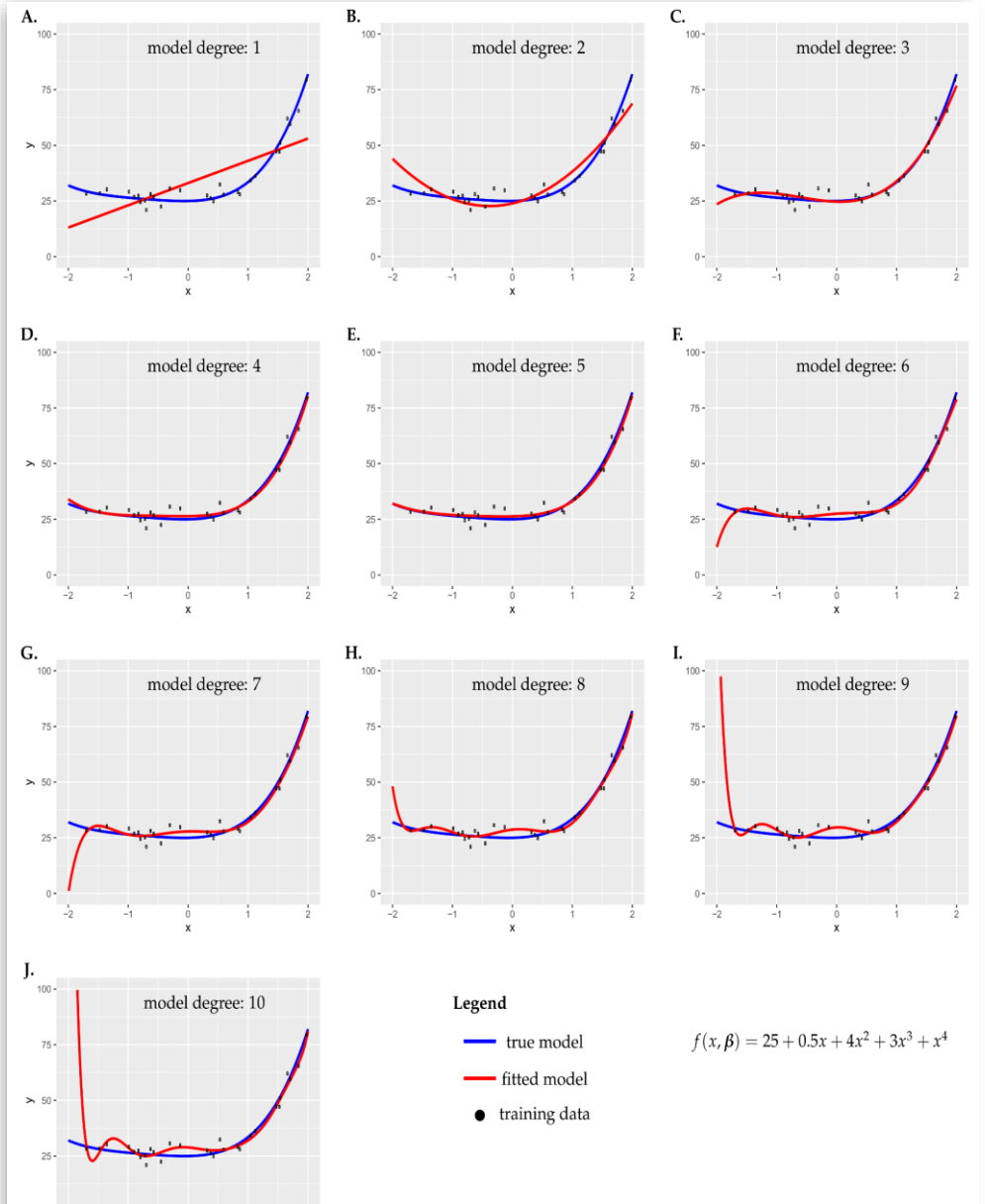
Podemos utilizar la biblioteca SciPy, que tiene una interfaz sencilla e intuitiva para ajustar modelos de regresión no lineal.

Simplemente podemos definir una función no lineal y usarla como argumento en la función `curve_fit`:

```
# Definir la función no lineal
def quadratic_func(x, a, b, c):
    return a + b * x + c * x**2
# Ajustar el modelo no lineal
popt, pcov = curve_fit(quadratic_func, X, y)
```

Usando los parámetros resultantes de la función, podemos predecir valores de salida para nuevos datos:

```
y_new = quadratic_func(x_new, *popt)
```



Ejemplo de modelos de regresión polinomial de diferente grado, ajustados con datos generados a partir de un modelo real.

Modelos no lineales

La regresión también incluye modelos no lineales que producen una relación funcional entre los predictores y las variables resultado. Es decir, cuando los predictores y la respuesta siguen una forma de función particular:

$$y = f(\beta, x) + \varepsilon$$

Por ejemplo:

Modelo no lineal

$$y = \beta^2 x + \varepsilon$$

$$y = \frac{1}{\beta} x + \varepsilon$$

$$y = e^{\beta x} + \varepsilon$$

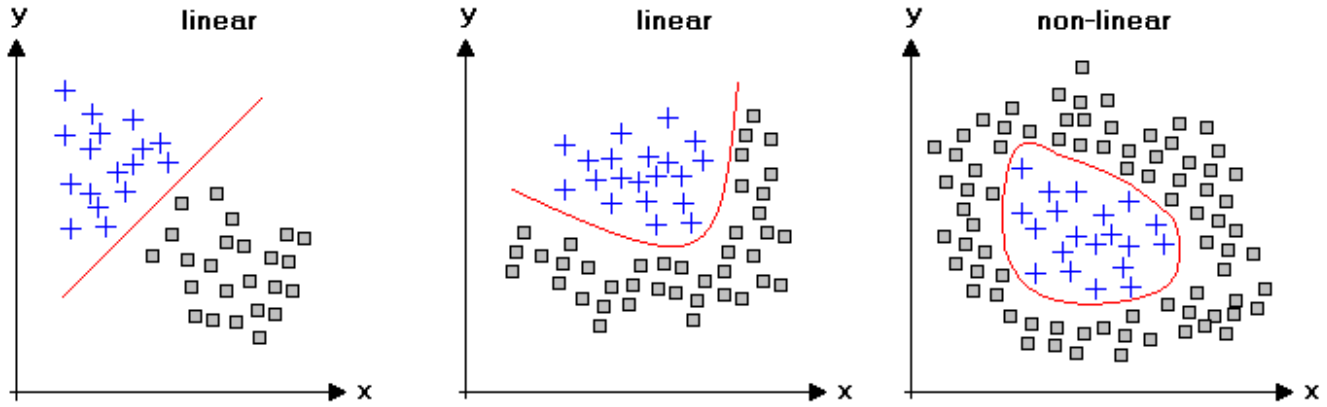
$$y = \frac{1}{1 + \beta x} + \varepsilon$$

Modelo lineal

$$y = \beta x^2 + \varepsilon$$

$$y = \beta \frac{1}{x} + \varepsilon$$

$$y = \beta \, lnx + \varepsilon$$



Estos modelos son más difíciles y computacionalmente más intensivos de ajustar porque requiere optimización numérica, por ello, generalmente se prefiere usar un modelo lineal si es posible.

Evaluación de modelos de regresión

Para decidir sobre la complejidad del modelo es importante cuantificar su rendimiento. Una métrica de rendimiento en las mismas unidades de la variable objetivo, normalmente se usa el error cuadrático medio (RMSE):

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

Donde

n es el número de muestras.

$\hat{y}_i, y_i$ son respectivamente el valor y predicho y medido.

Esto se puede calcular en función del conjunto de calibración (RMSEC), el conjunto de validación (RMSEV), una validación cruzada (RMSECV) o del conjunto de prueba (RMSEP).

Cuantificación del rendimiento

El error sistemático o promedio se denomina sesgo y se puede calcular como:

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

El error aleatorio o estándar se denomina error estándar de predicción:

$$SEP = \sqrt{\frac{n}{n-1}(RMSE^2 - Bias^2)}$$

La relación entre el RMSE y la desviación estándar (STD), cuantifica cuántas veces el modelo es más preciso al estimar la variable Y, que el valor promedio:

$$RPD = \frac{STD}{RMSE}$$

El coeficiente de determinación R^2, que expresa qué fracción de la variación de los datos Y es capturada por el modelo, se obtiene como:

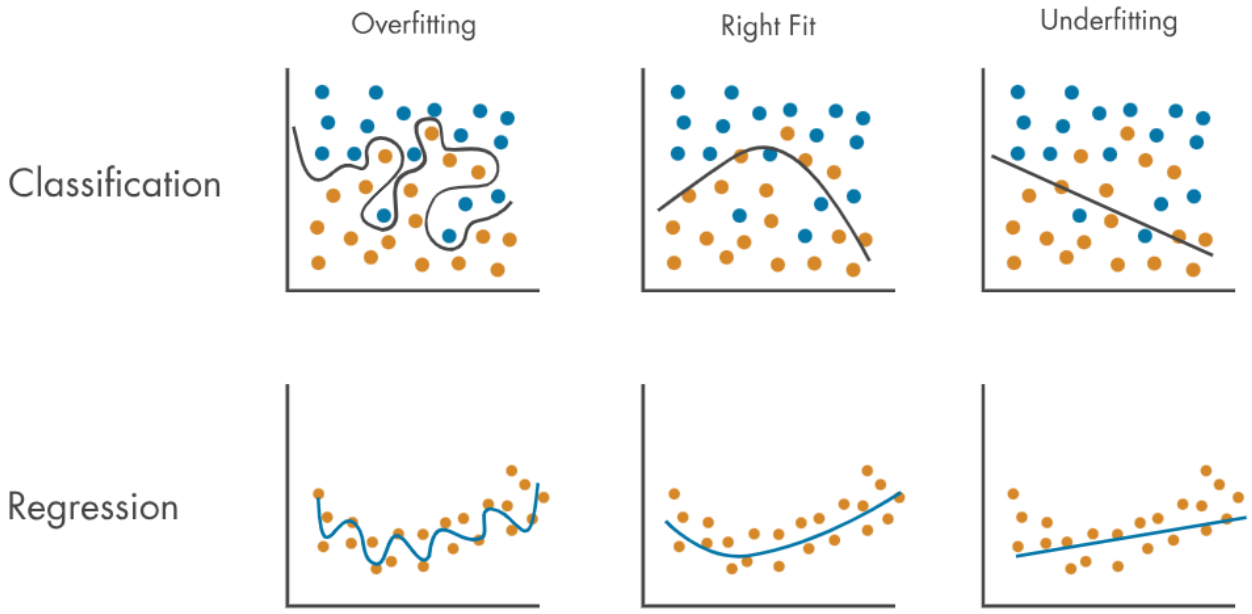
$$R^2 = 1 - \frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{\sum_{i=1}^{n}(\hat{y}_i - \bar{y}_i)^2}$$

Sobreajuste y desajuste

Cuanto mayor es la complejidad de modelo, más poder de discriminación posee, aunque también aumenta el riego de **sobreajuste.**

El sobreajuste es un fenómeno que se observa cuando un modelo entrenado tiene rendimiento extremadamente bueno en las muestras usadas para entrenamiento, pero tiene rendimiento deficiente en muestras desconocidas.

También puede ocurrir que el modelo no se alinee bien con los datos de entrenamiento o no logre aplicar patrones a datos nuevos, ocurriendo así el **desajuste.**



Selección y evaluación

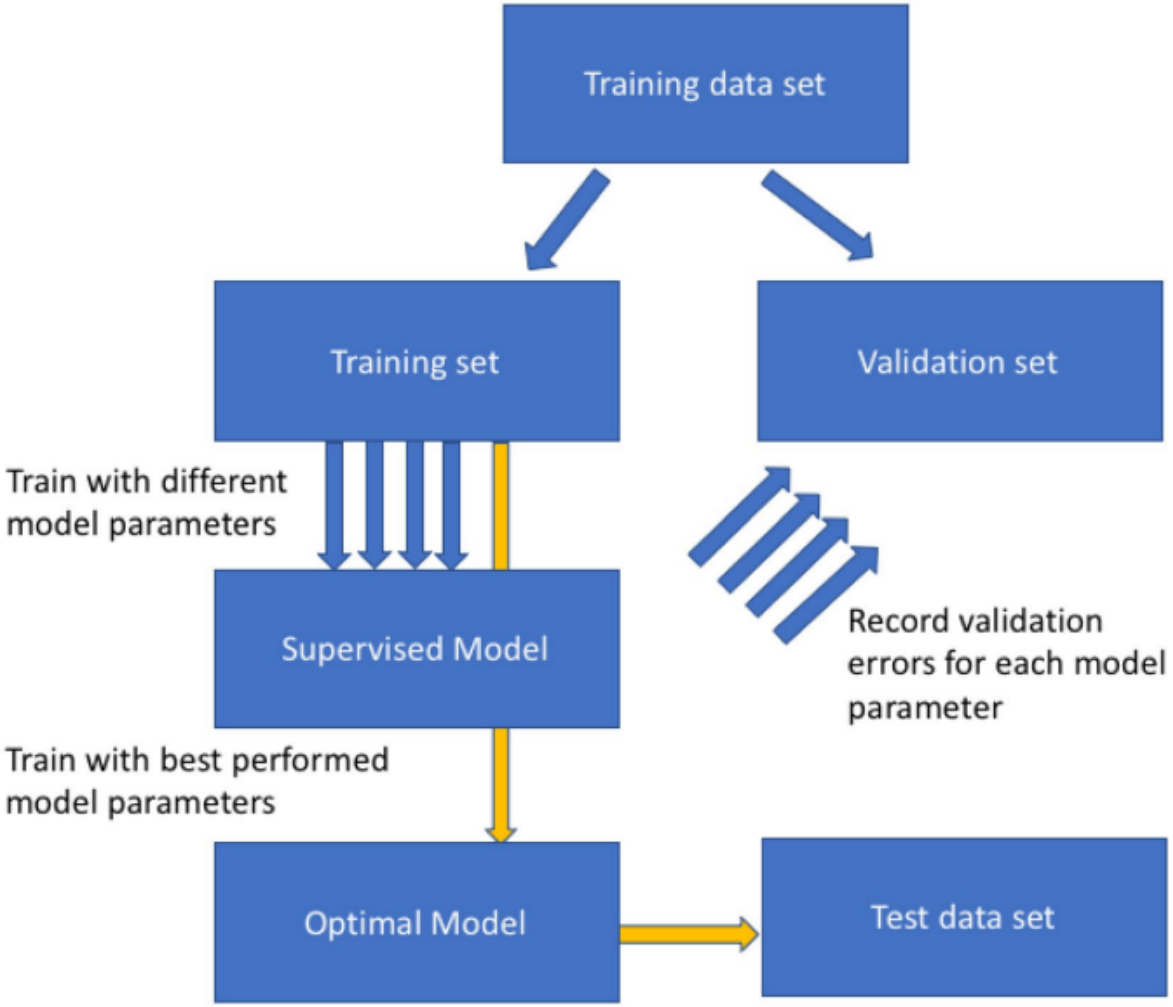
¿Cómo se puede elegir entre dos modelos competidores y cómo se pueden evaluar?

- Selección del modelo: estimar el rendimiento de diferentes modelos para elegir el mejor.
- Evaluación del modelo: para el mejor modelo, estimar su error de generalización.

El mejor enfoque de búsqueda del mejor modelo consiste en dividir aleatoriamente los datos en tres conjuntos que no se superpongan:

1. **Datos de entrenamiento**: para estimar o aprender los parámetros del modelo.
2. **Datos de prueba**: para estimar un criterio de selección del modelo.
3. **Datos de evaluación**: para estimar el error de generalización del modelo.

Selección del modelo



Para tener un **equilibrio entre sobreajuste y desajuste** es necesario dividir los datos en conjuntos de entrenamiento y validación.

En función de los errores en el conjunto de validación, el conjunto de parámetros óptimos del modelo se determina utilizando el que tiene el error de validación más bajo. Este procedimiento se llama **selección del modelo.**

Métodos de división de datos

Es probable que el rendimiento del modelo se vea afectado por factores como el algoritmo de modelado, la superposición entre los datos, la cantidad de muestra disponibles y lo más importante el método utilizado para dividir los datos.

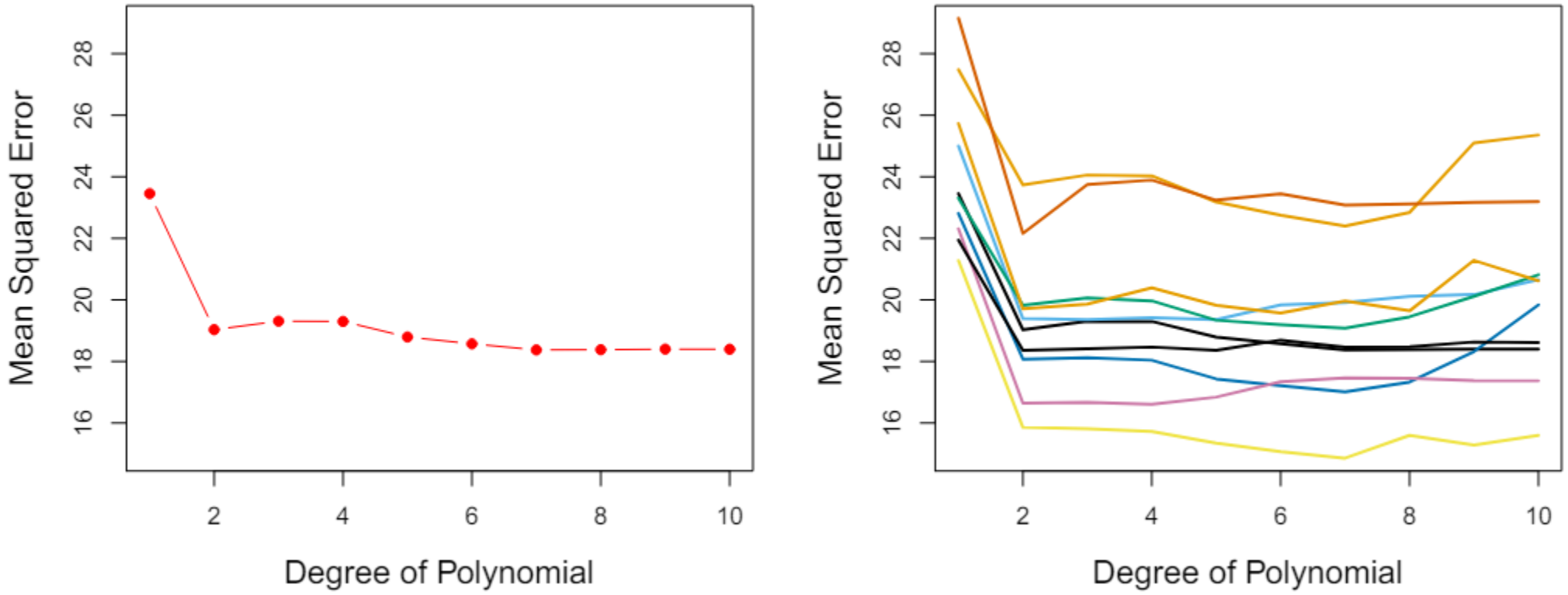
División aleatoria de datos

Este proceso se puede repetir muchas veces y la estimación final del desempeño del modelo se calcula como el desempeño promedio en los conjuntos de validación en todas las repeticiones.

En Python podemos utilizar la función *train_test_split* para dividir datos en subconjuntos de entrenamiento, validación y prueba. El estándar aproximado para las divisiones de train-tes-eval son **60-80%, 10-20% y 10-20%,** respectivamente.

División aleatoria de datos

La siguiente figura muestra el comportamiento del error con un modelo de regresión polinómica de diferentes grados en un conjunto de validación (izquierda) y varios conjuntos de validación (derecha), haciendo división aleatoria de datos.

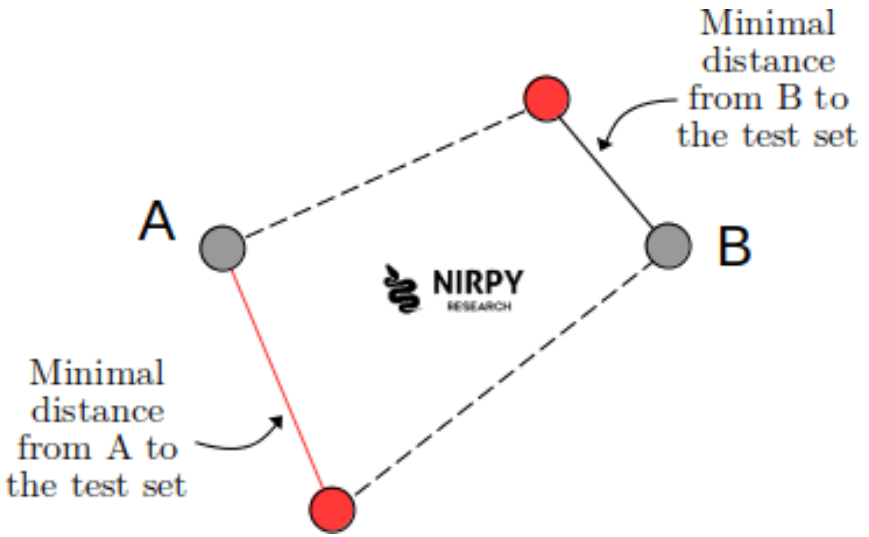


División Kennard-Stone

- **Basados en la distribución de los datos**

Los datos se pueden dividir seleccionando sistemáticamente un número determinado de las muestras más representativas de los conjuntos de datos y usar las restantes para la validación.

El método Kennard-Stone calcula la distancia euclidiana entre cada una de las muestras para agregar al conjunto de entrenamiento las que tengan la mayor distancia (A-B) y separar las de menor distancia para el conjunto de validación o prueba (puntos rojos).



Podemos encontrar esta función en Python y usarla de la siguiente manera:

```
import kennard_stone as ks
X_train, X_test, y_train, y_test = ks.train_test_split(X, y, test_size = 0.2)
```

Validación cruzada (CV)

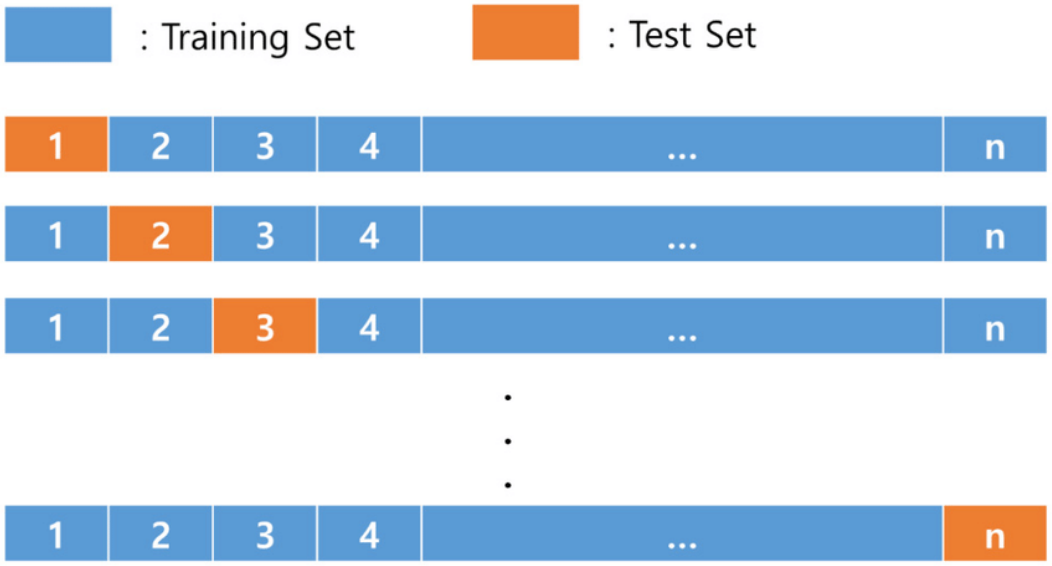
CV es probablemente el método más común de división de datos en la selección del modelo:

1. Divide los datos en *k* partes diferentes (*k* pliegues) y una de esas partes se mantiene como conjunto de validación.
2. El modelo se entrena con las partes *k-1* restantes y se aplica al conjunto de validación.
3. El proceso se repite *k* veces para que cada parte se haya usado como conjunto de validación una vez.
4. Se promedia los rendimientos registrados y se determina el parámetro óptimo como aquel que tuvo el mejor rendimiento promediado.

Validación cruzada (CV)

Este método suele llamarse k-fold CV y un caso especial cuando *k=n* (donde *n* es el número total de muestras) es llamado validación cruzada dejando uno fuera (LOO-CV)

Ejemplo de k-fold CV con k=5



Ejemplo de LOO-CV con *n* muestras

Validación cruzada (CV)

La manera más sencilla de hacer validación cruzada en Python es usando la función `cross_val_score` para evaluar la puntuación mediante validación cruzada.

La función requiere además de los datos, el modelo que se validará y la estrategia de división por CV, es decir k-fold o LOO, importando `KFold`, `LeaveOneOut`, respectivamente. Además, se debe configurar en el parámetro "scoring" la métrica de evaluación para cada pliegue y al final se puede obtener el promedio.

Nota: Por default la métrica que utiliza CV es R^2 en problemas de regresión y **precisión** en problemas de clasificación.

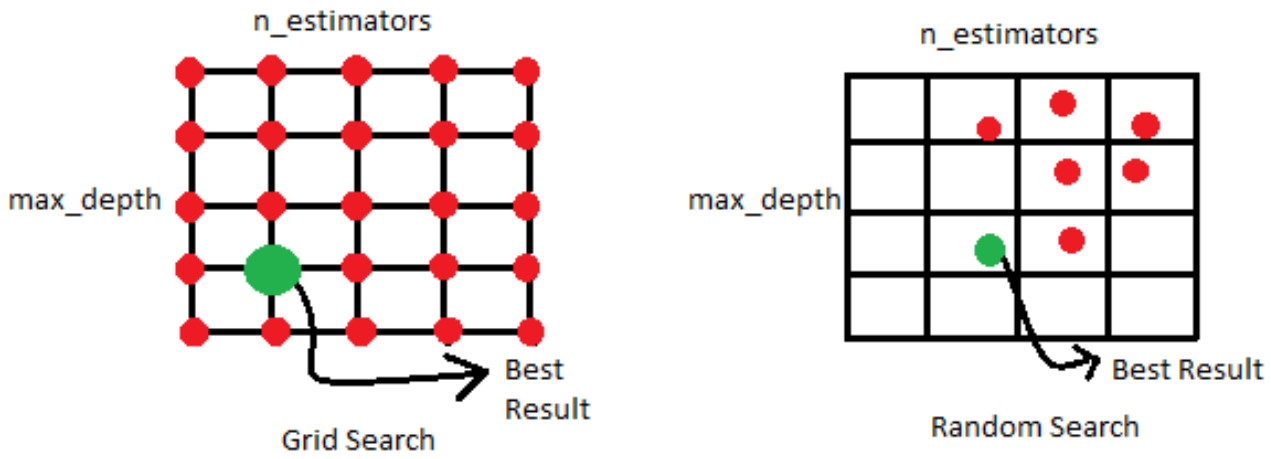
```
# Importar librerías para CV
from sklearn.model_selection import KFold, LeaveOneOut, cross_val_score
# Crear el modelo (Regresión lineal por ejemplo)
model = LinearRegression()
# Crear el método de división
k_folds = KFold(n_splits = 5)
leave_one_out = LeaveOneOut()
# Realizar la validacipon cruzada
scores = cross_val_score(model, X, Y, cv=k_folds)
# Obtener el promedio
print("Average CV Score: ", scores.mean())
```

Búsqueda de cuadrícula

La mayoría de los modelos contienen parámetros que se puede ajustar para optimizar su rendimiento.

La búsqueda por cuadrícula con validación cruzada (GridSearchCV) es una técnica de búsqueda de hiperparámetros óptimos de un modelo evaluando el rendimiento del modelo en todas las combinaciones de hiperparámetros.

Por otro lado, la búsqueda aleatoria con validación cruzada (RandomizedSearchCV) intenta combinaciones aleatorias de un rango de valores, teniendo que ajustar el número de iteraciones.



Búsqueda de cuadrícula

Podemos encontrar las dos técnicas de búsqueda en la librería de *sklearn.model_selection* y tenemos que pasarle un mínimo de dos argumentos:

- **estimator:** el modelo a usar
- **param_grid:** un diccionario con los hiperparámetros
- **cv** (opcional): estrategia de validación cruzada

Nota: GridSearchCV devolverá el resultado con la mayor puntuación, por ello no es conveniente utilizar la métrica RMSE, en su defecto se suele usar el negativo de dicha métrica, definido como: *'neg_mean_squared_error'*

```
#Importar librerías
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
# Crear el estimador (PLSR por ejemplo)
plsr = PLSRegression()
# Definir el espacio de búsqueda de hiperparámetros
param_grid = {'n_components': [1, 2, 3, 4, 5]}
# Crear el método de división
k_folds = KFold(n_splits = 5)
# Crear el método de búsqueda
grid_search = GridSearchCV(plsr, param_grid, cv=k_folds, scoring='neg_mean_squared_error')
# Realizar la búsqueda
grid_search.fit(X, y)
# Obtener los mejores hiperparámetros
best_params = grid_search.best_params_
```

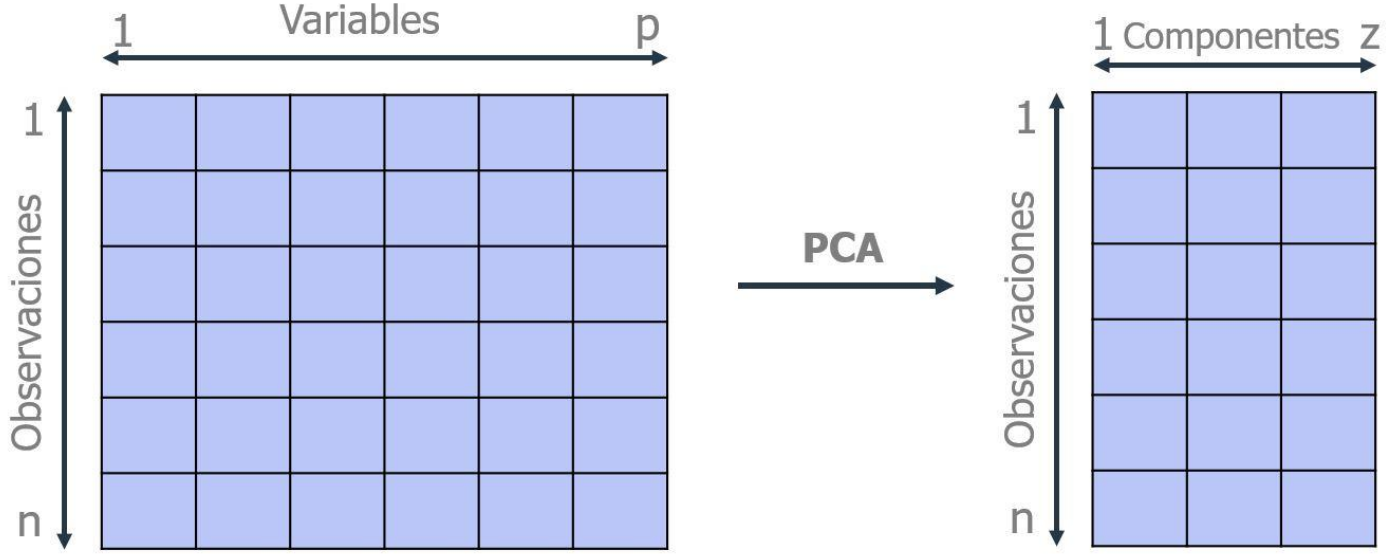
Análisis de componente principales

PCA es un método estadístico de reducción de dimensionalidad que permite simplificar la complejidad de espacios con múltiples dimensiones a la vez que conserva su información.

Estos componentes principales son algunas combinaciones lineales de las variables originales que explican al máximo la varianza de todas las variables.

Términos usados:

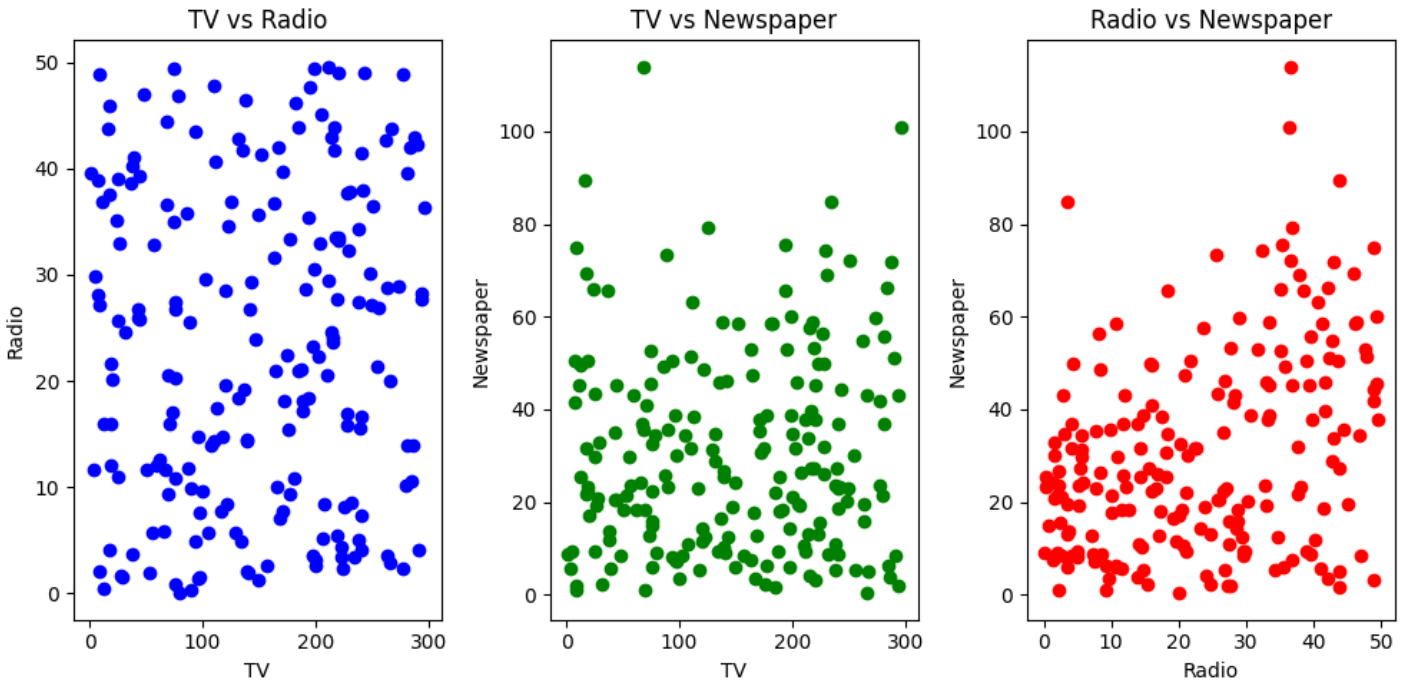
- **Componente Principal** (CP): Combinación lineal de las variables predictoras.
- **Cargas**: los pesos que transforman los predictores en componentes.
- **Screeplot**: gráfico de las varianzas de los componentes, mostrando su importancia relativa, ya sea como varianza explicada o como porción de esta.



Análisis de componente principales

PCA es una técnica de aprendizaje no supervisado que puede aplicarse como parte del análisis exploratorio de datos. En este caso solo usaremos un número de variables de las cuales nos interesa extraer información, por ejemplo, sobre la existencia de subgrupos entre variables u observaciones.

Suponga que quisiera representar *n* observaciones con medidas sobre *p* variables como parte de un análisis exploratorio. Se podría examinar en **representaciones bidimensionales**, por ejemplo, en el caso de los gastos en publicidad en tres diferentes medios:



Sin embargo, existen $\binom{p}{2} = p(p-1)/2$ posibles representaciones entre pares de variables y si *p* es muy grande, esto se hace inviable.

PCA permite representar estas combinaciones en un nuevo plano, de manera que sus ejes coincidan con la direccione de **máxima varianza** de los datos.

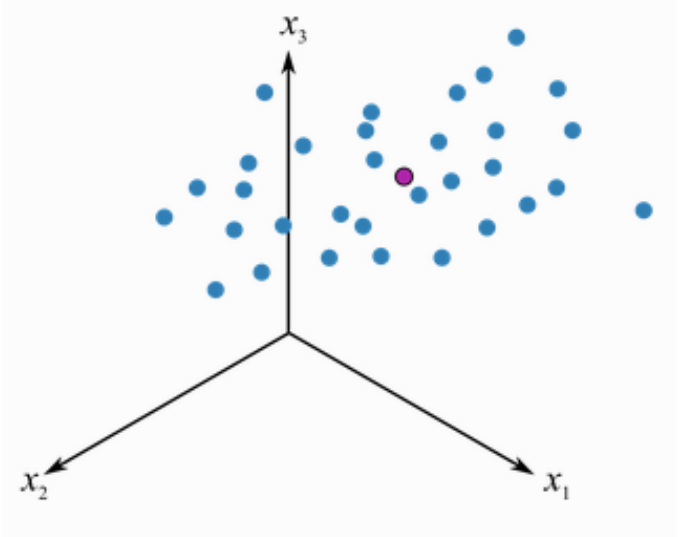
Recordemos la estandarización

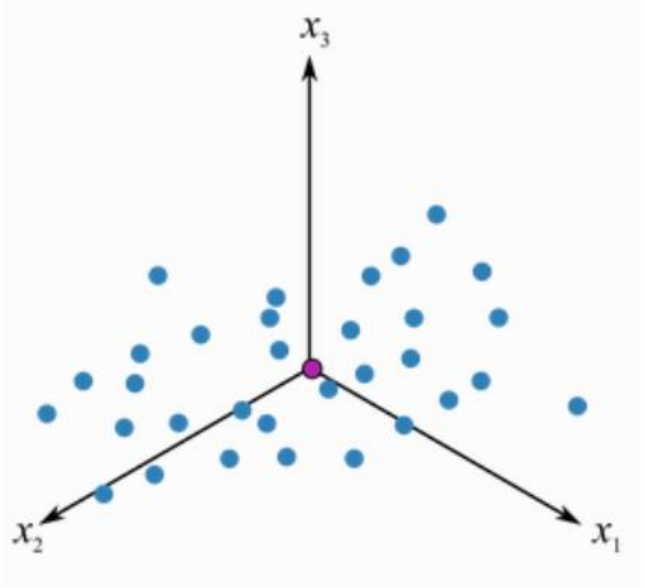
Normalmente la distribución de los datos puede verse como en la primera figura, donde el punto diferente representa la media.

Antes de aplicar PCA los datos deben moverse al origen para que tengan media 0 y escalarlos a una varianza unitaria para eliminar el efecto de las distintas unidades en la que puedan estar los datos, esto es, **estandarizar los datos**.

$$z_i = \frac{x_i - media(x)}{sd(x)}$$

Así los datos pasan a verse como en la segunda figura.



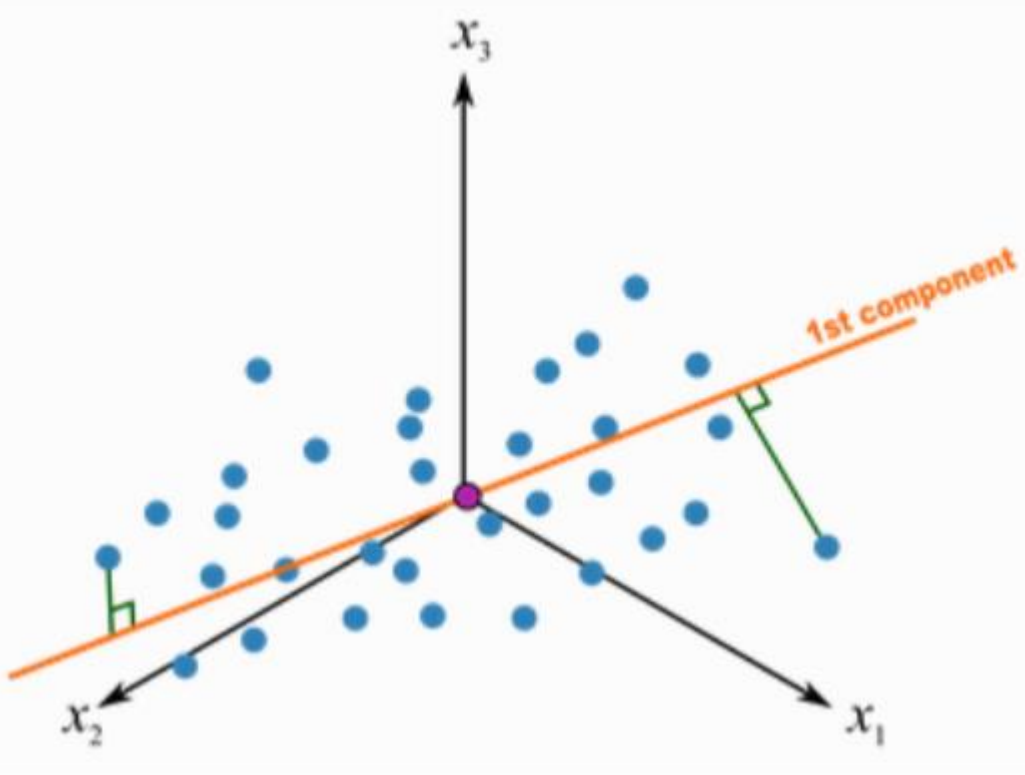


Componentes

El primer componente principal será una línea que se ajuste de la mejor manera a los datos estandarizados y explicará mejor dichos datos cuanta más correlación exista entre ellos:

El **vector** de cargas representará esta **componente**.

La distancia desde el origen a cada proyección (en color verde) representa el **score** o puntuación de cada observación, por lo que cada observación tendrá su propio score para cada componente.

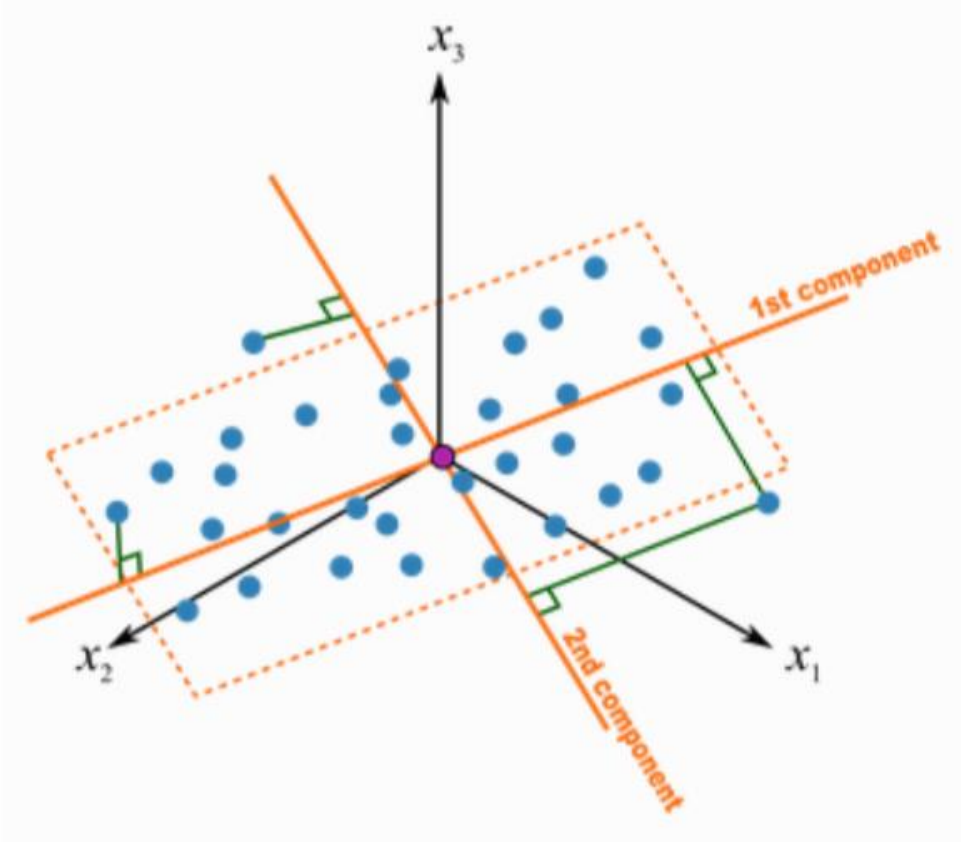


Componentes

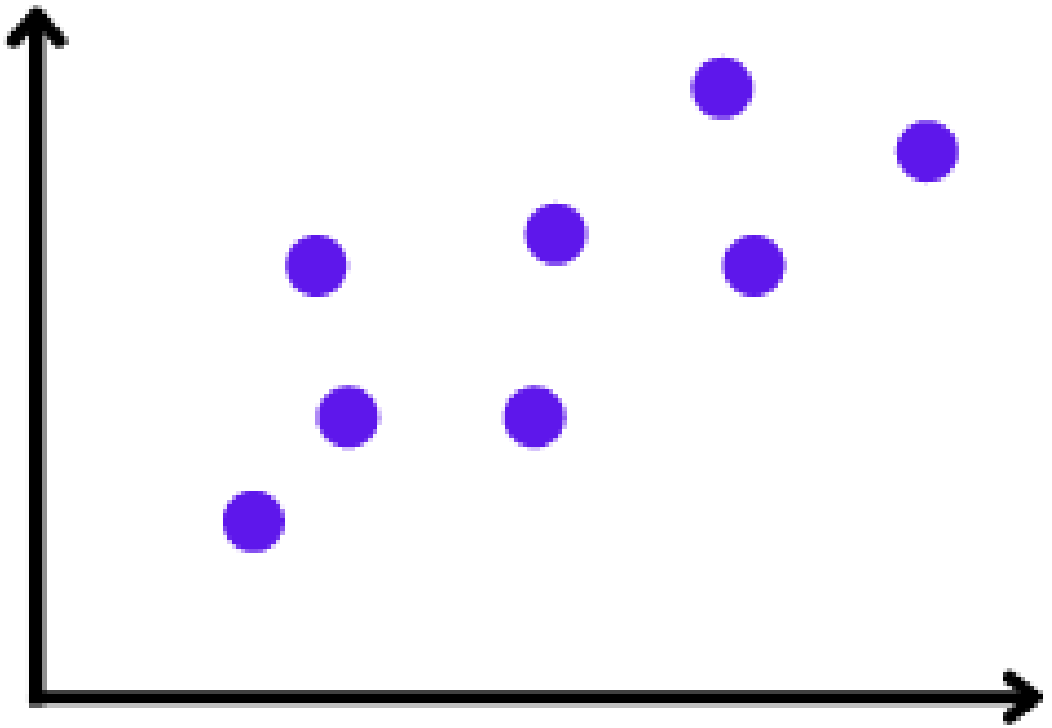
La segunda componente se calcula de forma perpendicular a la primera y pasará por el origen, maximizando también la varianza de los scores sobre la nueva dirección.

Las dos primeras componentes principales ya han representado los datos en un plano de menor dimensión.

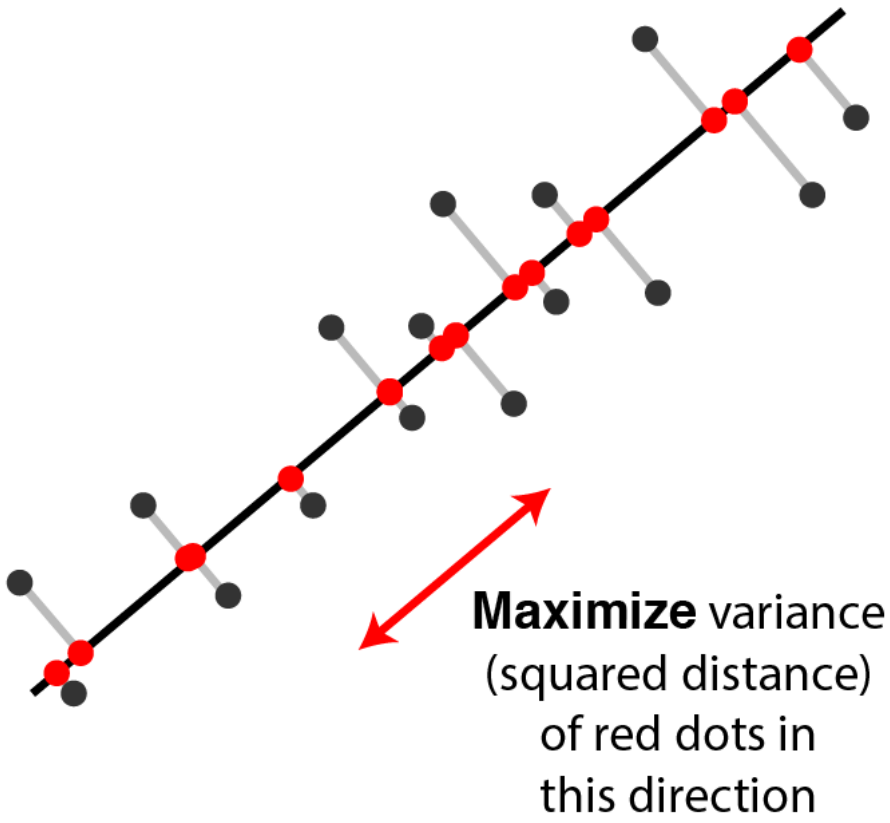
Si hubiera componentes adicionales, cada uno adicional sería ortogonal a los demás.



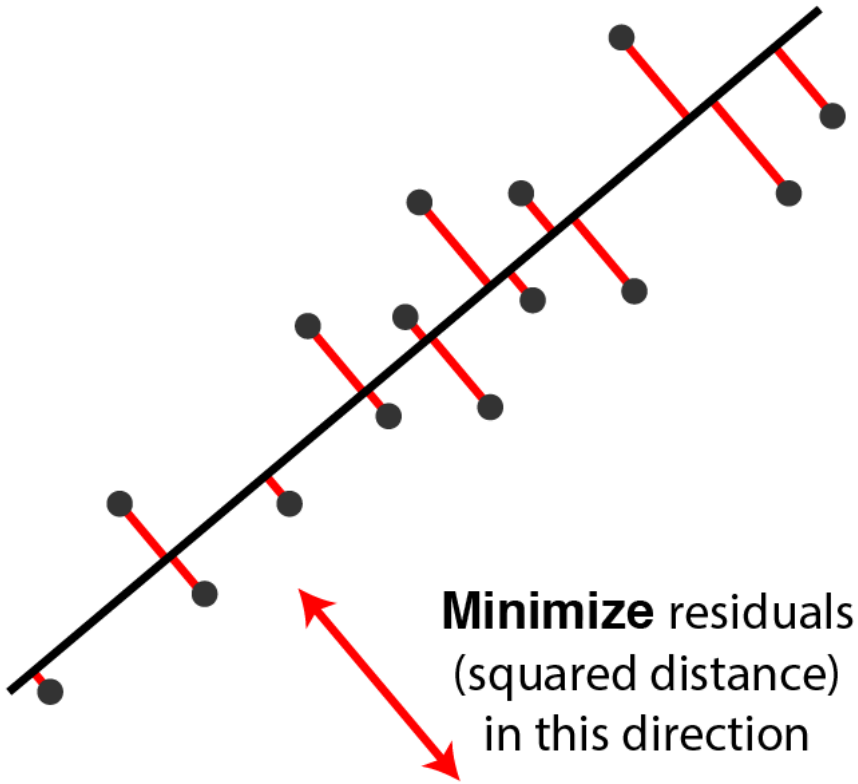
Componentes



Componentes

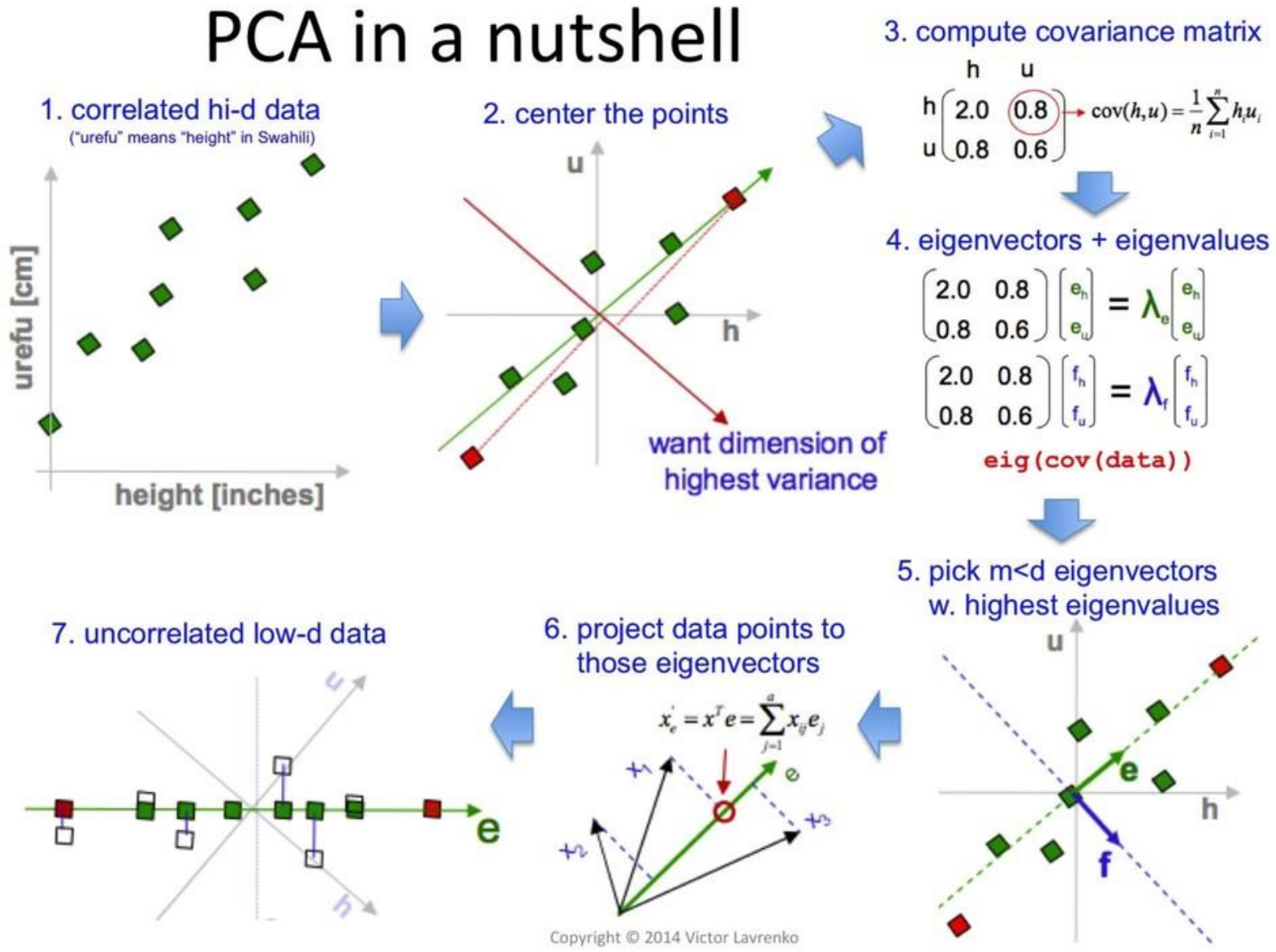


Maximize variance (squared distance) of red dots in this direction



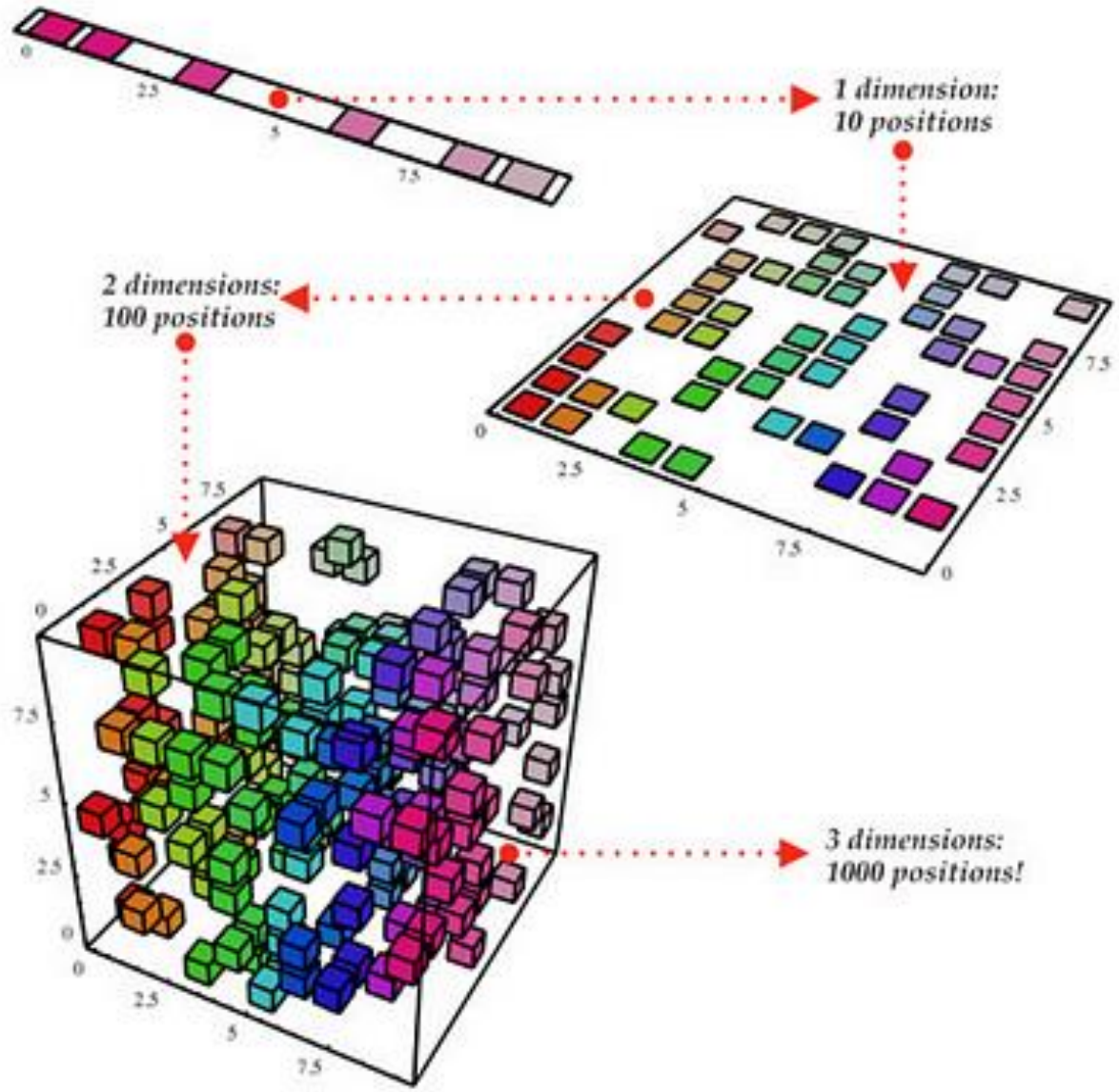
Minimize residuals (squared distance) in this direction

Componentes



Reducción de la dimensionalidad

En datos de alta dimensión muchos son redundantes y pueden reducirse eficientemente a un número menor de variables sin pérdida significativa de información.

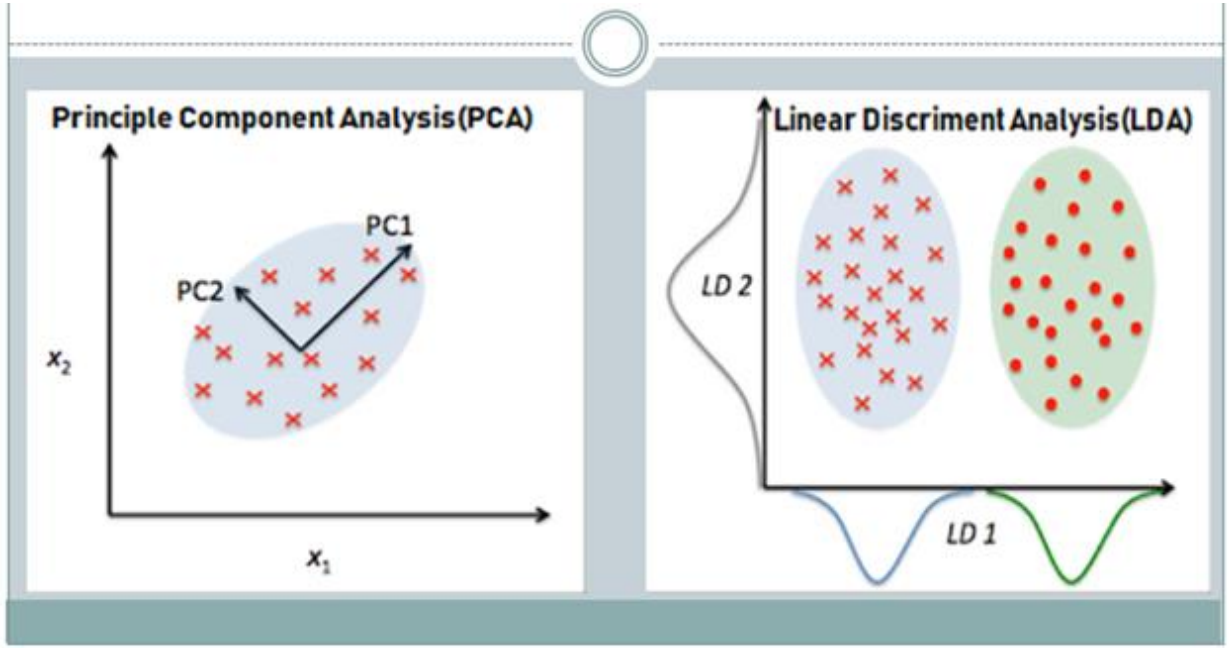


Reducción de la dimensionalidad

Las técnicas más populares de reducción de dimensionalidad son el análisis de componentes principales (PCA) y el análisis discriminante lineal (LDA).

A diferencia de PCA, LDA además de maximizar la **varianza** de datos también maximiza la **separación de múltiples clases.**

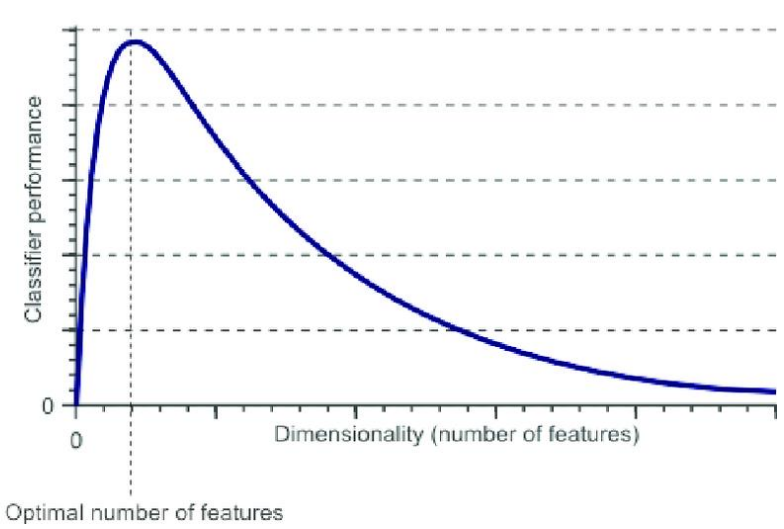
Ambas son técnicas de transformación lineal, sin embargo, mientras que PCA es un algoritmo no supervisado, LDA es supervisado.

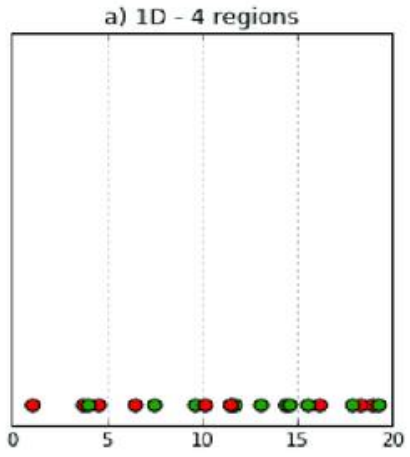


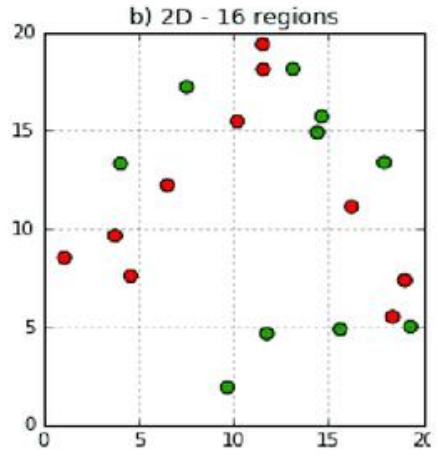
La maldición de la dimensionalidad

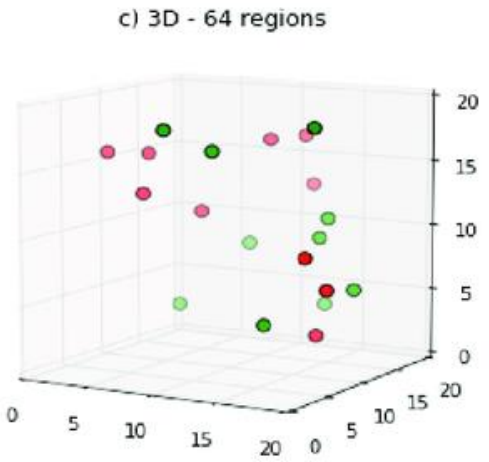
En el aprendizaje automático surge cuando se trabaja con datos de alta dimensión, lo que puede llevar a que:

- La eficiencia y eficacia de los algoritmos se deteriora a medida que la dimensionalidad de los datos aumenta exponencialmente.
- Los puntos de datos se vuelven dispersos, lo que dificulta discernir patrones o relaciones significativas.
- Genera una mayor complejidad computacional, tiempos de entrenamiento más prolongados, riesgo de sobreajuste y correlaciones falsas.









Componentes

En Python podemos calcular los PCs con la implementación de `scikit-learn` `sklearn.decomposition.PCA.`

```
from sklearn.decomposition import PCA
pcs = PCA(n_components=3)
pcs.fit(X)
loadings = pd.DataFrame(pcs.components_,
columns=name_columns)
loadings
```

La naturaleza de los componentes a menudo revela información sobre la estructura de los datos. Un screeplot ayuda a visualizar la importancia relativa de los CPs. La información para crear el gráfico de cargas con los resultados de PCA está disponible en `explained_variance_ratio.`

```
explained_variance = pd.DataFrame(pcs.explained_variance_ratio_)
ax = explained_variance.head(10).plot.bar(legend=False, figsize=(4, 4))
ax.set_xlabel('Component')
```

Practica

Predicción de materia seca en mango con datos espectrales

La espectroscopía NIR es una técnica que permite relacionar información espectral con diversos atributos en frutas, permitiendo una evaluación no invasiva. El contenido de materia es un indicador clave para el momento de cosecha.

Importe el conjunto de datos de mango con 11,692 mediciones de absorbancia en 281 longitudes onda (variables predictoras) y contenido de materia seca (objetivo) para **seleccionar** y **evaluar** un modelo de regresión, utilizando los métodos de MLR, PCR y PLSR. https://data.mendeley.com/datasets/46htwnp833/2

El conjunto de datos es muy variado e incluye muestras de diferentes cultivares, grados de madurez, temporadas de cosecha y regiones del cultivo, por lo que un modelo de regresión lineal podría no ajustarse correctamente a toda la variabilidad. Por ende, sería conveniente ajustar un **modelo específico a un cultivar**.

Procedimiento:

1. Seleccione uno de los 10 cultivares de mango.
2. Separe las muestras definidas como **“Val Ext” en un conjunto de prueba**, el cual usará para comparar los modelos seleccionados.
3. Agrupe los datos en las **variables X e Y**. Considere que X corresponde a las columnas con identificador del 309 al 1149.
4. Prepare un Pipeline con los regresores: **MLR, PCR y PLSR**
5. Defina **dos espacios de búsqueda** de elementos para PCR y PLSR, uno con pocos componentes y otro con un número mayor. Ejemplo:

   Espacio1=[1,2,3,...,30] Espacio2=[32,64,100,150,200,250,280]

6. Con cada espacio, realice una **búsqueda por cuadrícula con validación cruzada** del número óptimo de componentes principales y variables latentes a incluir en el modelo correspondiente. Utilice la estrategia k-foldCV con k=10 divisiones.
7. **Ajuste el modelo de MLR** con todas las variables predictoras. Con el número óptimo de CPs y LV, **ajuste los modelos de PCR y PLSR**
8. Utilice el conjunto de prueba para **evaluar los modelos seleccionados** indicando qué modelo captura la mayor variación del contenido de materia seca y tiene el menor error.
9. Indique si la reducción de la dimensionalidad tuvo algún efecto en la precisión de los modelos.

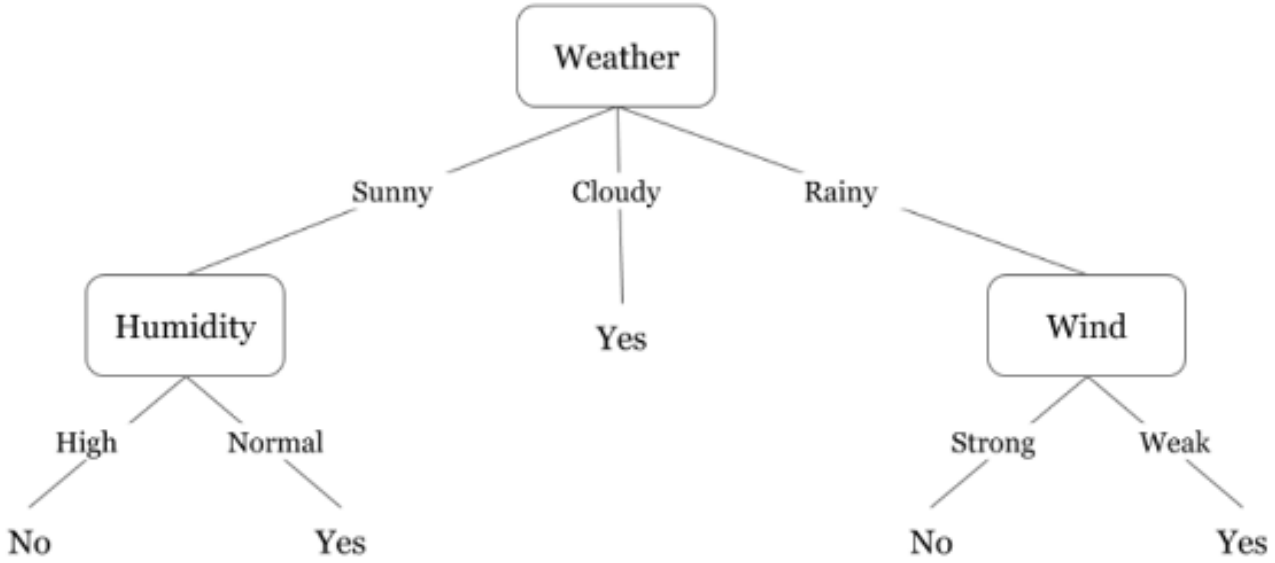
Árboles de decisión (DT)

Modelos de aprendizaje supervisado no paramétrico para tareas de clasificación y regresión (conocidos también como algoritmos CART).

Los modelos DT pueden predecir la variable objetivo aprendiendo reglas de decisión simples inferidas de las características de los datos.

Los DT se pueden ver como varias declaraciones if-else. Comprueba si es verdadera y, si lo es, pasa al siguiente nodo adjunto a esa decisión. Por ejemplo, para determinar si una persona puede ir a jugar o no:

Day	Weather	Temperature	Humidity	Wind	Play?
1	Sunny	Hot	High	Weak	No
2	Cloudy	Hot	High	Weak	Yes
3	Sunny	Mild	Normal	Strong	Yes
4	Cloudy	Mild	High	Strong	Yes
5	Rainy	Mild	High	Strong	No
6	Rainy	Cool	Normal	Strong	No
7	Rainy	Mild	High	Weak	Yes
8	Sunny	Hot	High	Strong	No
9	Cloudy	Hot	Normal	Weak	Yes
10	Rainy	Mild	High	Strong	No

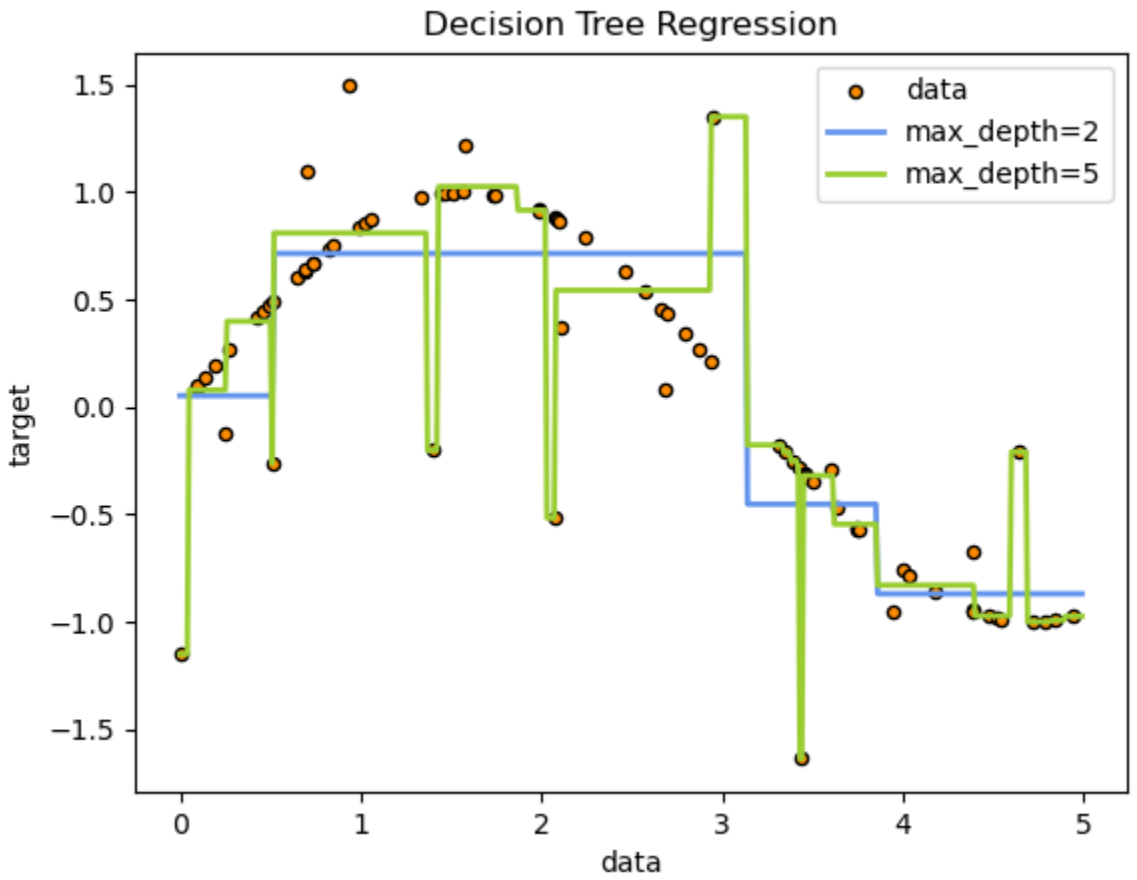


Árbol de decisión para regresión

En un problema de regresión un DT puede verse como una aproximación constante por partes.

Por ejemplo, en la figura se observa como los DT aprenden aproximarse a una curva sinosoidal con un conjunto de reglas de decisión if-else.

Cuanto más profundo sea el árbol, más complejas serán las reglas de decisión y más ajustado será el modelo.



Criterios de clasificación

Ganancia de información

La ganancia de información se utiliza para decidir en qué característica dividirse en cada paso de la construcción del árbol. A medida que avanza el proceso de división deseamos que más muestras dentro de cada nodo pertenezcan a una sola clase, es decir, aumentar la **pureza de cada nodo.**

La división con la mayor ganancia de información se tomará como la primera división y el proceso continuará hasta que todos los nodos de decisión sean puros o hasta que la ganancia de información sea 0.

Una de las medidas de impureza más usadas es la entropía (Ent), definida como:

$$\mathrm{Ent}(D) = -\sum_{k=1}^{|y|} p_k \log_2 p_k .$$

Donde:
D es el conjunto de datos
$|y|$ es el número de clases
p_k es la proporción de ejemplos de la clase k en el conjunto de datos

Cuanto **menor sea la entropía** del conjunto de datos **mayor será la pureza**.

Ganancia de información

La ganancia de información al dividir el conjunto de datos **D** con la característica **a** se calcula como:

$$\text{Gain}(D, a) = \text{Ent}(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} \text{Ent}(D^v).$$

Donde:

v es el conjunto de posibles valores del atributo **a**

D^v es el subconjunto de D para el cual el atributo **a** tiene el valor v

$|D|$ es el tamaño del conjunto de datos original

$|D^v|$ es el tamaño del subconjunto D^v

$Ent(D^v)$ es la entropía del subconjunto D^v

Ejemplo

Considerando las siguientes características para clasificación de melón en maduro e inmaduro.

ID	color	root	sound	texture	umbilicus	surface	ripe
1	green	curly	muffled	clear	hollow	hard	true
2	dark	curly	dull	clear	hollow	hard	true
3	dark	curly	muffled	clear	hollow	hard	true
4	green	curly	dull	clear	hollow	hard	true
5	light	curly	muffled	clear	hollow	hard	true
6	green	slightly curly	muffled	clear	slightly hollow	soft	true
7	dark	slightly curly	muffled	slightly blurry	slightly hollow	soft	true
8	dark	slightly curly	muffled	clear	slightly hollow	hard	true
9	dark	slightly curly	dull	slightly blurry	slightly hollow	hard	false
10	green	straight	crisp	clear	flat	soft	false
11	light	straight	crisp	blurry	flat	hard	false
12	light	curly	muffled	blurry	flat	soft	false
13	green	slightly curly	muffled	slightly blurry	hollow	hard	false
14	light	slightly curly	dull	slightly blurry	hollow	hard	false
15	dark	slightly curly	muffled	clear	slightly hollow	soft	false
16	light	curly	muffled	blurry	flat	hard	false
17	green	curly	dull	slightly blurry	slightly hollow	hard	false

Se calcula la entropía de cada subconjunto correspondiente a cada característica.
Por ejemplo, si se divide por color se crean 3 subconjuntos y su entropía es:

$$\mathrm{Ent}(D^1) = -\left(\frac{3}{6}\log_2\frac{3}{6} + \frac{3}{6}\log_2\frac{3}{6}\right) = 1.000,$$ → Green

$$\mathrm{Ent}(D^2) = -\left(\frac{4}{6}\log_2\frac{4}{6} + \frac{2}{6}\log_2\frac{2}{6}\right) = 0.918,$$ → Dark

$$\mathrm{Ent}(D^3) = -\left(\frac{1}{5}\log_2\frac{1}{5} + \frac{4}{5}\log_2\frac{4}{5}\right) = 0.722.$$ →Light

Ejemplo (continuación)

La ganancia de información al dividir los datos por color es:

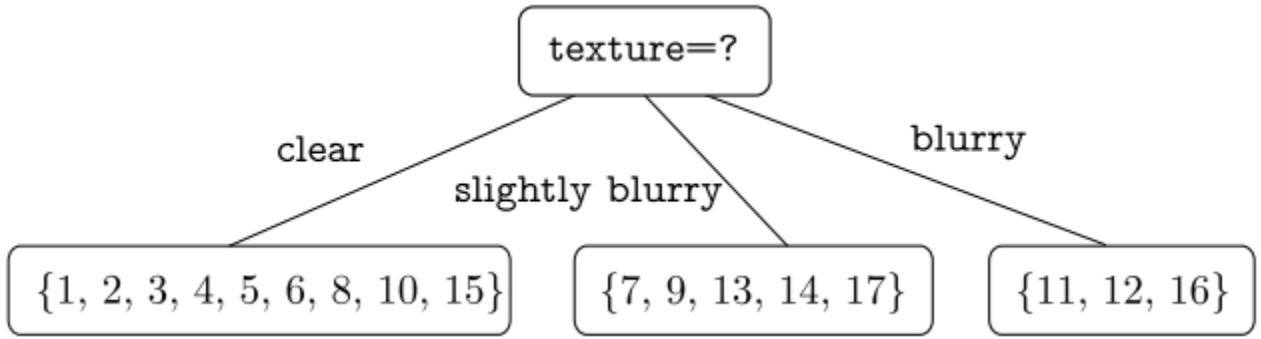
$$\begin{aligned}\text{Gain}(D, \text{color}) &= \text{Ent}(D) - \sum_{v=1}^{3} \frac{|D^v|}{|D|} \text{Ent}(D^v) \\ &= 0.998 - \left(\frac{6}{17} \times 1.000 + \frac{6}{17} \times 0.918 + \frac{5}{17} \times 0.722 \right) \\ &= 0.109.\end{aligned}$$

De la misma manera podemos calcular la ganancia al dividir por las otras características:

$$\begin{aligned}&\text{Gain}(D, \text{root}) = 0.143; \quad \text{Gain}(D, \text{sound}) = 0.141; \\ &\text{Gain}(D, \text{texture}) = 0.381; \quad \text{Gain}(D, \text{umbilicus}) = 0.289; \\ &\text{Gain}(D, \text{surface}) = 0.006.\end{aligned}$$

La textura se elige como característica de división para el nodo de raíz.

Hasta este punto el árbol queda como:

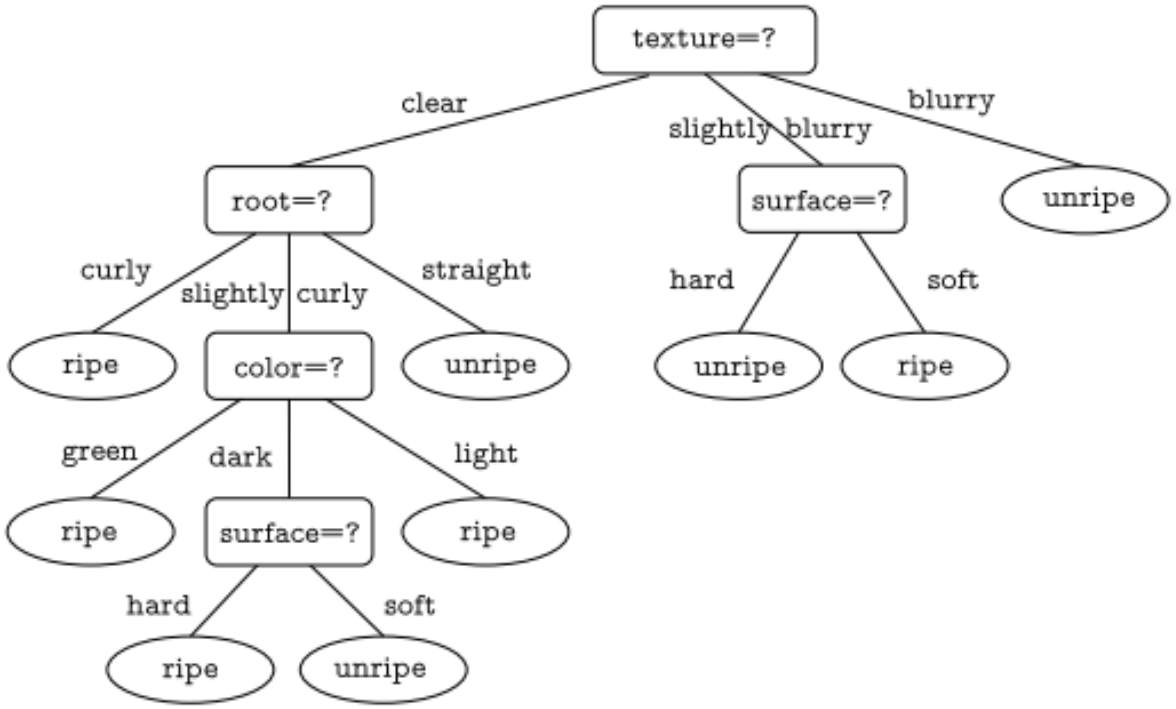


Ejemplo (finalización)

Cada nodo secundario se vuelve a dividir en el conjunto de características disponible y se obtienen las ganancias de dichas características candidatas en D^1. Por ejemplo, con el primer nodo:

$$\text{Gain}(D^1, \text{color}) = 0.043; \quad \text{Gain}(D^1, \text{root}) = 0.458;$$
$$\text{Gain}(D^1, \text{sound}) = 0.331; \quad \text{Gain}(D^1, \text{umbilicus}) = 0.458$$
$$\text{Gain}(D^1, \text{surface}) = 0.458.$$

Se elige cualquier característica con la mayor ganancia de información. Repitiendo el proceso para cada nodo, el árbol de decisión final queda como:



Índice de Gini

También se puede emplear el índice de Gini para seleccionar la característica de división. Gini(D) representa la probabilidad de que dos muestras seleccionadas aleatoriamente del conjunto de datos D pertenezcan a diferentes clases. Su valor se define como:

$$\text{Gini}(D) = 1 - \sum_{k=1}^{|\mathcal{Y}|} p_k^2.$$

Donde:
D es el conjunto de datos
$|y|$ es el número de clases
p_k es la proporción de ejemplos de la clase k en el conjunto de datos

El índice Gini de la característica a se define como:

$$\text{Gini\_index}(D, a) = \sum_{v=1}^{V} \frac{|D^v|}{|D|} \text{Gini}(D^v).$$

Donde:
v es el conjunto de posibles valores del atributo **a**
D^v es el subconjunto de D para el cual el atributo **a** tiene el valor v
$|D|$ es el tamaño del conjunto de datos original
$|D^v|$ es el tamaño del subconjunto D^v

En este caso, seleccionamos la característica con el menor índice de Gini como la característica de división.

Criterios de regresión

Cuando el objetivo es un valor continuo, entonces para el nodo *m*, los criterios más comunes para evaluar la calidad de la división de datos son:

Error cuadrático medio:

$$\text{MSE}_m = \frac{1}{N_m} \sum_{i \in N_m} (y_i - \hat{y}_m)^2$$

Error absoluto medio:

$$\text{MAE}_m = \frac{1}{N_m} \sum_{i \in N_m} |y_i - \hat{y}_m|$$

La mitad de la desviación de Poisson:

$$\text{Half Poisson Deviance}_m = \frac{1}{N_m} \sum_{i \in N_m} \left(y_i \log \left(\frac{y_i}{\hat{y}_m} \right) - (y_i - \hat{y}_m) \right)$$

Donde:
N_m es el conjunto de instancias en el nodo *m*
y_i es el valor observado en la instancia *i*
$\hat{y}_m$ es el valor predicho promedio en el nodo *m*

Sobreajuste en árboles de decisión

Sino se limitan las divisiones, los árboles de decisión pueden ajustarse perfectamente a las observaciones de entrenamiento creando una hoja por cada observación. Existen dos estrategias para prevenir el sobreajuste:

1. Parada temprana: Limitando el tamaño de árbol por medio de hiperparámetros.
   - max_depth: sino se especifica, los nodos se expanden hasta que todas las hojas sean puras o hasta que todas las hojas contengan menos de min_samples_split.
   - min_samples_split: número mínimo de muestras necesarias para dividir un nodo interno. Default = 2.
   - min_samples_leaf: número mínimo de muestras necesarias para estar en un nodo de hoja. Default = 1.
2. Poda. La poda de complejidad de costos proporciona otra opción para controlar el tamaño de un árbol. Está parametrizada por ccp_alpha, entre mayor es, más número de nodos son podados.

Para conocer el valor óptimo de cada hiperparámetro se puede recurrir a la validación cruzada.

Implementación de DT en regresión simple

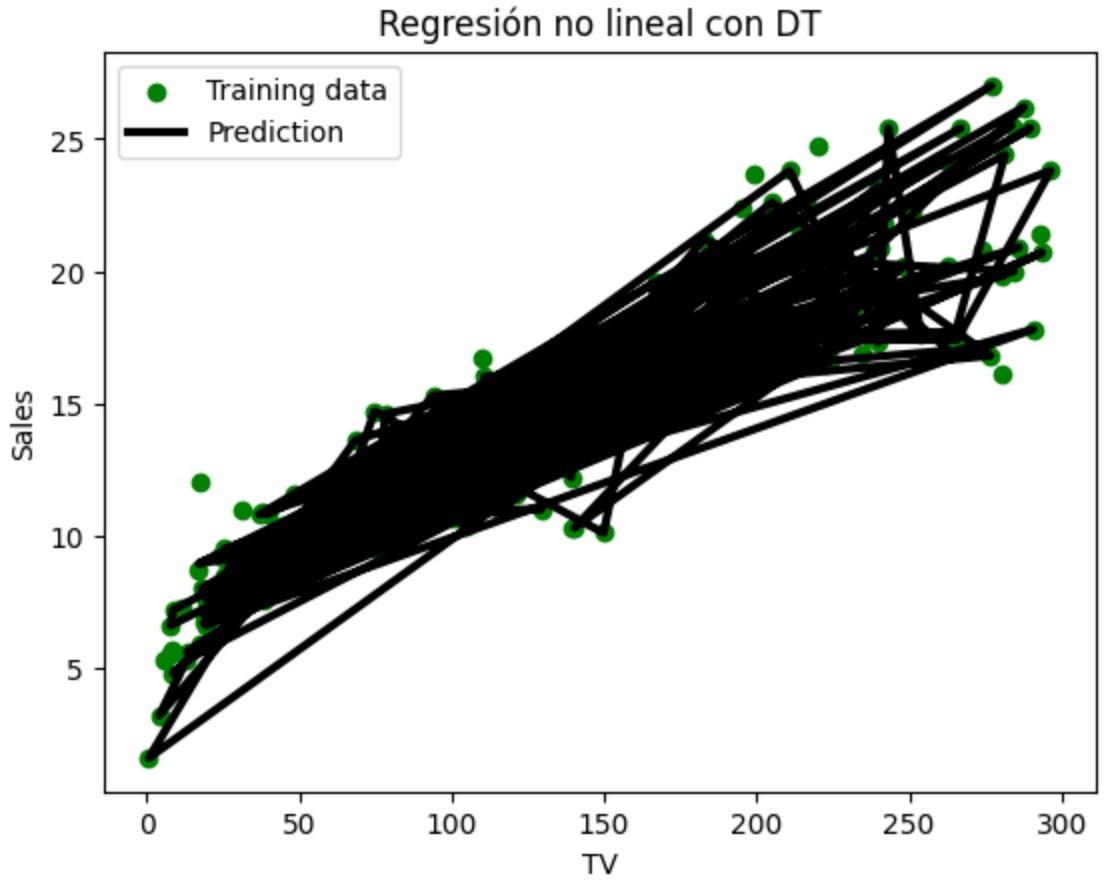
Tomemos los datos sobre gasto de publicidad y número de ventas para ajustar un modelo de DT que relacione el gasto de publicidad en TV con el número de ventas. Dejaremos los parámetros por default de `sklearn.tree`

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor
X =  df[['TV']]
y =  df[['Sales']]
X_train, X_test, y_train, y_test=
train_test_split(X,y, test_size=0.2,
random_state=42)
model_tree = DecisionTreeRegressor()
model_tree.fit(X_train, y_train)
```

Evaluación del modelo:

```
from sklearn.metrics import r2_score
y_pred_test = model_tree.predict(X_test)
y_pred_train = model_tree.predict(X_train)
print('r2_score_train:',r2_score(y_train, y_pred_train))
print('r2_score_test:',r2_score(y_test, y_pred_test))
```

```
r2_score_train: 0.9913617387206112
r2_score_test: 0.8040178990391927
```



Ajuste del modelo en datos de entrenamiento

Optimización con GridSearchCV

Realizando una poda y parada temprana por validación cruzada podemos obtener el mejor valor de los hiperparámetros:

```
# Poda
param_grid = {'ccp_alpha':np.linspace(0, 5, 20)}
# Estrategia de validación cruzada
kfold = KFold(n_splits=5, shuffle=True,
random_state=42)
#Búsqueda por validación cruzada
grid_tree = GridSearchCV(DecisionTreeRegressor(),
param_grid = param_grid, cv=kfold)
grid_tree.fit(X_train, y_train)
print(grid_tree.best_params_)
```

```
{'ccp_alpha': 0.2631578947368421}
```

```
# Parada temprana
param_grid = {'max_depth':[2,3,4,5,6],
              'min_samples_split':[2,3,4,5,6],
              'min_samples_leaf':[1,2,3,4,5]}
# Estrategia de división
kfold = KFold(n_splits=5, shuffle=True,
random_state=42)
#Búsqueda por validación cruzada
grid_tree = GridSearchCV(DecisionTreeRegressor(),
param_grid = param_grid, cv=kfold)
grid_tree.fit(X_train, y_train)
print(grid_tree.best_params_)
```

```
{'max_depth': 3, 'min_samples_leaf': 4, 'min_samples_split': 2}
```

Resultados:

```
r2_score_train: 0.8450033831699615
r2_score_test: 0.8110313358780219
```

```
r2_score_train: 0.8458116248246583
r2_score_test: 0.8037895383263218
```

Fuentes

- James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). *An Introduction to Statistical Learning with Applications in Python* (First Printing).
- Hastie, T., Tibshirani, R., & Friedman, J. (2017). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction* (Second Edition, Corrected 12th printing). Springer.
- Bruce, P., Bruce, A., & Gedeck, P. (2020). *Practical Statistics for Data Scientists: 50+ Essential Concepts Using R and Python* (Second Edition). O'Reilly Media.
- Kok, Z. H., Shariff, A. R. M., Alfatni, M. S. M., & Khairunniza-Bejo, S. (2021). Support vector machine in precision agriculture: A review. *Computers and Electronics in Agriculture, 191*, 106546. https://doi.org/10.1016/j.compag.2021.106546
- Yadav, T. N., Prasad, R. C., Reddy, R. O., & Gopal, P. (2020). Crop yield and fertilizers prediction using decision tree algorithm. *International Journal of Engineering Applied Sciences and Technology*, *5*(1), 187-193. http://www.ijeast.com
- G. T. Reddy *et al.*, Analysis of Dimensionality Reduction Techniques on Big Data, in *IEEE Access*, vol. 8, pp. 54776-54788, 2020, doi: 10.1109/ACCESS.2020.2980942.
- Zhou, Z.-H. (2021). *Machine Learning*. Springer. 459 p.
- Vujović, Ž. Đ. (2021). Classification model evaluation metrics. *International Journal of Advanced Computer Science and Applications, 12*(6). https://doi.org/10.14569/IJACSA.2021.0120670
- Xu, D., & Tian, Y. (2015). *A comprehensive survey of clustering algorithms*. Annals of Data Science, 2(2). DOI 10.1007/s40745-015-0040-1
- Tharwat, A. (2019). *Parameter investigation of support vector machine classifier with kernel functions*. Springer. https://doi.org/10.1007/s10115-019-01335-4
- Brereton, R.G., & Lloyd, G.R. (2009). Support vector machines for classification regression. *Analyst, 135*, 230-267. DOI: 10.1039/b918972f
- Singh, V., & Singh. A. (2019). Analysis of agricultura data using principal components análisis. *International Journal of Multidisciplinary Research and Development, 7*(1), 34-37. www.allsubjectjournal.com
- Emmert-Streib, F., & Dehmer, M. (2019). Evaluation of regression models: model assessment, model selection and generalization error. *Machine learning and knowledge extraction, 1*(1), 521-551. https://doi.org/10.3390/make1010032
- Gil Martínez, C. (Junioo, 2018). Análisis de componentes principales (PCA). Rpubs. https://rpubs.com/Cristina_Gil/PCA
- Amat Rodrigo, J. (Diciembre, 2020). PCA con Pyhton. Cieniadedatos. https://cienciadedatos.net/documentos/py19-pca-python
- https://github.com/Sarvandani/Machine_learning-deep_learning_11_algorithms-of-regression