

## Módulo I

### 1. Conceptos básicos y definición de aprendizaje de máquinas:

#### a. Definiciones:

- i. Conjunto de métodos que detectan automáticamente patrones complejos en los datos, estos patrones pueden ser utilizados para predecir el futuro (Murphyn, K. P. (2012)).
- ii. Un programa de computadora aprende de la experiencia E con respecto a alguna tarea T y alguna medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E (Mitchell, T (1997)).
- iii. Campo de estudios que entregan a los computadores la habilidad de aprender sin ser explícitamente programados (Samuel, A (1959)).

#### b. Aprendizaje de máquinas:



- i. La experiencia se asocia con los datos.
- ii. La tarea es el objetivo.
- iii. El rendimiento mide la claridad del modelo para la tarea dada.
- iv. Dos ejemplos:

Experiencia (Datos)	Tarea	Rendimiento
<ul style="list-style-type: none"> <li>Compras de clientes.</li> <li>Imágenes médicas con diagnósticos.</li> </ul>	<ul style="list-style-type: none"> <li>Agrupar clientes de acuerdo a preferencias.</li> <li>Predecir diagnóstico.</li> </ul>	<ul style="list-style-type: none"> <li>Similitud dentro los grupos.</li> <li>Error de predicción.</li> </ul>

### c. ¿Por qué estudiar/usar aprendizaje de máquinas?

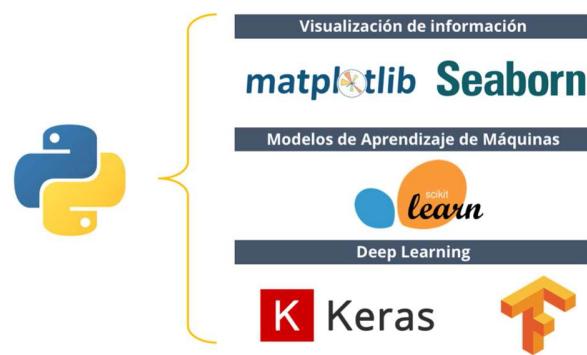
#### i. Situación actual:



#### 1. Disponibilidad de datos:

Fuente	Información generada
Twitter	500 millones de tweets son enviados cada 24 horas
Facebook	4 petabytes de datos son creados diariamente
Whatsapp	65 billones de mensajes son enviados cada día
Youtube	4.5 millones son vistos en 1 minuto
Buscadores	5 billones de búsquedas son realizadas a diario

#### 2. Oferta de modelos y librerías:



### 3. Tareas con buen rendimiento:

Detección de objetos en imágenes

Traducción de textos

Reconocimiento de voz

### d. Aplicaciones destacadas:



<https://www.biofournis.com/>



<https://cinnamon.is/en/>



<https://hireiqinc.com/>



## 2. Tipos de problemas en aprendizaje de máquinas:

### a. Aprendizaje de máquina:



- i. Los problemas vinculados a la **Clasificación** son aquellos donde tengo una categoría que me interesa aprender desde los datos.
- ii. Las **Regresiones**, interesa tratar de predecir a través de un conjunto de variables numéricas una variable dependiente. Ej., el puntaje de logro en una prueba estandarizada de un alumno.
- iii. **Clustering**, técnicas que tratan de detectar segmentos dentro de una base de datos. Ej., segmentos de clientes y generar una estrategia para ellos.

- iv. **Reducción de dimensiones:** muchas veces las bases de datos tienen una gran cantidad de variables y atributos y eso genera grandes complicaciones en nuestros métodos, una de las razones para la reducción es acelerar la ejecución de los algoritmos.
- v. **Navegación** de robot, por ejemplo, o decisiones en tiempo real.

### b. Aprendizaje supervisado:

- i. Aquellos algoritmos que aprenden desde una base de datos con pares del tipo **input/output**. La base de datos se compone de N pares **input/output**.

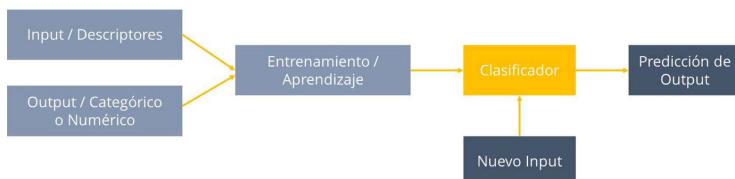
$$\mathcal{D} = \{(x_1, y_1), \dots, (x_N, y_N)\}$$

Input      Output

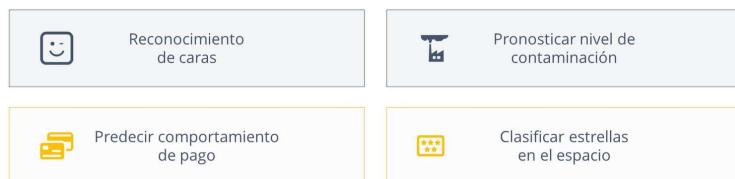
**Input:** edad, profesión, género, nivel de deuda, etc.

**Output:** comprará o no un nuevo producto

- ii. **Input** describe el objeto de interés.
- iii. Su **Objetivo** es aprender una función (clasificador) que permita obtener los valores del **output** por medio del **input**.
- iv. Para obtener el objetivo:

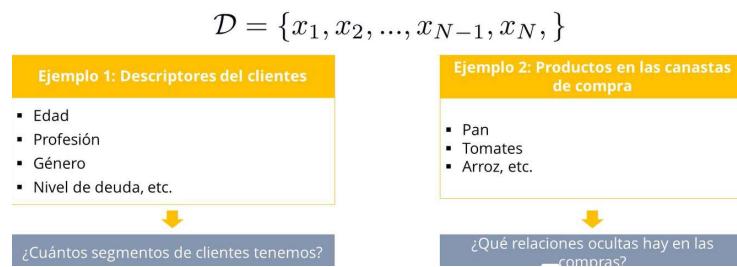


### v. Ejemplos:



### c. Aprendizaje no supervisado:

- i. No tienen **output** definido.
- ii. No hay variable de interés.
- iii. Solo contamos con variables descriptoras.
- iv. Permite entender patrones ocultos en ellos.



### v. Ejemplos,



### d. Aprendizaje reforzado:

- i. Se define un agente que interactúa dentro de un ambiente.
- ii. Dicho agente recibe evaluaciones respecto a sus acciones (refuerzo).
- iii. El agente no es informado respecto a la acción óptima para alcanzar su objetivo, solo recibe estímulos parciales cada vez que toma una decisión.

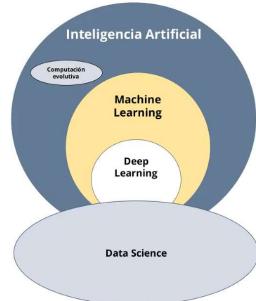


#### iv. Ejemplos,



### 3. Clase Inicial:

#### Tópicos relacionados a ML



**Inteligencia Artificial:** Concepto amplio donde las máquinas son capaces de ejecutar tareas inteligentes tales como «percibir», «razonar», «aprender» y «resolver problemas».

**Computación evolutiva:** Usa la teoría de la evolución natural y la genética en la adaptación evolutiva, para proporcionar una alternativa en la resolución de problemas complejos.

**Machine Learning:** Uso de herramientas estadísticas y de optimización para ayudar a las máquinas a aprender.

**Deep Learning:** Estructuras que modelan un sistema nervioso utilizando varias capas de aprendizaje. Dichas capas se especializan en aprender patrones particulares, el aprendizaje se caracteriza por ser jerárquico y compositivo.

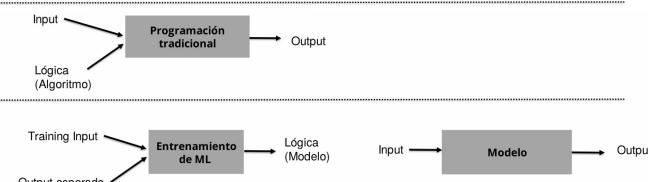
**Data Science:** Uso de modelos y algoritmos para obtener información desde los datos. Algunos plantean que se caracteriza por seguir un procedimiento de análisis científico (hipótesis, recolección, análisis, etc.) de los datos para obtener respuestas.

#### Machine Learning: Definiciones

Hablando:

Campo de estudios que entrega a los computadores la habilidad de aprender sin ser explícitamente programados.

Samuel, A. (1959).



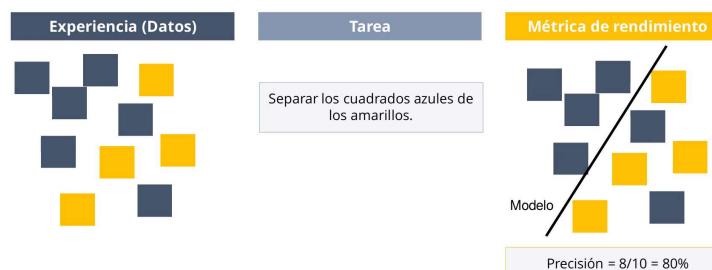
Conjunto de métodos que detectan automáticamente patrones complejos en los datos, estos patrones pueden ser utilizados para predecir el futuro.

Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT press.

Un programa de computadora aprende de la experiencia E con respecto a alguna tarea T y alguna medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E.

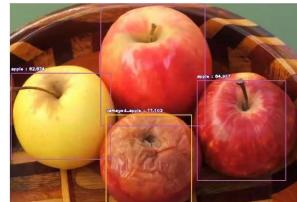
Mitchell, T. (1997). Machine Learning.

#### Machine Learning: componentes principales



## Machine Learning: ejemplos

Experiencia (Datos)	Tarea	Métrica de rendimiento
<ul style="list-style-type: none"><li>Imágenes de manzanas en una línea de proceso con etiqueta de calidad (defectuosa o no defectuosa).</li></ul>	<ul style="list-style-type: none"><li>Identificar manzanas con desperfecto (ej: golpe de sol o polilla).</li></ul>	<ul style="list-style-type: none"><li>Error de selección.</li></ul>



## Machine Learning:

Experiencia (Datos)	Tarea	Métrica de rendimiento
<ul style="list-style-type: none"><li>Mensajes en redes sociales.</li></ul>	<ul style="list-style-type: none"><li>Agrupar mensajes con contenidos similares.</li></ul>	<ul style="list-style-type: none"><li>Cercanía de los mensajes asignados al mismo grupo.</li></ul>



## Machine Learning: ejemplos

Experiencia (Datos)	Tarea	Métrica de rendimiento
<ul style="list-style-type: none"><li>Imagenes de aceitunas con mediciones del contenido de aceite de cada una.</li></ul>	<ul style="list-style-type: none"><li>Predecir el contenido de aceite dada una nueva imagen.</li></ul>	<ul style="list-style-type: none"><li>Error de predicción.</li></ul>



## Machine Learning: ejemplos

Experiencia (Datos)	Tarea	Métrica de rendimiento
<ul style="list-style-type: none"><li>▪ Imágenes con el rostro de personas.</li></ul>	<ul style="list-style-type: none"><li>▪ Crear rostros de personas artificialmente.</li></ul>	<ul style="list-style-type: none"><li>▪ Proporción de engaños a personas o algoritmos entrenados para detectar caras.</li></ul>



## Machine Learning: ejemplos

Experiencia (Datos)	Tarea	Métrica de rendimiento
<ul style="list-style-type: none"><li>▪ Registro de rendimiento académico, variable sociodemográfica y variable asociada a deserción académica.</li></ul>	<ul style="list-style-type: none"><li>▪ Identificar tempranamente alumnos con alta probabilidad de desertar.</li></ul>	<ul style="list-style-type: none"><li>▪ Porcentaje de alumnos desertores no identificados.</li></ul>



## Aprendizaje de máquinas: ejemplos

Experiencia (Datos)	Tarea	Métrica de rendimiento
<ul style="list-style-type: none"><li>▪ Histórico de transacciones y pagos de clientes.</li><li>▪ Variables climáticas y número de esporas asociadas a un hongo.</li><li>▪ Registro de la intensidad de luz en el tiempo de estrellas y su clase.</li></ul>	<ul style="list-style-type: none"><li>▪ Identificar clientes con alta probabilidad de fuga.</li><li>▪ Alertar respecto a la presencia del hongo.</li><li>▪ Asignar clase a nuevas estrellas dado su patrón de intensidad de luz en el tiempo.</li></ul>	<ul style="list-style-type: none"><li>▪ Proporción de clientes fugados no identificados por el modelo.</li><li>▪ Error de predicción.</li><li>▪ Error de clasificación.</li></ul>

## Acceso a modelos y librerías

¿Por qué Python?



Simple y consistente

Oferta de librerías

Multiplataforma y multiparadigma

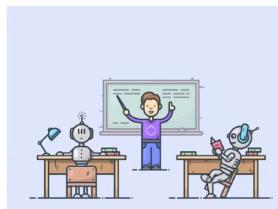
Gran comunidad

Flexibilidad para múltiples tipos de proyectos

## Acceso a modelos y librerías



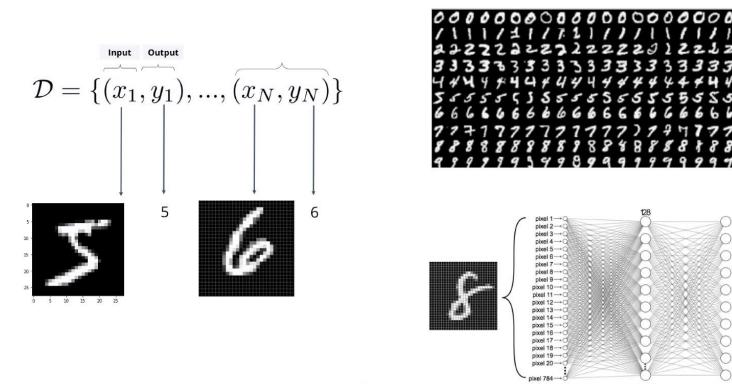
## ¿Cómo aprendemos los humanos ?



## Aprendizaje supervisado



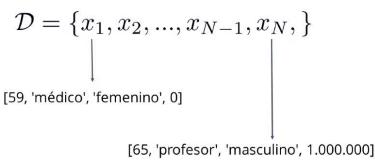
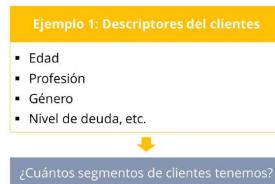
## Aprendizaje supervisado



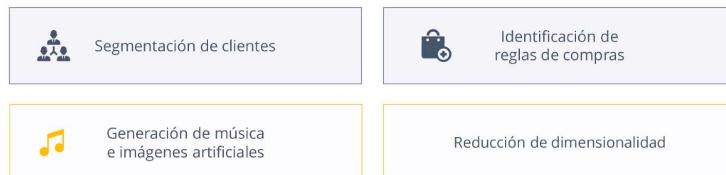
## Ejemplos de aprendizaje supervisado



## Aprendizaje no supervisado



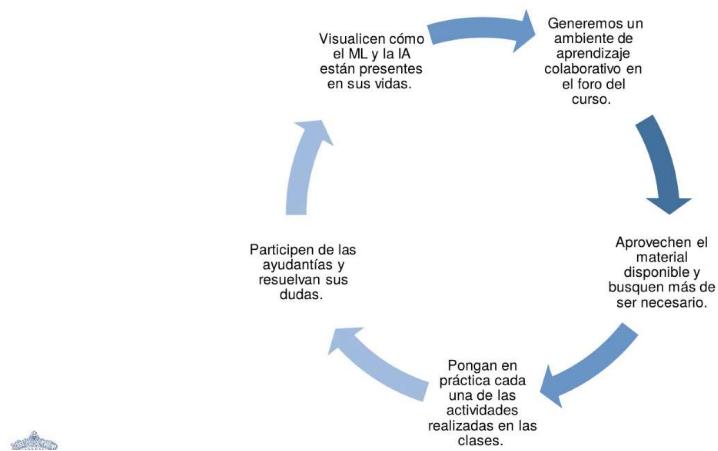
## Ejemplos de aprendizaje no supervisado



## Ejemplos de aprendizaje no supervisado



## Recomendaciones



#### **4. Introducción a las librerías del ecosistema de Data Science:**

##### **a. Ecosistema Python:**

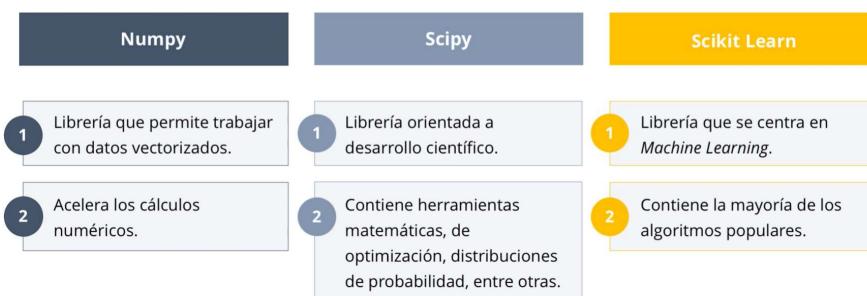
- i. Lenguaje de propósito general.
- ii. Sencillo y rápido de programar.
- iii. Multiplataforma.
- iv. Orientado a objetos.
- v. Muchos más...

##### **b. Librerías para manipulación y análisis de datos:**

###### **i. Ingesta y procesamiento de datos:**



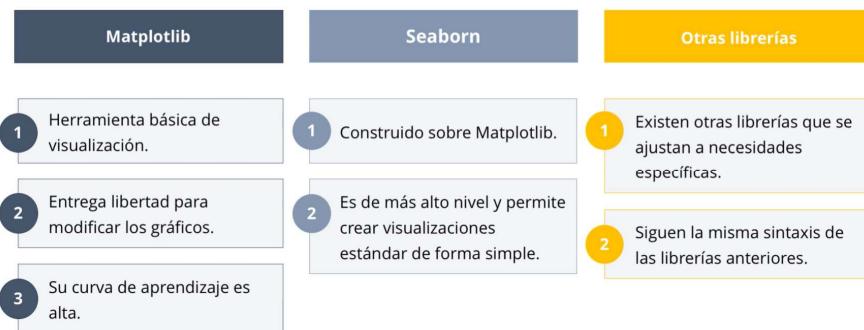
###### **ii. Análisis básicos:**



### iii. Deep Learning:



### c. Librerías para visualización:



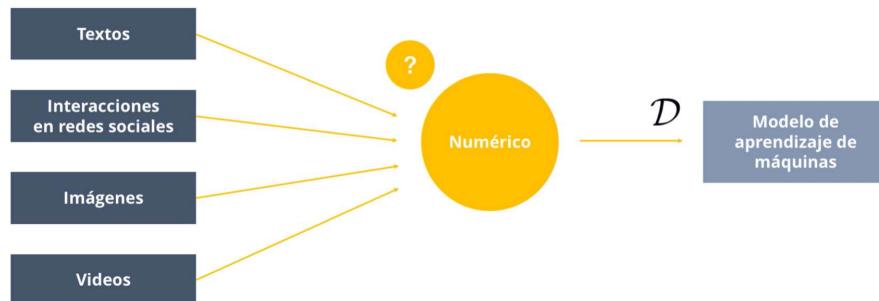
## 5. Tipo de variables

### a. Tipos de datos en aprendizaje de máquinas:

#### Datos en distintas formas



## Distintos tipos de datos distintos tratamientos



## Clasificación de datos

Numérico	Categórico	Temporal	Booleanas
<b>Continuo (float):</b> altura, peso, salario, valor del dólar. <b>Entero (int):</b> unidades vendidas, clientes en una hora, autos en un minuto.	<b>Ordinal:</b> dominio habilidad (bajo, medio, alto), nivel de ingreso o rango edad (joven, adulto, etc.). <b>No ordinal:</b> comuna, profesión o color.	Una serie de tiempo es una secuencia de datos indexados y ordenados en el tiempo (día, hora, año, etc).	Variables que toman dos posibles valores (True o False). <b>Algunas ejemplos son:</b> - ¿Es mayor a 65 años? - ¿Es deudor? - ¿Falló este mes?

### b. Tipos de datos en pandas:

- i. En la lectura del Pandas, la opción **index\_col** indica la columna que va a ser llave, si no se coloca, el dataset se numera.

```

import pandas as pd

# index_col indica el indice
df = pd.read_csv('ejemplo_data.csv', index_col='ID')

df
  
```

ID	Nombre	2016	2017	Crecimiento	Unidades	fecha	Activo
10002	Verde Mar	\$125,000.00	\$162500.00	30.00%	500	1-10-2015	1
552278	Manantial sa	\$920,000.00	\$101,2000.00	10.00%	700	6-23-2014	0
23477	ACME	\$50,000.00	62500.00	25.00%	125	3-12-2016	1
24900	Andes sur	\$350,000.00	490000.00	4.00%	75	10-28-2015	1
651029	San Pablo	\$15,000.00	\$12750.00	-15.00%	No	2-15-2014	0

ii. **Dtypes** indica los tipos de datos.

```
# Indica los tipos de datos  
df.dtypes
```

Nombre	object
2016	object
2017	object
Crecimiento	object
Unidades	object
fecha	object
Activo	int64
dtype:	object

iii. El tipo **Object** indica cuando es string o una mezcla.

iv. Con **astype** se cambia el tipo de datos.

```
df["Activo"] = df["Activo"].astype('bool')
```

```
df.dtypes
```

Nombre	object
2016	object
2017	object
Crecimiento	object
Unidades	object
fecha	object
Activo	bool
dtype:	object

```
df["Nombre"] = df["Nombre"].astype('category')
```

v. Con el **replace** se pueden reemplazar caracteres.

vi. Con **apply** se aplica una función.

```
# Función para eliminar comas y signo peso  
def convertir_monto(val):  
    nuevo_data = val.replace(',', '').replace('$', '')  
    return float(nuevo_data)
```

```
# Función para eliminar porcentaje  
def convertir_porcentaje(val):  
    nuevo_data = val.replace('%', '')  
    return float(nuevo_data) / 100
```

```
df['2016'] = df['2016'].apply(convertir_monto)  
df['2017'] = df['2017'].apply(convertir_monto)  
df['Crecimiento'] = df['Crecimiento'].apply(convertir_porcentaje)
```

vii. Con el método **to\_numeric** de pandas se pueden transformar a numérico aceptando los errores, y colocándole un NaN cuando

aparezcan. Luego con **fillna** se rellenan los NaN por ejemplo con un 0.

```
df['Unidades'] = pd.to_numeric(df['Unidades'], errors='coerce').fillna(0)
```

ID	Nombre	2016	2017	Crecimiento	Unidades	fecha	Activo
10002	Verde Mar	125000.0	162500.0	0.30	500.0	1-10-2015	True
552278	Manantial sa	920000.0	1012000.0	0.10	700.0	6-23-2014	False
23477	ACME	50000.0	62500.0	0.25	125.0	3-12-2016	True
24900	Andes sur	350000.0	490000.0	0.04	75.0	10-28-2015	True
651029	San Pablo	15000.0	12750.0	-0.15	0.0	2-15-2014	False

viii. Con pandas posee **to\_datetime** se pasa a fecha.

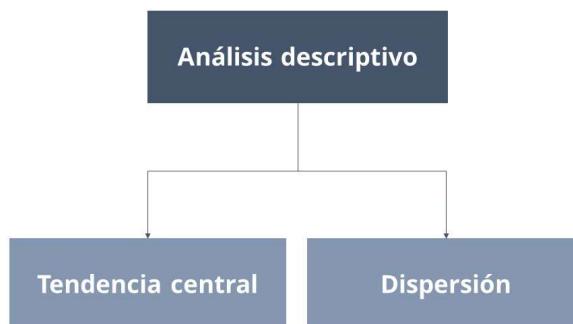
```
df['fecha'] = pd.to_datetime(df['fecha'])
```

ID	Nombre	2016	2017	Crecimiento	Unidades	fecha	Activo
10002	Verde Mar	125000.0	162500.0	0.30	500.0	2015-01-10	True
552278	Manantial sa	920000.0	1012000.0	0.10	700.0	2014-06-23	False
23477	ACME	50000.0	62500.0	0.25	125.0	2016-03-12	True
24900	Andes sur	350000.0	490000.0	0.04	75.0	2015-10-28	True
651029	San Pablo	15000.0	12750.0	-0.15	0.0	2014-02-15	False

## 6. Análisis descriptivos de variables

### a. Rol del análisis descriptivo de datos:

- Los datos reales vienen inconsistentes (variables con distintos valores según las bases), con ruido y/o incompletos.
- Dado el punto 1, hay que hacer una inspección inicial, como el Análisis Descriptivo.



iii. Una forma de representar una base de datos es:

```
1 import pandas as pd
2 d = {
3     'Nombre' : pd.Series(['Juan', 'Pedro', 'Ricardo', 'Viviana', 'Rosa', 'Soledad',
4     'Francisca',
5     'Leandro', 'David', 'Gaspar', 'Betina', 'Andrea'],
6     'Edad' : pd.Series([25,26,25,23,30,29,23,34,40,30,51,46]),
7     'Nota' : pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8,3.78,2.98,4.80,4.10,3.65])
8 }
9 df = pd.DataFrame(d)
```

b. **Medidas de tendencia central:**

- i. **Media:** Valor central de un conjunto de números, específicamente, la suma de los valores dividida por el número de valores.

$$\bar{x} = \frac{\sum_{i=1}^n x_i}{n}$$

```
1 print ('Edad promedio:', df ['Edad'].mean())
```

Edad promedio: 31.83333333333332

- ii. **Media aritmética con pesos:** Métrica de valor central de un conjunto de números apropiada cuando cada uno de ellos tiene una importancia relativa respecto a los demás datos.

$$\bar{x} = \sum_{i=1}^n w_i x_i$$

```
1 import numpy as np
2 df ['pesos'] =
3 pd.Series([1/20,1/20,1/20,1/20,1/20,1/20,1/20,1/20,1/20,1/20,5/20,5/2
0])
4 promedio_ponderado = np.average(df['Edad'],weights=df['pesos'])
5 print('Promedio ponderado:', promedio_ponderado)
```

- iii. Para excluir los datos atípicos se puede usar la **media truncada**, que es el promedio aritmético obtenido luego de descartar porciones de la distribución de probabilidades o muestra en el extremo inferior o superior.
- iv. Con **shape**, cuenta la cantidad de registros.

```

1 print (df.shape)
2 df = df.drop(df.index[[10,11]])
3 print(df.shape)
4 df

```

(12, 4)  
(10, 4)

	Nombre	Edad	Nota
0	Juan	25	4.23
1	Pedro	26	3.24
2	Ricardo	25	3.98
3	Viviana	23	2.56
4	Rosa	30	3.20
5	Soledad	29	4.60
6	Francisca	23	3.80
7	Leandro	34	3.78
8	David	40	2.98
9	Gaspar	30	4.80

```

1 print('edad media', df.Edad.mean())
2 print('nota media', df.Nota.mean())

```

edad media 28.5  
nota media 3.717

- v. **Moda** (Número que aparece más frecuentemente en la base de datos) y **mediana** (Valor de la variable de interés que ocupa el valor central o las dos posiciones de los datos ordenados):

```

1 df.Edad.mode()

```

0	23
1	25
2	30
	dtype: int64

```

1 df.Nota.median()

```

3.79

### c. Medidas de dispersión:

- i. **Rango:** Permite evaluar la dispersión de los datos a través de la diferencia entre sus extremos. Su desventaja es que no incorpora información más allá de los valores extremos.

$$rango = x_{max} - x_{min}$$

```

1 print('rango nota: ', np.round(df.Nota.max()-df.Nota.min(),2))
2 print('rango edad: ', np.round(df.Edad.max()-df.Edad.min(),2))
Rango nota: 2.24
Rango edad: 17

```

- ii. Varianza:** Mide la dispersión de los valores respecto a un valor central (media).

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

```

1 print('Varianza nota: ', np.round(df.Nota.var(),2))
2 print('Varianza edad: ', np.round(df.Edad.var(),2))
Varianza nota: 0.52
Varianza edad: 28.72

```

- iii. Alternativa a la varianza es la Desviación estándar:** Informa sobre la dispersión de los datos respecto al valor de la media; cuando mayor sea su valor, más dispersos estarán los datos.

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

```

1 print('desviación estándar nota: ', np.round(df.Nota.std(),2))
2 print('desviación estándar edad: ', np.round(df.Edad.std(),2))
Desviación estándar nota: 0.72
Desviación estándar edad: 5.36

```

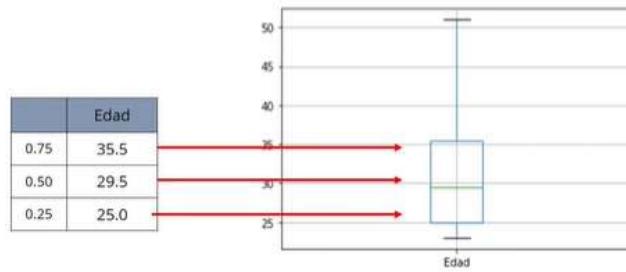
- iv. Una ventaja de la desviación estándar sobre la varianza es que se expresa en las mismas unidades del atributo de interés.**
- v. Cuantil:** Para un set de observaciones ordenadas, los cuantiles son puntos que dividen la distribución en intervalos que representan la misma proporción de valores.

	Edad	Nota
0.25	25.0	3.2300
0.50	29.5	3.7900
0.75	35.5	4.1325

```

1 import matplotlib.pyplot as plt
2 plot = df[['Edad', 'Nombre']]
3 plot.boxplot()
4 plt.show()

```



- vi. **Covarianza** permite determinar la variabilidad de 2 atributos en conjunto. Es la extensión de la varianza que divide los errores respecto a dos atributos.
- vii. **Covarianza**, indica el grado de variación conjunta de dos variables aleatorias respecto a sus medias. Permite determinar dependencias y correlaciones.

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

Nota promedio  
Edad promedio

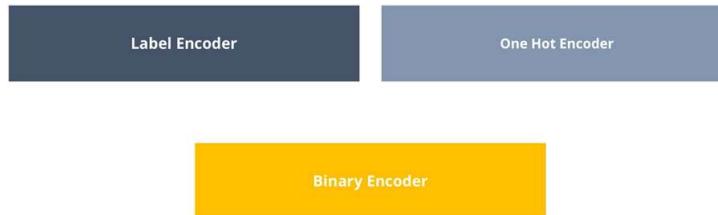
	Edad	Nota
Edad	85.242424	0.357879
Nota	0.357879	0.437752

## 7. Transformación de variables

- a. El preprocessamiento de la base de datos es fundamental.
- b. **Tratamiento de datos categóricos:**
  - i. Ejemplo, de lo que se quiere resolver, lo que se quiere es convertir las variables categóricas en números.



## ii. Escalar/estandarizar atributos:



## iii. Ejemplo 1: Transformar en atributo binario cada categoría.

```
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelBinarizer
import pandas as pd

# Variable a transformar, posee 3 categorías (Santiago, Curicó y Talca)
comuna = ['Santiago', 'Talca', 'Curicó', 'Curicó', 'Talca', 'Santiago']

# Se crea un objeto de la clase LabelBinarizer, si pertenece al objeto toma
# el valor 1, si no el valor 0
encoder = LabelBinarizer(neg_label=0, pos_label=1)
# Se hace un ajuste a los datos y se detectan las clases o categorías
binaryEncode = encoder.fit(comuna)
# Se imprimen las clases o categorías
print(binaryEncode.classes_)
# Se realiza la transformación
binaryEncode.transform(comuna)
# genera un matriz donde cada columna identica un categoría

['Curicó' 'Santiago' 'Talca']
array([[0, 1, 0],
       [0, 0, 1],
       [1, 0, 0],
       [1, 0, 0],
       [0, 0, 1],
       [0, 1, 0]])
```

## iv. Ejemplo 2:

```
# Se genera el objeto
encoder_comuna = LabelEncoder()
# Se hace un ajuste y transformación a comuna
encoder_comuna.fit_transform(comuna)
# En este caso mantiene la dimensión e indica el número
# de la categoría (0: Curicó, 1: Santiago, 2: Talca)

array([1, 2, 0, 0, 2, 1], dtype=int64)

# Muestra las categorías o clases
encoder_comuna.classes_

array(['Curicó', 'Santiago', 'Talca'], dtype='<U8')

# Se solicita el valor de la categoría, ejemplo la 0
encoder_comuna.inverse_transform([0])

array(['Curicó'], dtype='<U8')
```

## v. Ejemplo 3:

```
color = ['rojo', 'verde', 'rojo', 'amarillo', 'azul', 'verde']
comuna = ['Santiago', 'Talca', 'Curicó', 'Curicó', 'Talca', 'Santiago']

# Transformamos una lista de tuplas
list(zip(comuna, color))

[('Santiago', 'rojo'),
 ('Talca', 'verde'),
 ('Curicó', 'rojo'),
 ('Curicó', 'amarillo'),
 ('Talca', 'azul'),
 ('Santiago', 'verde')]

# Generamos el DataFrame
df = pd.DataFrame(list(zip(comuna, color)), columns=['comuna', 'color'])

# Creamos el objeto
encoder = OneHotEncoder()
# Hacemos el ajuste y transformación
comuna_onHot = encoder.fit_transform(df)

# Muestra las categorías (7, 3 a las comunas y 4 a los colores,
# las cuales son dos objetos array
encoder.categories_

[array(['Curicó', 'Santiago', 'Talca'], dtype=object),
 array(['amarillo', 'azul', 'rojo', 'verde'], dtype=object)]

# Transformamos en un array (primera fila: 1: santiago, 1: rojo, que
# corresponde a la primera tupla mostrada arriba)
comuna_onHot.toarray()

array([[0., 1., 0., 0., 0., 1., 0.],
       [0., 0., 1., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 1., 0.],
       [1., 0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 1., 0., 0.],
       [0., 1., 0., 0., 0., 0., 1.]])
```

## c. Escalamiento y normalización:

- i. Otro preprocesamiento es llevar a todos los atributos numéricos a rango similar, ej. Entre 0 y 1 o -1 y 1. Esto se hace con el objetivo de que aquellos atributos que tiene mayor magnitud no dominen el modelo de aprendizaje de máquina.
- ii. Se estudiará:

Escalamiento MinMax

Normalización z-score

- iii. **MinMax:** Permite transformar los atributos conociendo los extremos originales y los extremos nuevos. El rango deseado es 0 y 1 o -1 y 1.

- Realiza una transformación lineal en los datos originales.
  - Si los **valores extremos iniciales** son **min** y **max**, y los **valores extremos finales** son **newMin** y **newMax**, la transformación pasa de un valor  $x$  a un valor  $x'$ .

$$x' = \underbrace{\frac{x - min}{max - min}}_{\text{Rango inicial}} * \underbrace{(newMax - newMin)}_{\text{Rango final}} + newMin$$

```
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Hacer el minmax nosotros con numpy

# Conjunto de datos
X = np.array([1.1, 2.5, 3.2, 1.0, 4.5])
# Mínimo y máximo deseado
newMin = 0
newMax = 1
# Fórmula
X_scaled = (X - np.min(X)) / (np.max(X) - np.min(X)) * (newMax - newMin) + newMin
# Se obtiene los datos normalizados entre 0 y 1
X_scaled

array([0.02857143, 0.42857143, 0.62857143, 0. , 1.        ])

# Lo mismo pero usando MinMaxScaler

# Se crea el objeto y se le da el rango
minMax = MinMaxScaler(feature_range=[0,1])
# Se hace el ajuste y transformación
xNew = minMax.fit_transform(np.asarray(X).reshape(-1,1))
# Se llega al mismo resultado
xNew

array([[0.02857143],
       [0.42857143],
       [0.62857143],
       [0. ],
       [1. ]])
```

iv. **Z-Score:** Lleva los valores originales a valores normal estándar (0, 1).

Los valores para un atributo X son normalizados en base a la media y la desviación estándar.

$$x' = \frac{x - \bar{X}}{\sigma_X}$$

```

# Importamos la clase y pandas
from sklearn.preprocessing import StandardScaler
import pandas as pd
# Generamos el DataFrame
df = pd.DataFrame([1.1, 2.5, 3.2, 1.0, 4.5])
# Generamos el objeto
ss = StandardScaler()
# Realizamos la transformación
ss.fit_transform(df)

array([[-1.0314877 ],
       [ 0.03033787],
       [ 0.56125066],
       [-1.10733238],
       [ 1.54723154]])

```

## 8. Visualización de variables

- Visualización como primer paso del análisis
- Tipos de visualización de variables continuas
- Tipos de visualización de variables categóricas

### a. Objetivos:

#### Analizar estructura

- Distintas visualizaciones ayudan a estudiar la distribución de los datos. Nos da una pista de cómo proceder en el análisis.

#### Detectar fallas o sesgos

- Graficar los datos siempre ayuda a encontrar posibles errores en el formato. Además, permite evidenciar sesgos que perjudiquen el análisis.

#### Detectar patrones

- Nuestro cerebro es mejor que cualquier algoritmo para encontrar patrones. Nos permite ahorrar tiempo y trabajo.

### b. Variables continuas:

#### Creamos datos en 2D

Usando "**multivariate\_normal**" de la librería "scipy.stats", nos permite **crear una nube de puntos**.

```

from scipy.stats import multivariate_normal
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

data1 = multivariate_normal(mean=[0,0], cov=1).rvs(1000)
data1 = np.hstack([data1, np.zeros((data1.shape[0],1))])
data1 = pd.DataFrame(data1, columns=['x','y','c'])

data2 = multivariate_normal(mean=[3,3], cov=1).rvs(1000)
data2 = np.hstack([data2, np.ones((data2.shape[0],1))])
data2 = pd.DataFrame(data2, columns=['x','y','c'])

data = np.concatenate([data1,data2],axis=0)
data = pd.DataFrame(data, columns=['x','y','c'])

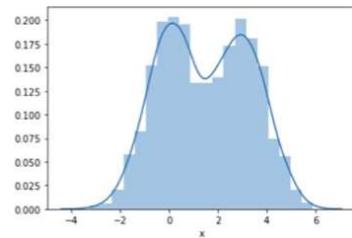
```

## Histogramas 1D

Podemos ver que la variable "x" es bimodal.

Si el formato de los datos fuera incorrecto, la función retornaría un error.

```
sns.distplot(data['x'], bins=20, kde=True, rug=False);
```

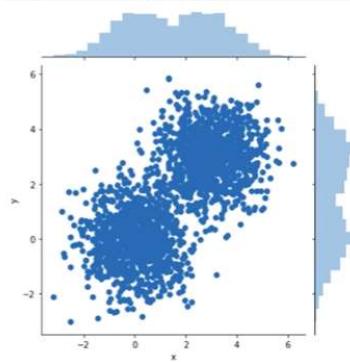


## Dispersión 2D

Un gráfico de dispersión permite ver la distribución de los datos en dos dimensiones.

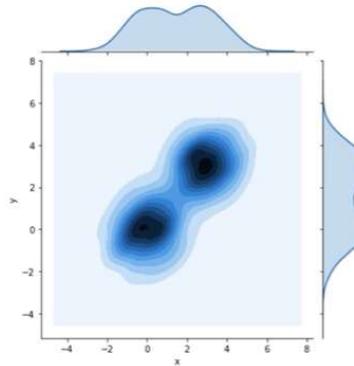
Los histogramas nos permiten ver la proyección en cada una de las variables.

```
sns.jointplot(x='x',y='y', data=data);
```

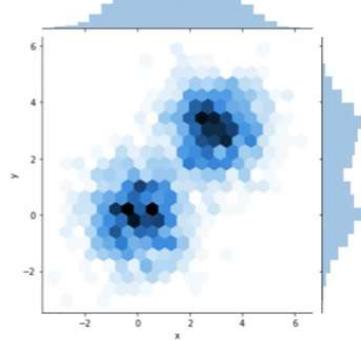


## Otras visualizaciones 2D

```
sns.jointplot(x='x',y='y', data=data, kind="kde");
```



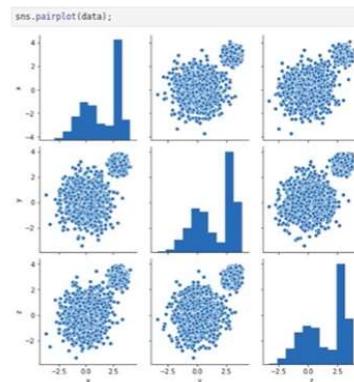
```
sns.jointplot(x='x',y='y', data=data, kind="hex");
```



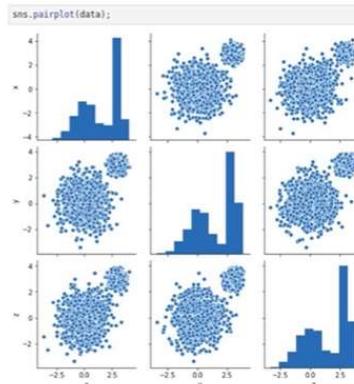
## Visualizaciones 3D

Se puede graficar una **matriz de gráficos de dispersión**.

En la diagonal vemos el **histograma de la variable**.



Siempre preferir una **visualización 2D** a una isométrica.

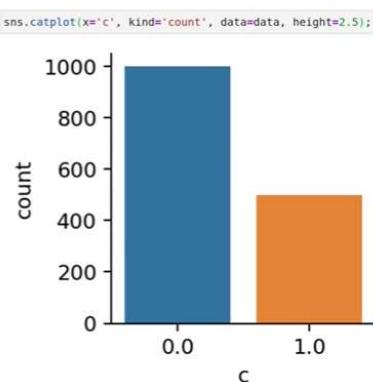


### c. Variables categóricas:

#### Histogramas

Permite ver la **frecuencia** de las variables.

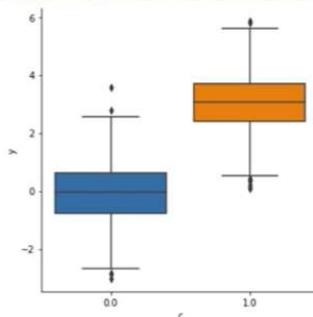
Desbalance en los datos serán **obvios**.



## **Box plots**

Permite ver la distribución de una variable continua 1D data una variable discreta.

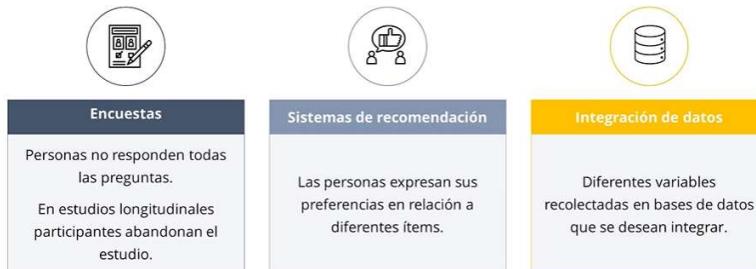
```
sns.catplot(x="c", y="y", kind="box", data=data);
```



## **9. Imputación de datos**

### **a. ¿Qué es la imputación de datos?**

- i. Un dato perdido se asocia a vacíos en la base de datos. Esto puede limitar el uso de la información.
- ii. Muchos algoritmos no pueden soportar la perdida de datos.
- iii. Si no se imputan, se deberá eliminar filas y columnas.

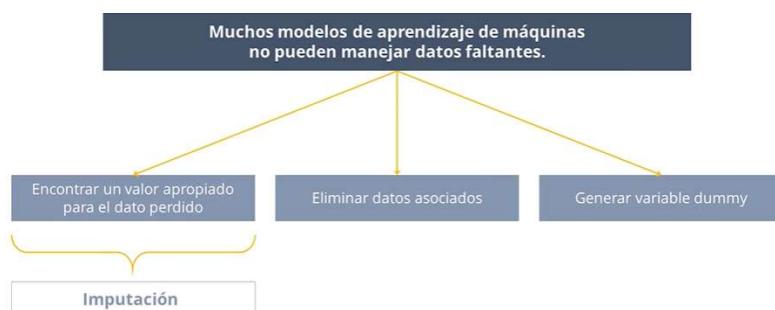


### **iv. Fuente de valores perdidos: La naturaleza de los datos perdidos.**



- v. Se considera datos perdidos completamente aleatorio cuando no están relacionado con ninguna variable presente o no en el dataSet.

- vi. Se considera datos perdidos aleatorio, cuando asume que existe una razón predecible, es decir, son causados por un efecto aleatorio estimable. Ej. Cuando parte de la población se niega a responder ciertos estudios.
- vii. Datos no aleatorios, la razón se encuentra en la variable en sí misma. No hay información disponible para estimar. Ejemplo, consultar consumo de tabacos a adolescentes, la pregunta por si sola es la causa de los datos perdidos.
- viii. Problema > Solución:

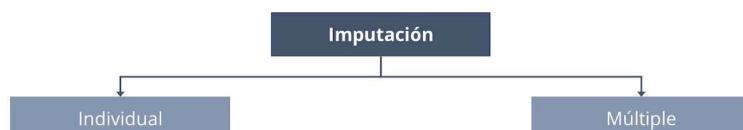


#### ix. Eliminar datos asociados:



#### b. Métodos básicos de imputación

- i. Considerar si la imputación considera información de otros atributos.



## ii. Imputación Individual:



1. Imputación basada en estadísticas de tendencia central (Media o Moda): Modifica la distribución de probabilidad del atributo en el cual estamos haciendo la imputación, pero asigna un valor que es altamente probable para ese atributo.
2. Aleatorio desde columnas: No modifica la distribución, pero puede aplicar valores poco probables.
3. Regresión: Tratar de generar una predicción donde las variables independientes serán otras variables de la base de datos.

## iii. Imputación Múltiple:



c. Estrategias de imputación con Python: Buscar estrategias de imputación avanzadas.

```

import numpy as np
import pandas as pd

df1 = pd.read_csv('dataimputar.csv')

# Aquí se puede apreciar los datos perdidos (NaN)
df1

```

	A	B	C	D
0	1.0	NaN	2.0	Curicó
1	0.0	3.0	5.0	Pichilemu
2	1.0	3.5	NaN	Santiago
3	NaN	4.0	6.0	Santiago
4	0.0	2.8	2.0	Chiloé
5	0.0	3.1	3.0	Curicó
6	1.0	5.2	4.0	Santiago
7	0.0	0.0	2.0	Pichilemu
8	1.0	NaN	NaN	Santiago
9	1.0	NaN	2.1	Santiago

```

df1.shape
(10, 4)

```

```

# Opción 1: Eliminación de filas con datos perdidos
df_clean = df1.dropna()
df_clean

```

	A	B	C	D
1	0.0	3.0	5.0	Pichilemu
4	0.0	2.8	2.0	Chiloé
5	0.0	3.1	3.0	Curicó
6	1.0	5.2	4.0	Santiago
7	0.0	0.0	2.0	Pichilemu

```

# Opción 2: Eliminación de columnas con datos perdidos
df_clean = df1.dropna(axis='columns')
df_clean

```

	D
0	Curicó
1	Pichilemu
2	Santiago
3	Santiago
4	Chiloé
5	Curicó
6	Santiago
7	Pichilemu
8	Santiago

```
# Opción 3: eliminar con condiciones (filtro), en este ejemplo,
# Eliminar columnas que tengan menos de 8 datos validos.
df_clean = df1.dropna(thresh=8, axis='columns')
df_clean
```

	A	C	D
0	1.0	2.0	Curicó
1	0.0	5.0	Pichilemu
2	1.0	NaN	Santiago
3	NaN	6.0	Santiago
4	0.0	2.0	Chiloé
5	0.0	3.0	Curicó
6	1.0	4.0	Santiago
7	0.0	2.0	Pichilemu
8	1.0	NaN	Santiago
9	1.0	2.1	Santiago

```
# Opción 4: Otro filtro de ejemplo, eliminar todas las filas que
# tengan casos perdidos pero solo de las filas A y B
df_clean = df1.dropna(subset=['A', 'B'])
df_clean
```

	A	B	C	D
1	0.0	3.0	5.0	Pichilemu
2	1.0	3.5	NaN	Santiago
4	0.0	2.8	2.0	Chiloé

```
# Ver cuantos valores nulos tiene la columna D
df1['D'].isnull().sum()
```

0

```
# Lo mismo pero en todas las columnas
for i in df1.columns:
    print(i + ' : ' + str(df1[i].isnull().sum()))
```

A : 1  
B : 3  
C : 2  
D : 0

```
# Llenado de NaN usando backfill, es decir, reemplaza
# el NaN por el valor de la fila inmediatamente siguiente
df1.fillna(method="backfill")
```

	A	B	C	D
0	1.0	3.0	2.0	Curicó
1	0.0	3.0	5.0	Pichilemu
2	1.0	3.5	6.0	Santiago
3	0.0	4.0	6.0	Santiago
4	0.0	2.8	2.0	Chiloé
5	0.0	3.1	3.0	Curicó
6	1.0	5.2	4.0	Santiago
7	0.0	0.0	2.0	Pichilemu
8	1.0	NaN	2.1	Santiago

```
# Llenado de NaN usando ffill, es decir, reemplaza
# el NaN por el valor de la fila inmediatamente anterior
df1.fillna(method="ffill")
```

	A	B	C	D
0	1.0	NaN	2.0	Curicó
1	0.0	3.0	5.0	Pichilemu
2	1.0	3.5	5.0	Santiago
3	1.0	4.0	6.0	Santiago
4	0.0	2.8	2.0	Chiloé
5	0.0	3.1	3.0	Curicó
6	1.0	5.2	4.0	Santiago
7	0.0	0.0	2.0	Pichilemu
8	1.0	0.0	2.0	Santiago
9	1.0	0.0	2.1	Santiago

```
# Se puede aplicar backfill y ffill en conjunto
df1.fillna(method="ffill").fillna(method="bfill")
```

	A	B	C	D
0	1.0	3.0	2.0	Curicó
1	0.0	3.0	5.0	Pichilemu
2	1.0	3.5	5.0	Santiago
3	1.0	4.0	6.0	Santiago
4	0.0	2.8	2.0	Chiloé

```
# Se puede imputar con la media de cada columna, ejemplo
df1.fillna(value=df1.mean(numeric_only=True))
```

	A	B	C	D
0	1.000000	3.085714	2.0000	Curicó
1	0.000000	3.000000	5.0000	Pichilemu
2	1.000000	3.500000	3.2625	Santiago
3	0.555556	4.000000	6.0000	Santiago
4	0.000000	2.800000	2.0000	Chiloé
5	0.000000	3.100000	3.0000	Curicó
6	1.000000	5.200000	4.0000	Santiago
7	0.000000	0.000000	2.0000	Pichilemu
8	1.000000	3.085714	3.2625	Santiago
9	1.000000	3.085714	2.1000	Santiago

## Módulo III

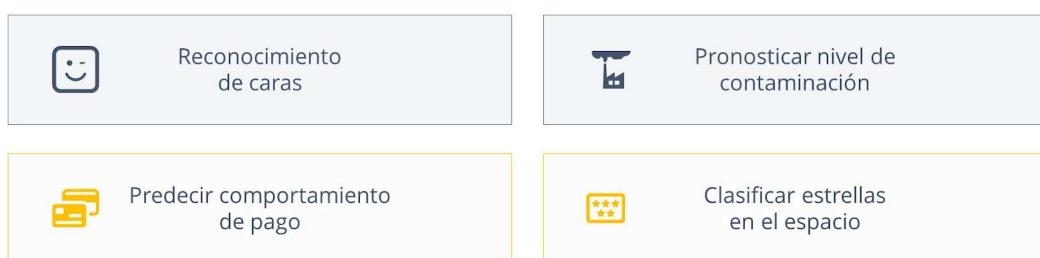
### 10. Aprendizaje supervisado

- a. Son algoritmos que aprenden desde una base de datos con pares del tipo **input/output**.
- b. El input son los descriptores y el output es clase.
- c. También, se puede entender el aprendizaje supervisado como el ajuste de una distribución de probabilidades. Luego se quiere encontrar la clase de mayor probabilidad con un input dado.

- d. Dentro del aprendizaje supervisado se pueden aprender 2 tipos de funciones, una función que aprende un Output discreto como es el caso de los métodos de clasificación y un Output continuo como las regresiones.



- e. Algunos ejemplos de este aprendizaje son:



- f. Los modelos para estudiar son los siguientes, con estos modelos podremos estudiar conceptos de sobreajuste y penalización:



- g. El segundo clasificador que estudiar es **Naive Bayes**, basado en el teorema de Bayes, simple, pero ha demostrado ser muy potente para algunas tareas de clasificación.

h. La tercera familia es el árbol de decisión, la última familia son las redes neuronales.



- 1 Árbol de decisión
- 2 Random Forest para clasificación
- 3 Random Forest para Regresión

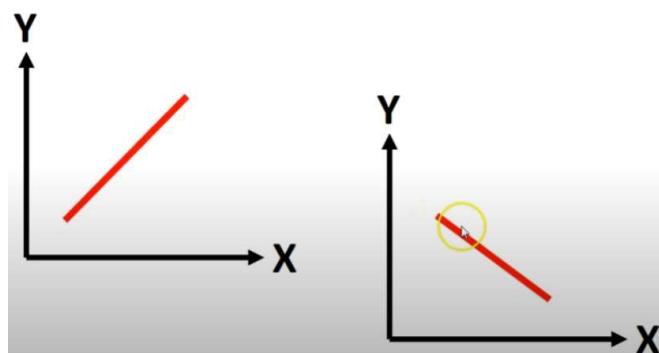


- 1 Perceptrón Simple
- 2 Redes Recurrentes
- 3 Redes Convolucionales

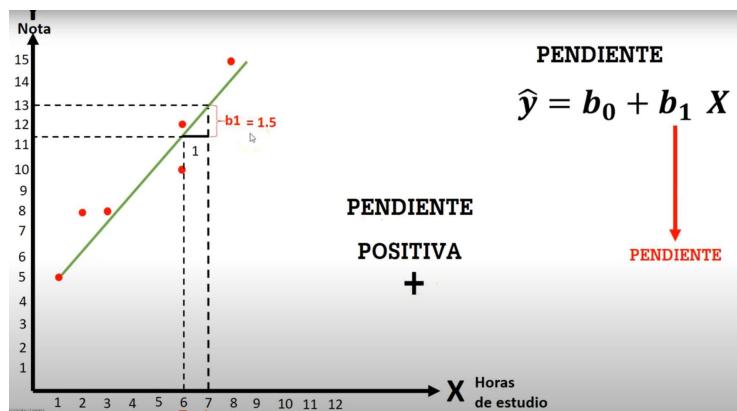
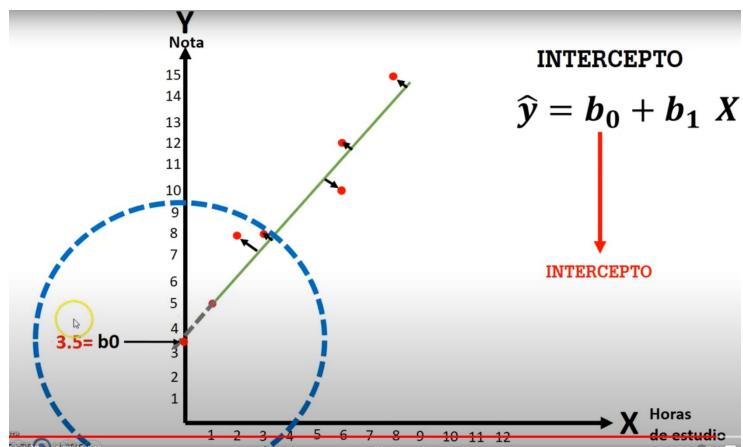
## 11. Regresión lineal

### a. Explicación básica:

- i. Es una técnica estadística utilizada para predecir o estimar una variable **cuantitativa** en función de otra variable **cuantitativa**.
- ii. Se usan dos variables, **Y** llamada dependiente porque depende de **X**, y **X** llamada independiente porque explica a **Y**.
- iii. **Y** es la variable que deseamos predecir o estimar.
- iv. **X** es la variable explicativa.
- v. La técnica de regresión lineal consiste en modelar una ecuación de una recta.



- vi. Si la variable **X** e **Y** suben se podría decir que tienen una relación directa.
- vii. Si la variable **X** sube e **Y** baja o viceversa se podría decir que tienen una relación inversa.
- viii. Se puede generar un gráfico de dispersión.
- ix. Existen infinitas rectas posibles que pasan cerca de los 6 puntos.
- x. Para calcular la mejor recta la usa la ecuación estimada  $\hat{y} = b_0 + b_1X$
- xi. Donde  $\hat{y}$  y **X** son variables y  $b_0$  con  $b_1$  son constantes.
- xii.  $b_0$  se conoce como interceptor y  $b_1$  es la pendiente.



xiii. La pendiente puede ser positiva o negativa.

**b. Modelo de regresión lineal:**

- Hacen una predicción usando una función lineal de los descriptores.
- Descriptor  $x_1$ , variable dependiente y.

Variable dependiente	Parámetros del modelo
$y = \theta_0 + \theta_1 x_1 + e$	
Intercepto o Bias	Descriptores/variables

- En el modelo anterior se supone que existe información de  $x$  e  $y$ , y el aprendizaje se realiza sobre  $\Theta_0$  y  $\Theta_1$ .

- iv. Para identificar los estimadores de los parámetros del modelo, identificaremos los estimadores con  $\hat{\Theta}$ .

Predicción	Estimadores de pesos
$\hat{y} = \hat{\theta}_0 + \hat{\theta}_1 x_1$	
Intercepto o Bias	Descriptores/variables

- v. Regresión lineal con n descriptores,

$$\hat{y} = \hat{\theta}_0 + \hat{\theta}_1 x_1 + \dots + \hat{\theta}_i x_i + \dots + \hat{\theta}_n x_n$$

$\bar{y}$  = predicción

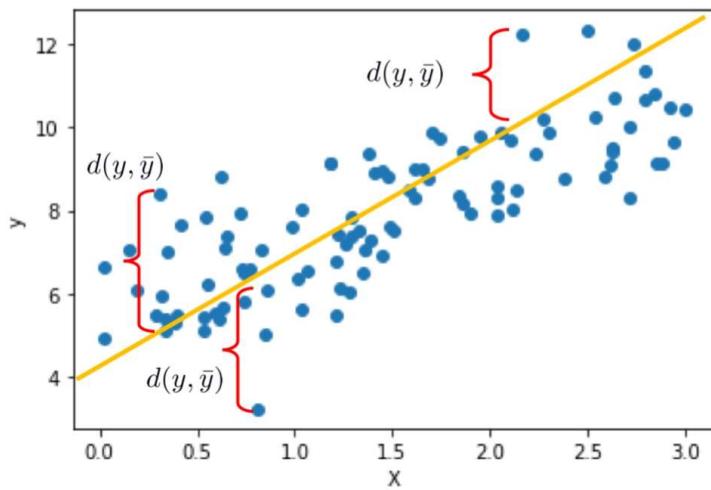
$n$  = número de descriptores

$x_i$  = valor del descriptor  $i$

$\hat{\theta}_j$  =  $j$ -ésimo parámetro del modelo

### c. Obtención e interpretación de parámetros:

- i. Para entender el ajuste de los parámetros  $\Theta$ , se debe definir una medida de calidad del modelo para cada punto dentro de la base de datos. En este caso se define la distancia de predicción del punto y su valor real.
- ii. La línea amarilla representa el modelo o las predicciones y cada punto azul representa el valor real.
- iii. Cada combinación de parámetros generará una distancia diferente entre el modelo y el valor real.



- iv. Si se considera solo la diferencia entre la predicción y el valor real, podríamos obtener diferencias negativas y positivas. Para evitar que estas se contrarresten se ocupara el error cuadrático medio.

**Predicción**

$$\frac{1}{m} \sum_{i=1}^m (\hat{\theta}^T \mathbf{x}_i - y_i)^2 \quad \left. \right\} \text{Mínimo error cuadrático medio}$$

**Solución cerradas**

$$\hat{\theta} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

- v. Lo que se quiere hacer es encontrar el mínimo valor del error, a través de la correcta selección de los parámetros  $\Theta$ , luego el problema se traduce el optimizar el error cuadrático medio.
- vi. En la solución cerrada X representa la base de datos completa e y representa el vector de las variables dependientes.

**d. Predicción con regresión lineal usando Python:**

i. Supuestos:

1. Relación lineal.
2. No hay multicolinealidad.
3. Distribución normal.
4. Libre de ruido.
5. Homocedasticidad.

ii. Algunas aplicaciones:

1. Predicción de demanda.
2. Rendimiento académico.
3. Satisfacción de clientes.

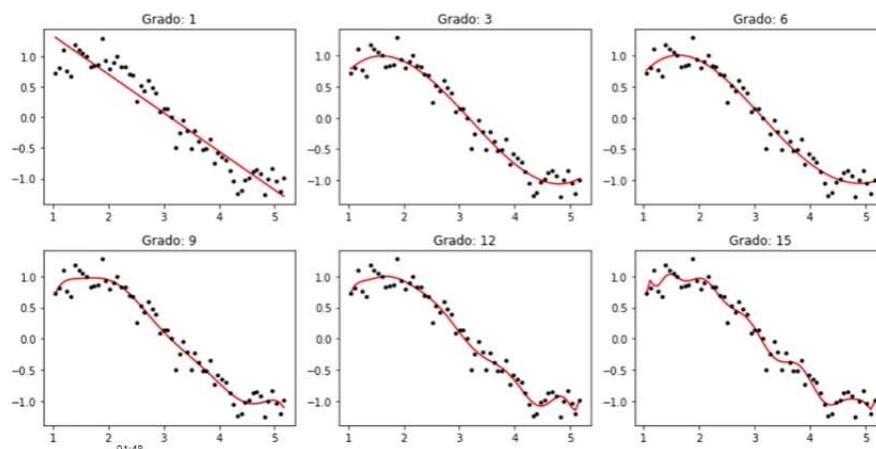
## 12. Regresiones polinomiales

a. Regresión polinomial:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \dots + \theta_{p-1} x_1^{p-1} + \theta_p x_1^p$$

- i. Este modelo sigue siendo lineal en los parámetros, sin embargo, se puede aumentar su expresividad por medio de una transformación (polinomial al input).

ii. Ejemplo:

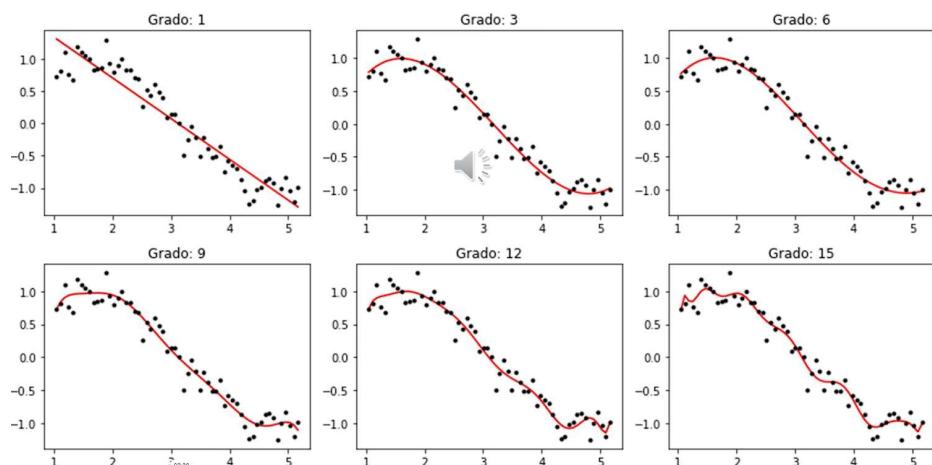


1. A través de diferentes regresiones polinomiales se obtienen diferentes niveles de ajuste de modelado de datos.
2. A medida que aumenta el grado del polinomio, el modelo naturalmente se ajustara a los datos de forma más compleja.

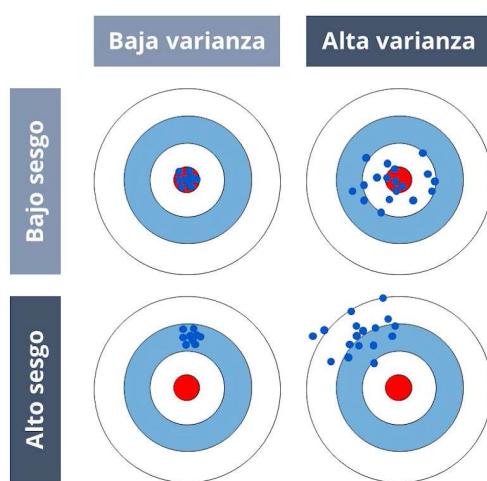
**b. Implementación en Python.**

### 13. Regresiones con penalización

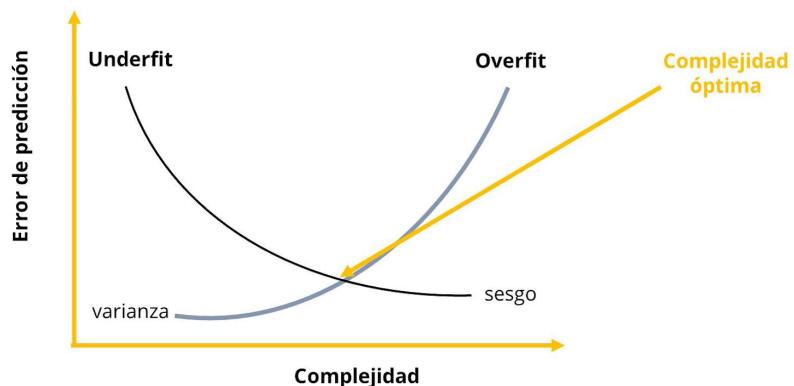
**a. Complejidad del modelo:**



**i. Sesgo – varianza en modelos:**



- ii. Si asumimos que la distancia entre los puntos azules al círculo rojo es el error de predicción, entonces:
  - 1. El escenario ideal es el caso de bajo sesgo – baja varianza.
  - 2. Bajo sesgo – alta varianza, quiere decir que los puntos azules están cerca del círculo rojo, pero hay mayor dispersión entre ellos.
  - 3. Entre otros.
- iii. Modelos con baja complejidad tienden a tener alto sesgo, pero baja varianza. Y mientras se aumenta la complejidad se reduce el sesgo, y se aumenta la varianza.



- iv. A medida que se aumenta la complejidad de los modelos se tiende a sobre ajustar, es decir, aprender los datos de memoria.

Cuando se ajusta un modelo altamente flexible, se debe tener cuidado con sobreajustarse a los datos de entrenamiento.

Es decir, se debe evitar que el modelo aprenda patrones leves del input, debido a que ellos pueden ser solo producto de ruido y no de los patrones a aprender.

- v. Navaja de Occam: "Si dos modelos explican de igual manera, se debe seleccionar el modelo que lo haga de forma más simple"
- vi. La Navaja de Occam quiere decir que si dos modelos tienen el mismo error de predicción se debe seleccionar al que tiene menor complejidad.

**b. Regresiones con penalización:**

- i. Limitan los valores posibles que pueden tomar los parámetros de las regresiones. Bajo el supuesto de que al acotar esos valores se reduce la flexibilidad y la complejidad de los modelos.

L1 - Lasso	L2 - Ridge
$\min \frac{1}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^i - y^i)^2$ <p>sujeto a:</p> $\sum_{n=1}^N  \theta_n  \leq \alpha$	$\min \frac{1}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^i - y^i)^2$ <p>sujeto a:</p> $\sum_{n=1}^N \theta_n^2 \leq \alpha$

- ii. **L1** y **L2** se diferencian en la estrategia que usan para acotar los valores posibles de los parámetros.
- iii. **L1**, acota a través de la sumatoria de los valores absolutos.
- iv. **L2**, a través de la suma de los pesos al cuadrado.
- v. A medida que se aumenta el Alpha los modelos se hacen más suaves y simples.
- vi. En la práctica:

$$\begin{aligned}
 \text{Minimizar} & \left[ \underbrace{\frac{1}{m} \sum_{i=1}^m (\theta^T \cdot \mathbf{x}^i - y^i)^2}_{\text{Mínimo MSE}} + \lambda f(\theta) \right] \\
 & \quad \text{Predicción} \\
 & \quad \text{Penalización} \\
 \text{Ejemplos} & \quad \sum_{n=1}^N |\theta_n| \quad \text{y} \quad \sum_{n=1}^N \theta_n^2
 \end{aligned}$$

**c. Implementación con Python.**

## 14. Regresión logística

### a. Conceptos básicos de regresión:

- i. El problema de clasificación en aprendizaje de máquina plantea el desafío de separar objetos dentro de una base de datos de acuerdo con su clase.

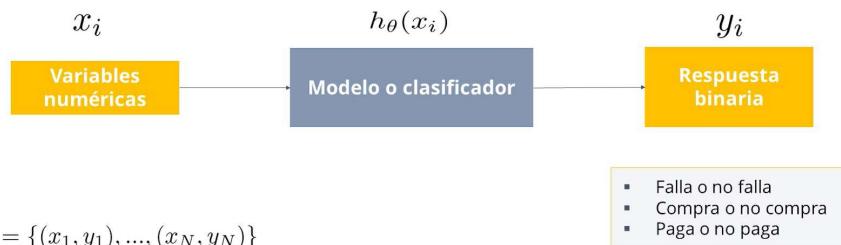
Decidir a qué clase pertenece un objeto dentro de un conjunto de posibles clases.

(2 clases para el caso de clasificación binaria)

Comportamiento de pago	Detección de planetas	Participación en elecciones
 Evaluar, de acuerdo a su comportamiento histórico de pago, si un cliente tiene alta o baja probabilidad de pagar un crédito.	 Identificar desde una base de datos de imágenes astronómicas si un objeto particular tiene alta probabilidad de ser un planeta.	 Determinar si un ciudadano participará en las próximas votaciones, considerando su historial de votaciones y características socio-demográficas.

- ii. Los 3 ejemplos anterior son clasificación binaria.

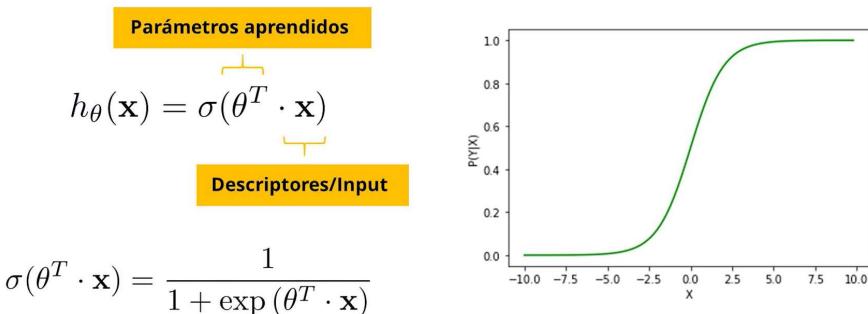
### iii. desafío:



1. Base de datos denotada por “D” y pares x, y.
2. Donde x representa los descriptores e y la etiqueta.
3. Dicha base de datos se usa para ajustar una función llamada clasificador.
4. En clasificación binaria se entrena al clasificador con un output binario, donde 1 representa la clase de interés.
5. Ej. Determinar si una maquina fallara en el futuro o no, o si un cliente compra un producto determinado o no, o si el cliente pagara el crédito otorgado.

## b. Regresión logística:

- i. En las regresiones lineales el output es un número real, por lo tanto, se limita la posibilidad de utilizar dichos modelos para la tarea de clasificación.
- ii. Para resolver el punto anterior, en las regresiones logísticas se usa una función que permite acotar el output a valores entre 0 y 1. En particular se usa la función  $\sigma$  para realizar dicha tarea.
- iii. La función anterior solo depende de theta y x que son los que deseamos aprender.



- iv. Para encontrar los pesos en una regresión logística, un primer paso consiste en representar la probabilidad de cada dato de pertenecer a la clase de interés. Esto se puede hacer a través de la distribución de Bernoulli. Dicha distribución permite modelar la probabilidad de éxito o fracaso de un evento, en este caso el éxito representa la pertenencia a la clase de intereses. Como se puede ver en la formula el valor de la función  $\sigma$  representa la probabilidad de pertenecer a la clase, que es el parámetro que describe la distribución de Bernoulli, luego, el desafío considera encontrar los valores Theta que esta dentro de la función sigmoide.

$$p(y_i|x_i, \theta) = \sigma(\theta^T \cdot x_i)^{y_i} (1 - \sigma(\theta^T \cdot x_i))^{1-y_i}$$

$p(y_i = 1)$        $p(y_i = 0)$   
 $\text{Ber}(y_i | \sigma(x_i, \theta))$

- v. Utilizando la probabilidad de cada dato, se puede crear la función de verosimilitud. Que corresponde al producto de las probabilidades de todos los datos de la muestra, después de un procesamiento tradicional podemos representar la función de verosimilitud para la regresión logística, tal como se representa en la función  $J(\Theta)$ . Finalmente se aplica una estrategia de optimización para encontrar los parámetros que maximizan la función de verosimilitud, es decir que maximizan la probabilidad de los datos.

$$J(\theta) = -\frac{1}{N} \sum_i^N y_i \log \sigma(\theta^T \cdot x_i) + (1 - y_i) \log(1 - \sigma(\theta^T \cdot x_i))$$

Función de pérdida = - Logaritmo de función de verosimilitud

$$\hat{\theta} = \arg \min_{\theta \in \Theta} J(\theta)$$

### c. Implementación y clasificación:

- i. Dentro del modulo **datasets** de sklearn se encuentran diversos conjuntos de datos experimentales.

```

1 from sklearn import datasets
2 import pandas as pd
3 iris = datasets.load_iris()
4 data = pd.DataFrame(iris['data'], columns=['LargoSépalo', 'AnchoSépalo', 'LargoPétalo', 'AnchoPétalo'])
5 data['clase'] = iris['target']
6 data.head()

```

	LargoSépalo	AnchoSépalo	LargoPétalo	AnchoPétalo	clase
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

clases

setosa = 0  
versicolor = 1  
virginica = 2

```

1 X = iris[ "data" ][:,3:]
2 y = (iris["target"]==2).astype(np.int)

```

```

3 from sklearn.linear_model import LogisticRegression
4 regLog = LogisticRegression()
5 regLog.fit(X,y)
6 reglog.predict(X)

```

```

LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
verbose=0, warm_start=False)

```

ii. Probar con “iris.feature\_names”, “iris.target”, “iris.data”.

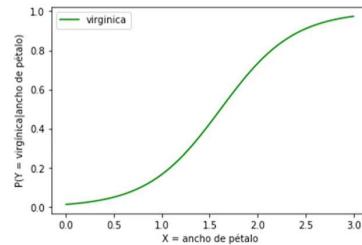
iii. Resumiendo:

1. Línea 1: información al ancho del pétalo.
2. Línea 2: Se usarán los objetos de la clase Virginica, se asignará valor 1 a los objetos que pertenecen a esta clase o 0 en caso contrario.
3. Línea 3: Se importa la clase.
4. Línea 4: Se crea un objeto de la clase.
5. Línea 5: Se ajusta la regresión.
6. Línea 6: Se realiza la predicción.
7. Se entregan los descriptores y el labels.

\* Regresión logística

\* Implementación

```
Import matplotlib.pyplot as plt
x_nuevo = np.linspace(0,3,100).reshape(-1,1)
y_proba = reglog.predict_proba(x_nuevo)
plt.plot(x_nuevo, y_proba[:,1], "g-", label="virginica")
plt.xlabel('X = ancho de pétalo')
plt.ylabel("P(Y = virginica|ancho de pétalo)")
plt.legend()
plt.show()
```



## 15. Naive Bayes

### a. Recordatorio de probabilidades:

- i. Es un clasificador probabilístico.

Probabilidad	Al no estar seguros sobre el resultado de un evento
Es un cálculo de posibilidad de que algo suceda.	Se puede hablar sobre las probabilidades de ciertos resultados.

Variable aleatoria
Variable cuyos valores posibles son resultados numéricos de un fenómeno aleatorio.

- ii. Variables aleatorias:



- iii. Ejemplo de distribución discreta es la distribución binomial y distribución Bernoulli.
- iv. Ejemplo de la distribución continua es la distribución normal y la distribución exponencial.
- v. Algunas propiedades:
  1. Asumamos que tenemos 2 eventos (A y B).
  2. Regla de la suma, establece que la probabilidad de ocurrencia del evento A o B es igual a la suma de las

probabilidades individuales menos la probabilidad de ocurrencia simultanea de los dos elementos. Si los eventos son independientes solo se considera la suma de las probabilidades individuales.

3. Regla del producto, establece que la probabilidad de ocurrencia de A y B de forma simultanea es igual al producto de la probabilidad condicional del primero dado el segundo por la probabilidad del segundo. Si los eventos son estadísticamente independientes este calculo se traduce al producto de las probabilidades individuales.

Regla de suma

$$p(A \cup B) = p(A) + p(B) - p(A \cap B)$$

Regla del producto

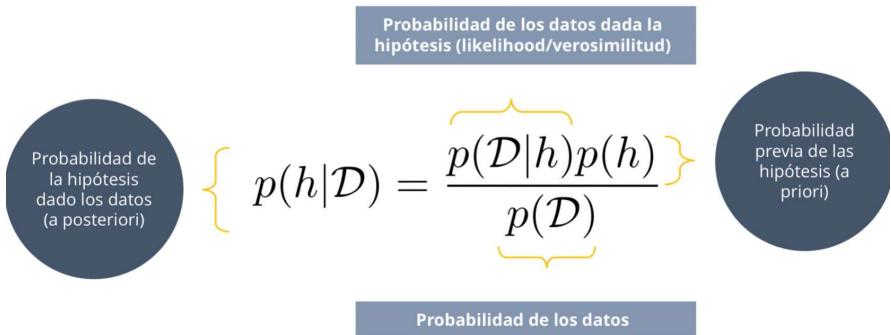
$$p(A, B) = p(A \cap B) = p(A|B)p(B)$$

4. Probabilidad marginal, indica que si tenemos una probabilidad conjunta de A, B para calcular la probabilidad de A como se presenta, se suman todos los estados posibles de B.

Probabilidad Marginal

$$p(A) = \sum_b p(A, B = b) = \sum_b p(A|B = b)p(B = b)$$

## vi. Teorema de Bayes:



1. Si consideramos un conjunto de datos ( $D$ ) y una hipótesis por validar ( $h$ ).
2. La probabilidad de la hipótesis dado los datos ( $p(h|D)$ ) se conoce como probabilidad a posteriori.
3. La probabilidad de los datos dada la hipótesis ( $p(D|h)$ ) se conoce como función de verosimilitud o likelihood.
4. La probabilidad sobre las hipótesis se conoce como probabilidad a priori.
5. La probabilidad sobre los datos se conoce como probabilidad marginal o evidencia.

## b. Clasificación Naïve Bayes:

- i. Siguiendo con el teorema, lo que buscamos es entregar la probabilidad de cierta clase dado los valores para un conjunto de descriptores.
- ii. Del mismo teorema, esto es igual a la probabilidad de los descriptores por la probabilidad a priori de esta clase, dividido por la probabilidad marginal de los descriptores.

$$p(y|x_1, \dots, x_n) = \frac{p(x_1, \dots, x_n|y)p(y)}{p(x_1, \dots, x_n)}$$

Descriptores Probabilidad priori de la clase  
Probabilidad posteriori de la clase

- iii. El segundo truco de este clasificador es asumir que los descriptores son independientes dada la clase. Esto permite simplificar la función de verosimilitud. Para estimarla solo necesitamos una multiplicación de probabilidades unidimensionales, una por cada descriptor.

$$\begin{aligned}
 & \text{Supuesto de independencia condicional} \\
 & p(x_i|y, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n) = p(x_i|y) \quad \forall i \in 1, \dots, n \\
 & p(y|x_1, \dots, x_n) = \frac{p(x_1, \dots, x_n|y)p(y)}{p(x_1, \dots, x_n)} \\
 & \downarrow \\
 & p(y|x_1, \dots, x_n) = \frac{\prod_{i=1}^n p(x_i|y)p(y)}{p(x_1, \dots, x_n)} \\
 & \downarrow \\
 & p(y|x_1, \dots, x_n) = \frac{\prod_{i=1}^n p(x_i|y)p(y)}{p(x_1, \dots, x_n)} \quad \text{Constante} \\
 & \downarrow \\
 & p(y|x_1, \dots, x_n) \propto \prod_{i=1}^n p(x_i|y)p(y) \\
 & \downarrow \\
 & \hat{y} = \arg \max_y \prod_{i=1}^n p(x_i|y)p(y) \quad \text{Clasificador}
 \end{aligned}$$

- iv. Finalmente, la clase que entregue una mayor probabilidad dado los descriptores será la clase predicha.
- v. Finalmente, hay que notar que dependiendo del tipo de descriptor que tengamos en nuestra base de datos se tendrán que realizar distintos supuestos respecto al tipo de distribución utilizada. Por ejemplo, las más utilizadas:

$$\hat{y} = \arg \max_y \prod_{i=1}^n p(x_i|y)p(y)$$

#### Supuesto respecto al tipo de distribución de los descriptores

- Gaussiana
- Bernoulli
- Categórica
- Mixta

- vi. Es rápido, pero mal estimador de probabilidad, por lo tanto, las probabilidades entregadas no deben ser tomadas en serio.

### c. Implementación:

```
# Porcentaje de aciertos
import numpy as np
import pandas as pd
from sklearn.naive_bayes import CategoricalNB # MultinomialNB, GaussianNB, BernoulliNB
from sklearn.preprocessing import LabelEncoder # Para realizar transformación sobre la base de datos
from sklearn.model_selection import train_test_split # Para separar el dataset, para entrenar y evaluar
from sklearn.metrics import accuracy_score # Para performance del clasificador

data = pd.read_csv("tennis.csv")
data
```

	outlook	temp	humidity	windy	play
0	sunny	hot	high	False	no
1	sunny	hot	high	True	no
2	overcast	hot	high	False	yes
3	rainy	mild	high	False	yes

```
# Transformar a número el dataset
numeric = LabelEncoder()
for c in data.columns:
    data[c] = numeric.fit_transform(data[c])
```

```
# El clasificador asumira que son datos que vienen de una distribución categorica
data
```

	outlook	temp	humidity	windy	play
0	2	1	0	0	0
1	2	1	0	1	0
2	0	1	0	0	1

```
# Guardamos la columna play y luego la eliminamos
label = data['play']
del data['play']
```

```
data
```

	outlook	temp	humidity	windy
0	2	1	0	0
1	2	1	0	1
2	0	1	0	0
3	1	2	0	0

```
# Separamos los datos para test (20%) y para entrenar (54%)
x_train, x_text, y_train, y_test = train_test_split(data, label, test_size=0.2, random_state=54)
```

```
# Se crea un objeto y se realiza el ajuste
model = CategoricalNB()
model.fit(x_train, y_train)
```

```
+ CategoricalNB
CategoricalNB()
```

```
# Se hace una predicción y se mide la performance (accuracy)
pred = model.predict(x_text)
accuracy = accuracy_score(y_test, pred)
print(accuracy)
```

```
1.0
```

## 16. Evaluación de clasificadores

### a. Introducción:

- i. Al entrenar un modelo es lograr que no solo se obtenga un buen resultado en el entrenamiento, sino también un buen resultado en un conjunto independiente de datos, es decir, que no se sobreajuste.

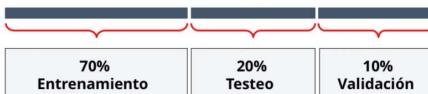
### b. Estrategia de entrenamiento:

- i. Un **clasificador** debe ser evaluado en un set de datos distinto al de entrenamiento, de otra forma el clasificador aprenderá de memoria los datos.
- ii. Mientras más grande es el set de validación mejor será la estimación del rendimiento del modelo, pero menos los datos de entrenamiento.
- iii. Hay algoritmos que necesitan monitorear métricas durante el entrenamiento, normalmente se usa una fracción pequeña, cerca del 10% del total.
- iv. Al finalizar el entrenamiento se realiza el entrenamiento con un set de datos que generalmente es el 20% del total.

Un **clasificador** debe ser evaluado en un set de datos distinto al de entrenamiento.



Realizar muestreo **estratificado**, asegurando la proporción de clases es crucial y siempre debe hacerse.



### v. La validación cruzada:

1. Se usa para validar el modelo sin depender de la muestra aleatoria echada para separar los distintos set.
2. El set de entrenamiento se divide en un número fijo, por ejemplo 5, y se entrena con 4 y se testea con 1.
3. El proceso se repite hasta que se ha testeado con todas las partes.
4. Al final se tendrá una predicción para todo el set de datos, de ahí se podrá calcular las distintas métricas de evaluación.
5. Mientras más divisiones se haga, mejor será la validación del modelo, pero mayor será el tiempo total de entrenamiento.



### c. Matriz de confusión:

- i. Es una forma gráfica de ver el resultado de un clasificador.
- ii. En el eje vertical esta la etiqueta original y en el horizontal la predicha.
- iii. El caso perfecto es cuando la matriz es diagonal.

Muestra de forma gráfica el resultado del clasificador por clase.

En el **caso normalizado** las filas suman 1 y el **número de ejemplos** el **caso contrario**.

		Predicción		
		A	B	C
Etiqueta	A	0.8	0.15	0.05
	B	0.06	0.89	0.05
	C	0.21	0.03	0.76

### d. Métrica de evaluación:

- i. La **precisión** es una métrica que mide los falsos positivos y debe ser estimada por cada clase. Se calcula como la cantidad de elementos correctamente predichos divididos por el total de elementos predichos para esa clase. En el mejor de los casos es 1.

Medida de falsos negativos		
Implica qué fracción del total de elementos el clasificador puede rescatar.		

$$\frac{80}{80 + 15 + 5} = 80\%$$

Predicción					
Etiqueta	A	B	C		
	A	80	15	5	
	B	6	89	5	
			21	3	76

- ii. En el caso en el que la precisión es 1, pero clasificando una porción muy pequeña del total. El **Recall** ayuda a resolver este problema, detectando los falsos negativos, en esencia mide la fracción de elementos que el clasificador puede rescatar. Al igual que la precisión debe medirse por clase. Se calcula como la división entre los elementos clasificados correctamente y el total de los clasificados para esa clase, el caso ideal es 1 donde todos los elementos son clasificados correctamente. El recall no es la única métrica a analizar, se puede tener recall 1 cuando el modelo asigna a todos los elementos la misma clase, común cuando se trabajan con clases desbalanceadas.

Medida de falsos negativos		
Implica qué fracción del total de elementos el clasificador puede rescatar.		

$$\frac{80}{80 + 15 + 5} = 80\%$$

Predicción					
Etiqueta	A	B	C		
	A	80	15	5	
	B	6	89	5	
			21	3	76

- iii. El **F-score**, es una medida que mezcla **precisión** y **Recall** en un solo valor. El mejor valor que puede tomar es 1 cuando la precisión y el recall es máximo. Si bien no entrega exactamente la misma información que ambos valores por separado, es una buena forma de evaluar el clasificador usando una sola métrica.

Medida que
Integra precisión y <i>recall</i> en un solo valor.

$$\frac{2}{\frac{1}{Recall} + \frac{1}{Precision}}$$

		Predicción		
		A	B	C
Etiqueta	A	80	15	5
	B	6	89	5
	C	21	3	76

- iv. También, se pueden tener medidas generales del rendimiento de un clasificador, ahora bien, su interpretación depende del problema específico ya que el desbalance de clases puede sesgar los resultados. El **Accuracy** es la cantidad de elementos clasificados correctamente divididos por el total de elementos, ideal es el valor de 1.

### accuracy

Medida rendimiento a nivel general del clasificador
No toma en cuenta desbalance de las clases.

		Predicción		
		A	B	C
Etiqueta	A	80	15	5
	B	6	89	5
	C	21	3	76

$$\frac{80 + 89 + 76}{80 + 15 + 5 + 6 + 89 + 5 + 21 + 3 + 76} = 81,66\%$$

## 17. Árboles de decisión:

### a. Entropía y ganancia de información:

#### i. ¿Qué atributo seleccionar?



1

Cada selección de atributo generará bases de datos diferentes para cada ramificación.

2

Para tener mayor certeza de la clasificación en los nodos hoja no interesa tener grupos homogéneos en cada uno de ellos.



Opción 1

Separar por atributo binario A retorna

- a) Subconjunto con 50 objetos de clase 1 y 10 objetos de clase 0.
- b) Subconjunto con 50 objetos de clase 0 y 10 objetos de clase 1.

Opción 2

Separar por atributo binario B retorna

- a) Subconjunto con 35 objetos de clase 1 y 25 objetos de clase 0.
- b) Subconjunto con 35 objetos de clase 0 y 25 objetos de clase 1.

¿Cuál opción es mejor?, la opción 1 ya que las BD resultantes son más homogéneas, eso representa una mejor separación entre las clases.

ii. Un concepto fundamental para determinar las ramificaciones es la **entropía**, para entender esto, consideremos que un indicador de información de un evento se puede calcular como  $-\log_2 p_c$  (menos logaritmo de la probabilidad de dicho evento). Es decir, eventos con baja probabilidad serán altamente informativos y eventos con alta probabilidad serán bajamente informativos. Luego la entropía se puede entender como la esperanza de información en una base

de datos. Al final lo que se busca que es nuestras bases de datos generen baja entropía.

$H(\mathcal{D}) = - \sum_{c \in C} p_c \log_2 p_c$

Esperanza de información en base de datos

Información =  $-\log_2 p_c$

$H(\mathcal{D}) = -0 \log_2 0 - 1 \log_2 1 = 0$

$$H(\mathcal{D}) = - \sum_{c \in C} p_c \log_2 p_c$$
$$H(\mathcal{D}) = -0.5 \log_2 0.5 - 0.5 \log_2 0.5 = 0.301$$

- iii. Un criterio específico para seleccionar los atributos que generan mejores ramificaciones (**Ganancia de información**), definida como  $G(D, A)$ , donde  $A$  es el atributo evaluado y  $D$  como la base de datos. Entonces:

1.  $H(D)$ : entropía de la base de datos previa la ramificación.
  2.  $a$ : Valores que toma el atributo.
  3.  $|D_a|/|D|$ : Proporción de objetos en cada ramificación.
  4.  $H(D_a)$ : entropía de la ramificación dada por el valor  $a$ .

**Entropía Original**

**Entropía promedio de división**

$$G(\mathcal{D}, A) = H(\mathcal{D}) - \sum_{a \in \text{values}(A)} \frac{|\mathcal{D}_a|}{|\mathcal{D}|} H(\mathcal{D}_a)$$

**Ganancia de información por utilizar atributo A**

## b. Implementación:

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import plot_confusion_matrix
import matplotlib.pyplot as plt

iris = load_iris()

# Se cargan los descriptores y la features
iris_data = pd.DataFrame(iris.data, columns=iris.feature_names)

# Se carga el target, el level, la clase
target = pd.DataFrame(iris.target, columns=['target'])

iris.target_names
array(['setosa', 'versicolor', 'virginica'], dtype='|<U10')

# Al conocer iris cargamos directamente el conjunto de testeo entrenamiento
X, y = load_iris(return_X_y=True)
# Dividimos los elementos en test y entrenamiento. Se pide, X: base de datos de descriptores completo,
# y: los labels, test_size: tamaño de el test y random_state: perfiles de replicaridad en 0 para que funcione siempre igual
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4, random_state=0)
```

```
1 # profundidad del arbol y criterio a utilizar
2 clf = DecisionTreeClassifier(max_depth=4, criterion='entropy')
3 # Ajuste de entrenamiento
4 clf.fit(X_train, y_train)
5 # Predicción
6 target_pred = clf.predict(X_test)
7 # Predicciones en el conjunto de test
8 print('Predicción: ', target_pred)
9 # Valor real en el conjunto de test
10 print('Real: ', y_test)
```

```
Predicción: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
2 1 1 2 0 2 0 0 1 2 2 1 2 1 2 1 1 2 2 1 2 1 2]
Real: [2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0 2 2 1 0
1 1 1 2 0 2 0 0 1 2 2 2 1 2 1 1 2 2 2 2 1 2]
```

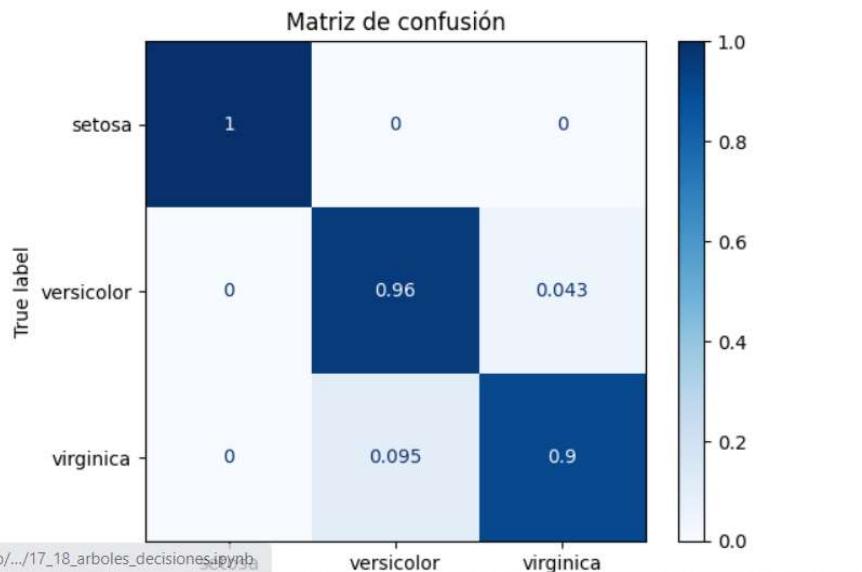
```

disp = plot_confusion_matrix(clf, X_test, y_test, display_labels=iris.target_names,
                             cmap=plt.cm.Blues, normalize='true')
disp.ax_.set_title('Matriz de confusión')

C:\Users\Usuario\anaconda3\envs\CURSO_IV\lib\site-packages\sklearn\utils\deprecation
warning: ConfusionMatrixDisplay is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0 and will be removed in 1.1. Please use its methods: ConfusionMatrixDisplay.from_predictions or ConfusionMatrixDisplay.from_es
warnings.warn(msg, category=FutureWarning)

Text(0.5, 1.0, 'Matriz de confusión')

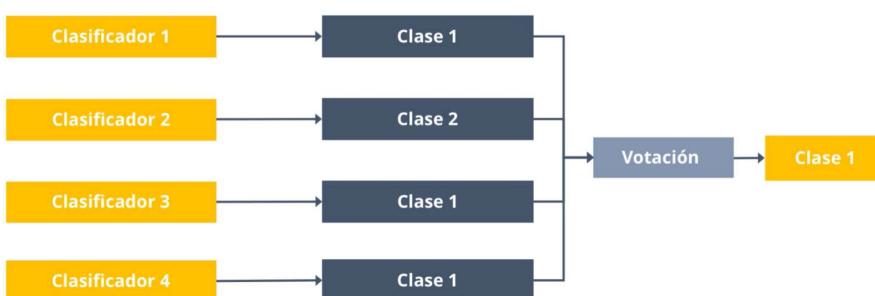
```



## 18. Random Forest

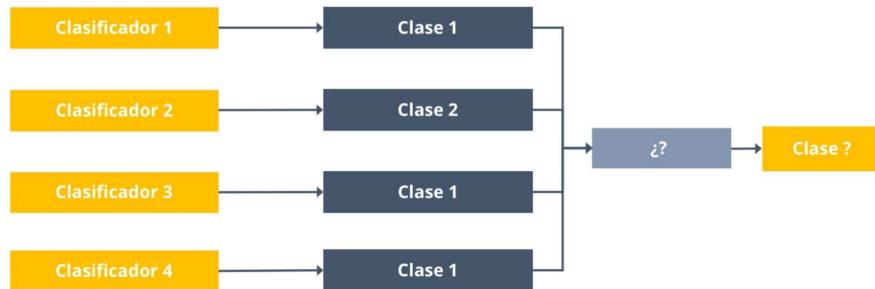
### a. Votación de clasificadores:

- i. ¿Qué pasa si se quiere unir varias predicciones de varios clasificadores?, ¿Qué ocurre si la clase entregada por cada clasificador es diferente?, una forma es evaluar cada clase como voto, el clasificador votara una clase, luego la clase más votada será la predicción.



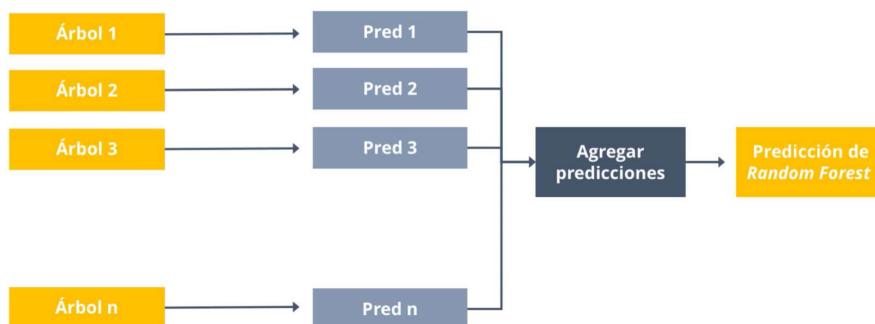
**ii. Otras ideas para votar:**

1. El promedio.
2. Otorgar mayor credibilidad a algunos clasificadores.
3. Aprender el nivel de credibilidad a los clasificadores.



**b. Random Forest:**

- i. Es uno de los métodos más potentes de aprendizaje de máquina.
- ii. Algoritmo que trata con múltiples clasificadores.
- iii. Todos los clasificadores son basados en árboles de decisiones.
- iv. Se basa en la generación de múltiples árboles, esto a través de un muestreo de los descriptores de la base de datos durante la selección de cada ramificación y también muestreando los pasos que se incluirán para entrenar ese árbol. De esta forma se puede obtener una gran cantidad de predicciones y luego seleccionar la clase más votada.



**c. Implementación en Python:**

```

import matplotlib.pyplot as plt

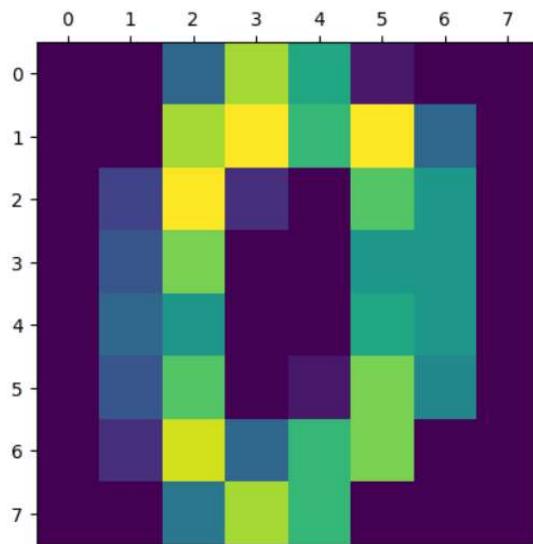
from sklearn.datasets import load_digits
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

# Para separar los conjuntos de test y entrenamiento
from sklearn.model_selection import train_test_split
# Graficar la matriz de confusión
from sklearn.metrics import plot_confusion_matrix

digits = load_digits()
digits.images[0]

array([[ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.],
       [ 0.,  0., 13., 15., 10., 15.,  5.,  0.],
       [ 0.,  3., 15.,  2.,  0., 11.,  8.,  0.],
       [ 0.,  4., 12.,  0.,  0.,  8.,  8.,  0.],
       [ 0.,  5.,  8.,  0.,  0.,  9.,  8.,  0.],
       [ 0.,  4., 11.,  0.,  1., 12.,  7.,  0.],
       [ 0.,  2., 14.,  5., 10., 12.,  0.,  0.],
       [ 0.,  0.,  6., 13., 10.,  0.,  0.,  0.]])
```

# Transforma la matriz a imagen  
plt.matshow(digits.images[0])  
plt.show()



```
# Cargamos directamente los descriptores y labels de cada objeto
X, y = load_digits(return_X_y=True)
```

```
X[0]
```

```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
      12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
      0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
     10., 12.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

```
# Ahora viene como vector de 64 valores
```

```
X[0].shape
```

```
(64,)
```

```
# El labels de la imagen 0
```

```
y[0]
```

```
0
```

```
# 1797 imagenes y cada imagen tiene un vector descriptor de 64
```

```
X.shape
```

```
(1797, 64)
```

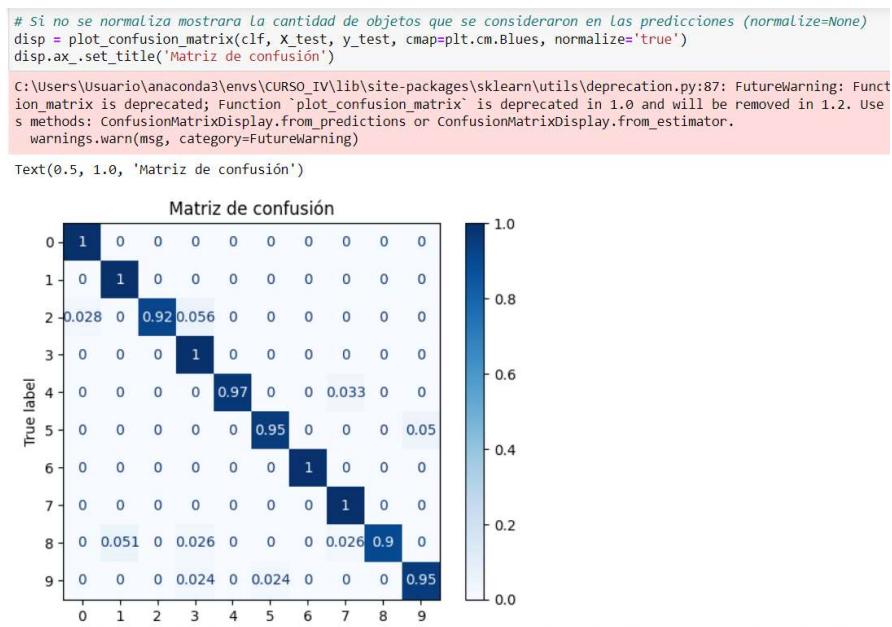
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
```

```
* RandomForestClassifier
RandomForestClassifier()
```

```
target_pred = clf.predict(X_test)
print("Predicción: ", target_pred)
print('Real: ', y_test)
```

```
Predicción: [2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 9 1 6 3 0 2 3 4 1 9 2 6 9 1 8 3 5 1
2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
4 8 9 7 9 8 0 6 5 2 5 3 4 1 7 0 6 1 5 5 3 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
5 9 9 1 5 3 6 1 8 9 7 7 6 7 6 5 6 0 8 8 9 8 6 1 0 4 1 6 3 8 6 7 4 9 6 3 0
3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
9 3 7 6 2 3 3 1 6 9 3 6 3 3 2 0 7 6 1 1 9 7 2 7 1 5 5 7 5 2 3 7 2 7 5 5 7
0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 3 6 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 9 8 5 7 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 1 9
5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8]
Real: [2 8 2 6 6 7 1 9 8 5 2 8 6 6 6 1 0 5 8 8 7 8 4 7 5 4 9 2 9 4 7 6 8 9 4 3
1 0 1 8 6 7 7 1 0 7 6 2 1 9 6 7 9 0 0 5 1 6 3 0 2 3 4 1 9 2 6 9 1 8 3 5 1
2 8 2 2 9 7 2 3 6 0 5 3 7 5 1 2 9 9 3 1 7 7 4 8 5 8 5 5 2 5 9 0 7 1 4 7 3
4 8 9 7 9 8 2 6 5 2 5 3 8 4 8 7 0 6 1 5 9 9 9 5 9 9 5 7 5 6 2 8 6 9 6 1 5 1
5 9 9 1 5 3 6 1 8 9 8 7 6 7 6 5 6 0 8 8 9 8 6 1 0 4 1 6 3 8 6 7 4 5 6 3 0
3 3 3 0 7 7 5 7 8 0 7 8 9 6 4 5 0 1 4 6 4 3 3 0 9 5 9 2 1 4 2 1 6 8 9 2 4
9 3 7 6 2 3 3 1 6 9 3 6 3 2 2 0 7 6 1 1 9 7 2 7 8 5 5 7 5 2 3 7 2 7 5 5 7
0 9 1 6 5 9 7 4 3 8 0 3 6 4 6 3 2 6 8 8 8 4 6 7 5 2 4 5 3 2 4 6 9 4 5 4 3
4 6 2 9 0 1 7 2 0 9 6 0 4 2 0 7 9 8 5 4 8 2 8 4 3 7 2 6 9 1 5 1 0 8 2 1 9
5 6 8 2 7 2 1 5 1 6 4 5 0 9 4 1 1 7 0 8 9 0 5 4 3 8 8]
```



## 19. Random Forest para regresión:

- La variable de interés ya no es una categoría es una variable numérica. Entonces se usa el promedio de los datos de cada hoja.
- ¿Cómo seleccionamos el atributo en cada ramificación?, ¿Cómo seleccionamos el corte adecuado en cada ramificación?, usamos el error cuadrático medio.
- Nos quedamos con el que muestre menor error cuadrático medio.
- Para agregar las respuestas de los distintos árboles, estas se promedian.

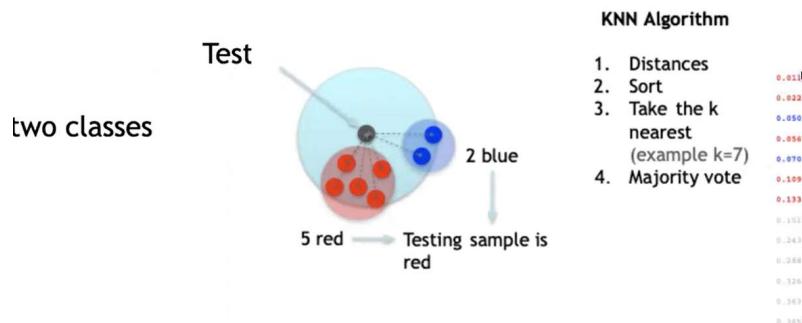
## ayudantía 2:

### 1. KNN (k-nearest neighbors)

- a. Aprendizaje supervisado.
- b. Mide la distancia Euclidiana.

$$d(x_i, x_j) = \sqrt{\sum_{r=1}^p (x_{ri} - x_{rj})^2}$$

- c. Esto es la sumatoria de la diferencia de 2 puntos al cuadrado.



- d. La cantidad de vecino óptimo es impar ( $k = 1, 3, 5, 7, 9$ ).

- e. Ejemplo:

Importamos librerías necesarias

```
import matplotlib.pyplot as plt

import pandas as pd
import numpy as np

from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn import tree

from sklearn.model_selection import train_test_split
from sklearn.metrics import plot_confusion_matrix

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import accuracy_score
```

Lectura y Procesamiento de datos

```
df = pd.read_csv('gender.csv')

df
```

	Favorite Color	Favorite Music Genre	Favorite Beverage	Favorite Soft Drink	Gender
0	Cool	Rock	Vodka	7UP/Sprite	F
1	Neutral	Hip hop	Vodka	Coca Cola/Pepsi	F
2	Warm	Rock	Wine	Coca Cola/Pepsi	F
3	Warm	Folk/Traditional	Whiskey	Fanta	F
4	Cool	Rock	Vodka	Coca Cola/Pepsi	F
...	...	...	...	...	...
61	Cool	Rock	Vodka	Coca Cola/Pepsi	M
62	Cool	Hip hop	Beer	Coca Cola/Pepsi	M
63	Neutral	Hip hop	Doesn't drink	Fanta	M
64	Cool	Rock	Wine	Coca Cola/Pepsi	M
65	Cool	Electronic	Beer	Coca Cola/Pepsi	M

Verificamos si hay valores nulos, si hay, tratamos los datos:

```
#df.loc[df.Gender=='F'].drop(0)
```

```
df.isna().sum()
```

Favorite Color	0
Favorite Music Genre	0
Favorite Beverage	0
Favorite Soft Drink	0
Gender	0
dtype: int64	

```
df.describe()
```

	Favorite Color	Favorite Music Genre	Favorite Beverage	Favorite Soft Drink	Gender
count	66	66	66	66	66
unique	3	7	6	4	2
top	Cool	Rock	Doesn't drink	Coca Cola/Pepsi	M
freq	37	19	14	32	33

Si existe una clase que tenga muy pocas muestras, a veces es mejor eliminar para obtener mejores resultados. Para ubicar y eliminar usando el drop con el índice, si se desea se puede usar:

```
df.loc[df.Gender=='F']
```

```
#.drop(0)
```

```
[15]: df = df.loc[df.Gender=='F'].drop([2,3])
df
```

	Favorite Color	Favorite Music Genre	Favorite Beverage	Favorite Soft Drink	Gender
0	Cool	Rock	Vodka	7UP/Sprite	F
1	Neutral	Hip hop	Vodka	Coca Cola/Pepsi	F
4	Cool	Rock	Vodka	Coca Cola/Pepsi	F
5	Warm	Jazz/Blues	Doesn't drink	Fanta	F
6	Cool	Pop	Beer	Coca Cola/Pepsi	F

Entre otras cosas que se pueden hacer:

```
## Visualizaciones...., normalización, transformación de variables categóricas a numéricas, imputación, etc. etc.
```

```
df.columns
```

```
Index(['Favorite Color', 'Favorite Music Genre', 'Favorite Beverage',
       'Favorite Soft Drink', 'Gender'],
      dtype='object')
```

```
target = 'Gender'
df[target].unique()
```

```
array(['F', 'M'], dtype=object)
```

```
df['Favorite Color'].value_counts()
```

```
Cool      37
Warm     22
Neutral    7
Name: Favorite Color, dtype: int64
```

```
df['Favorite Music Genre'].value_counts()
```

```
Rock        19
Pop         17
Hip hop      8
Electronic    8
R&B and soul   6
Folk/Traditional  4
Jazz/Blues     4
Name: Favorite Music Genre, dtype: int64
```

Nos damos cuenta que hay variables categóricas y hay que pasarlas a numéricas.

En este caso todas son categóricas, sino hay que hacer una por una.

---

```

le = LabelEncoder()
df = df.apply(le.fit_transform)
df

```

	Favorite Color	Favorite Music Genre	Favorite Beverage	Favorite Soft Drink	Gender
0	0	6	3	0	0
1	1	2	3	1	0
2	2	6	5	1	0
3	2	1	4	2	0
4	0	6	3	1	0
...	...	...	...	...	...
61	0	6	3	1	1
62	0	2	0	1	1
63	1	2	1	2	1
64	0	6	5	1	1
65	0	0	0	1	1

---

Ahora sepáramos el set a training y testing:

#### Separamos nuestro set a training y testing

```

y = df[target]
X = df.drop('Gender', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
print(X_train.shape, X_test.shape)
(52, 4) (14, 4)

```

Y ahora entremos el modelo:

#### Definición del modelo KNN

```

# definimos modelo KNN
clf = KNeighborsClassifier(n_neighbors = 5)

#entrenamiento
clf.fit(X_train, y_train)

KNeighborsClassifier()

```

```

y_pred = clf.predict(X_test)
y_pred

array([1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1])

disp = plot_confusion_matrix(clf, X_test, y_test,
                             cmap=plt.cm.Blues,
                             normalize='true')
disp.ax_.set_title('Matriz de confusión')

/Users/helem/venv/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function 'n_matrix' is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods: ConfusionMatrix.
  warnings.warn(msg, category=FutureWarning)
Text(0.5, 1.0, 'Matriz de confusión')


    Matriz de confusión
    <matplotlib>

```

Predicted label \ True label	0	1
0	0.95	0.05
1	0.12	0.88

## Definición del modelo Random Forest

```

clf = RandomForestClassifier(n_estimators=3, max_depth=3,
                            random_state=0)

clf.fit(X_train, y_train)

RandomForestClassifier(max_depth=3, n_estimators=3, random_state=0)

feature_names = X.columns
feature_names

Index(['Favorite Color', 'Favorite Music Genre', 'Favorite Beverage',
       'Favorite Soft Drink'],
      dtype='object')

```

Antes de evaluarlo, se va a graficar, se usa la función tree que permite graficar los arboles:

```

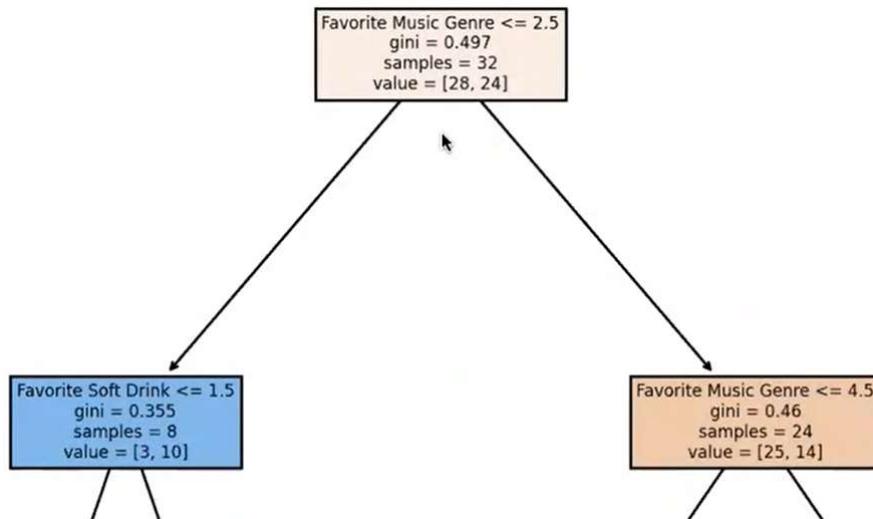
feature_names = X.columns
feature_names

fn = feature_names

fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(10,10), dpi=200)
tree.plot_tree(clf.estimators_[2], feature_names = fn, filled=True )

[Text(0.4583333333333333, 0.875, 'Favorite Music Genre <= 2.5\ngini = 0.497\nsamples = 32\nvalue = [28, 24]'),
 Text(0.25, 0.625, 'Favorite Soft Drink <= 1.5\ngini = 0.355\nsamples = 8\nvalue = [3, 10]'),
 Text(0.1666666666666666, 0.375, 'Favorite Music Genre <= 1.0\ngini = 0.48\nsamples = 4\nvalue = [3, 2]'),
 Text(0.0833333333333333, 0.125, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.25, 0.125, 'gini = 0.5\nsamples = 3\nvalue = [2, 2]'),
 Text(0.3333333333333333, 0.375, 'gini = 0.0\nsamples = 4\nvalue = [0, 8]'),
 Text(0.6666666666666666, 0.625, 'Favorite Music Genre <= 4.5\ngini = 0.46\nsamples = 24\nvalue = [25, 14]'),
 Text(0.5, 0.375, 'Favorite Color <= 1.0\ngini = 0.266\nsamples = 10\nvalue = [16, 3]'),
 Text(0.4166666666666667, 0.125, 'gini = 0.32\nsamples = 8\nvalue = [12, 3]'),
 Text(0.5833333333333334, 0.125, 'gini = 0.0\nsamples = 2\nvalue = [4, 0]'),
 Text(0.8333333333333334, 0.375, 'Favorite Music Genre <= 5.5\ngini = 0.495\nsamples = 14\nvalue = [9, 11]'),
 Text(0.75, 0.125, 'gini = 0.32\nsamples = 3\nvalue = [1, 4]'),
 Text(0.9166666666666666, 0.125, 'gini = 0.498\nsamples = 11\nvalue = [8, 7]')]

```



Luego evaluamos:

```

y_test.values
array([0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1])

target_pred = clf.predict(X_test)
print('Predicción: ', target_pred)
print('Real: ', y_test.values)

Predicción:  [1 0 0 0 0 0 1 0 1 0 1 1 1]
Real:          [0 0 1 0 1 0 1 1 1 0 1 0 1]

disp = plot_confusion_matrix(clf, X_test, y_test,
                             cmap=plt.cm.Blues,
                             normalize='true')
disp.ax_.set_title('Matriz de confusión')

/Users/helem/venv/lib/python3.7/site-packages/sklearn/utils/deprecation.py:87: FutureWarning:
n_matrix` is deprecated in 1.0 and will be removed in 1.2. Use one of the class methods:
iminator.
warnings.warn(msg, category=FutureWarning)
Text(0.5, 1.0, 'Matriz de confusión')

Matriz de confusión
  
```

	0	1
0	0.5	0.5
1	0.5	0.5

```
accuracy_score(y_test, target_pred)
```

0.5

## Otras del Mini proyecto:

Regresión línea cerrada:

1. Random entre 0 y 3 (dependiente):

```
x = np.random.rand(100, 1)*3
x

array([[1.46460672],
       [2.48236893],
       [2.41927683],
       [2.16730835],
       [2.26823898],
       [1.96048338],
       [2.66382346],
       [1.30070421],
       [2.24119277],
       [0.329402031...])
np.random.uniform(0, 3, 100)
```

2. Random 100 datos (y dependiente), también hay que redimensionar porque deben estar en la misma dimensión (-1 para que mantenga el valor por defecto que en este caso es 100):

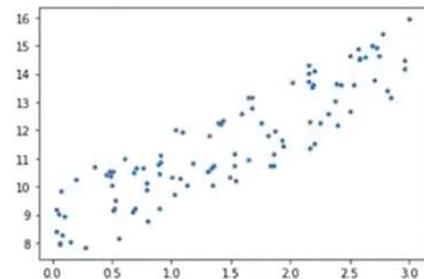
```
x = np.random.rand(100, 1)
x.shape
(100, 1)
np.random.uniform(2, 3, 100).shape
(100,)
x.shape
()

Con la misma librería, genere los 100 datos de la variable dependiente de la siguiente forma  $y = 5 + 2x + u(2, 3)$ .
u = np.random.uniform(2, 3, 100)
u = u.reshape(-1, 1)
u.shape
```

Luego generamos la variable dependiente (el  $u$  es una variable que le entrega dispersión, agregando una desviación estándar mayor):

```
#variable dependiente
y = 5 + 2*x + u

plt.scatter(x,y, s=10)
<matplotlib.collections.PathCollection at 0x7fe6956f8750>
```



$$\bar{y} = \bar{\Theta}_0 + \bar{\Theta}_1 x_1 + \dots + \bar{\Theta}_n x_n$$

y = predicción

n = descriptores

$x_i$  = valor del descriptor

## Regresión Polinomial.

Regresión con penalización:

```
dataset = pd.read_csv('ts.csv').sample(200)

y = dataset['magnitud']
x = dataset['tiempo']

print(x.shape)
x=x.values.reshape(-1,1)
x.shape

(200,)
(200, 1)
```

```
x
array([[ 1.52837183],
       [-0.98023578],
       [ 0.96390969],
       [-0.55116589],
       [-1.25085467],
       [ 2.17450466],
       [ 0.11260993],
       [ 0.84022422],
       [ 0.73578013],
       [ 1.20666516]]]

data_poly #  $x^0, x^1, x^2$ 
array([[ 1.0000000e+00,  1.52837183e+00,  2.33592044e+00],
       [ 1.0000000e+00, -9.80235778e-01,  9.60862180e-01],
       [ 1.0000000e+00,  9.63909692e-01,  9.29121894e-01],
       [ 1.0000000e+00, -5.51165886e-01,  3.03783833e-01],
       [ 1.0000000e+00, -1.25085467e+00,  1.56463740e+00],
       [ 1.0000000e+00,  2.17450466e+00,  4.72847052e+00],
       [ 1.0000000e+00,  1.12609926e-01,  1.26809955e-02],
       [ 1.0000000e+00,  8.40224219e-01,  7.05976739e-01],
       [ 1.0000000e+00,  7.35780129e-01,  5.41372399e-01],
       [ 1.0000000e+00,  1.20666516e+00,  1.45604082e+00]]]

#alpha = [1, 0.0001, 0.001]
modelo = Ridge(alpha = 100)
modelo

Ridge(alpha=100)

modelo.fit(data_poly, y)

Ridge(alpha=100)

y_pred = modelo.predict(data_poly)

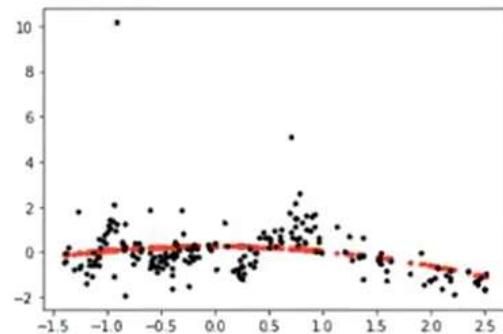
from sklearn.metrics import mean_squared_error

ECMp = np.around(mean_squared_error(y, y_pred), decimals = 5)
print("ECM = "+str(ECMp))

ECM = 1.24146
```

```
plt.plot(x, y_pred, '.', color='red')
plt.plot(x,y, '.', color='black')
```

```
[<matplotlib.lines.Line2D at 0x7fe6955728d0>]
```



Para hacer varios:

```
alpha = [1, 0.0001, 0.001]
grados = [2,3,4,5,6,7,8]

for i in grados:
    for j in alpha:
        polinomio = PolynomialFeatures(degree = 7)
        .....
        predict...

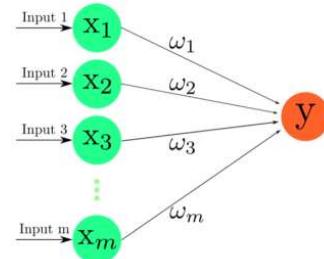
        modelo = Ridge(alpha=j, 0.0001)
        modelo
```

## **20. Introducción a las redes neuronales artificiales:**

## El perceptrón

#### Mapea de forma no lineal a un valor de entrada

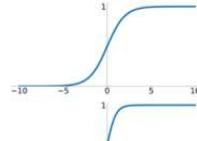
$$y = f(W \cdot \mathbf{x} + b)$$



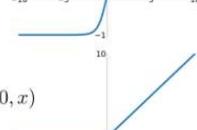
## Función de activación

## "Función de activación"

$$\text{Sigmoid}$$



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



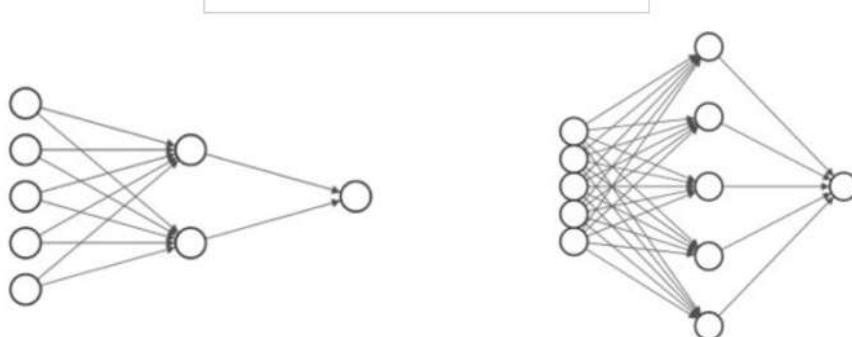
ReLU

$$\text{ReLU}(x) = \max(0, x)$$

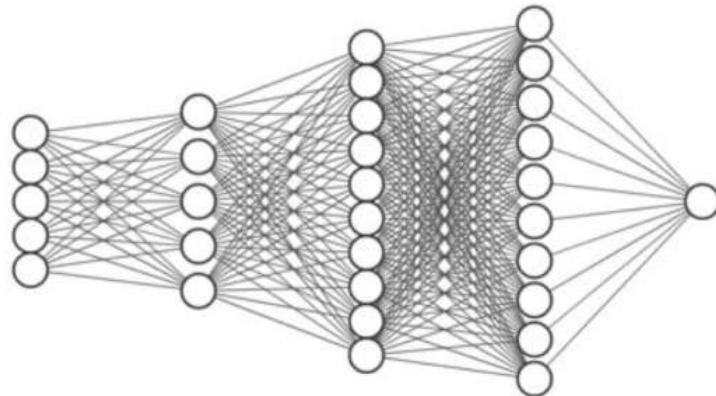
- a. **Red neuronal densa**, consiste en múltiples perceptores, de esta forma el valor de salida no es un único valor, si no un vector. En la imagen de abajo, las capas iniciales y finales se conocen como capas de entradas y salidas, las capas del medio se conocen como capas ocultas.

## Más perceptrones

#### Aumento de su expresividad

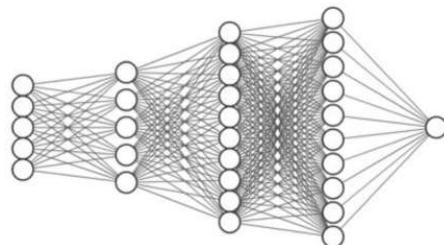


- b. Para aumentar la expresividad del modelo se pueden agregar más preceptores a las capas o agregar múltiples capas.



- c. Las redes con más capas aprenden mejor que las de una sola. De aquí viene el concepto de Depth Learning o aprendizaje profundo, son modelos de muchas capas ocultas concatenada.
- d. Las redes neuronales densa permite realizar muchas tareas, como clasificación o regresión.

Salida del modelo
Usada para clasificación o regresión
Entrenamiento
Algoritmo de "Backpropagation"



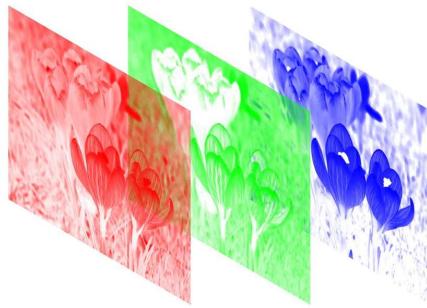
- e. La cantidad de neuronas en la capa de salida dependerá de la tarea, si es una regresión solo necesitamos un valor en le caso unidimensional. Con clasificación necesitamos tantas neuronas como clases tengamos.
- f. Cada tarea entrega una función objetivo a minimizar. Esta función reflejara el rendimiento del modelo.
- g. El algoritmo de entrenamiento es el **Backpropagation**. Consiste en ajustar los pesos de la red, en base a la función de cambio de esta de esta función objetivo.

## 21. Redes neuronales convolucionales

### a. Convoluciones, kernels y filtros:

- i. Cada elemento de la matriz indica la intensidad del color en dicha posición.

### Imágenes como matrices



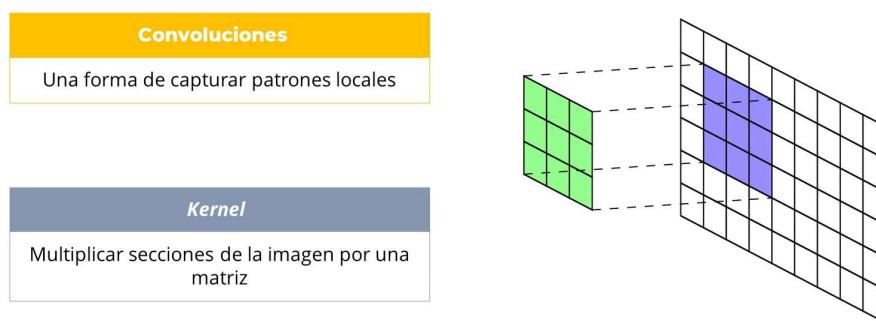
- ii. Retorna un número.

### Convoluciones



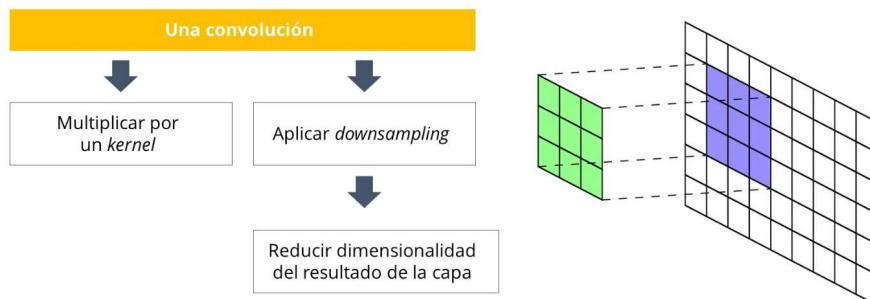
- iii. Los parámetros de la convolución son aprendidos de los datos y no fijados inicialmente.

- iv. Kernel convolucional:

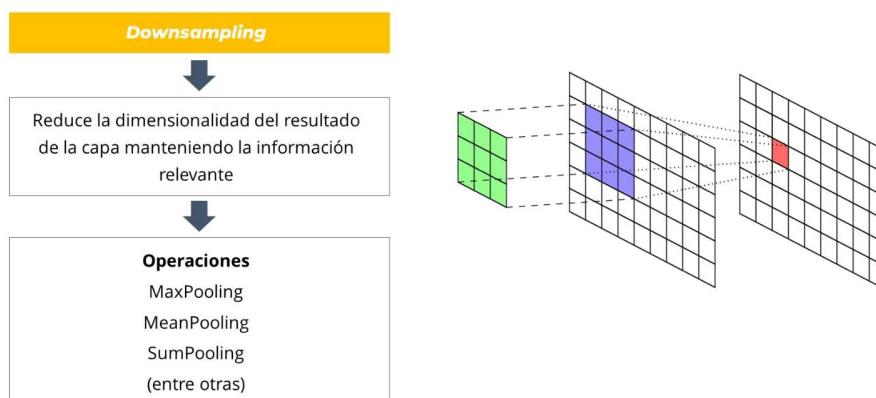


v. La convolución consiste en dos etapas (multiplicación y downsampling).

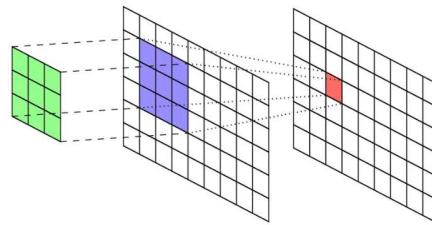
### Kernel convolucional



vi. Downsampling:

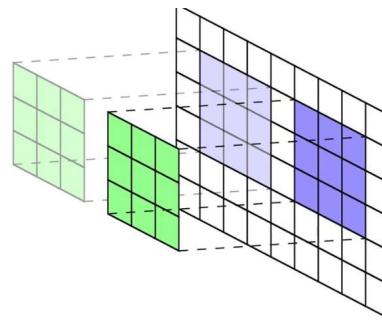
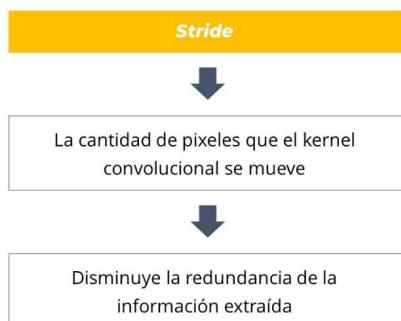


vii. Una vez echo el downsampling se aplica la función **no lineal**.

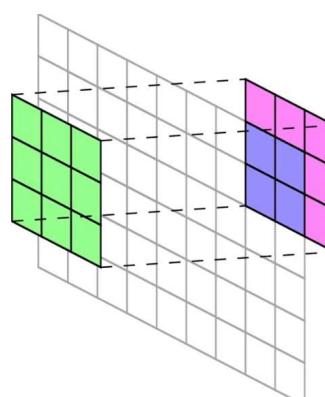


$$ReLU(x) = \text{Max}(0, x)$$

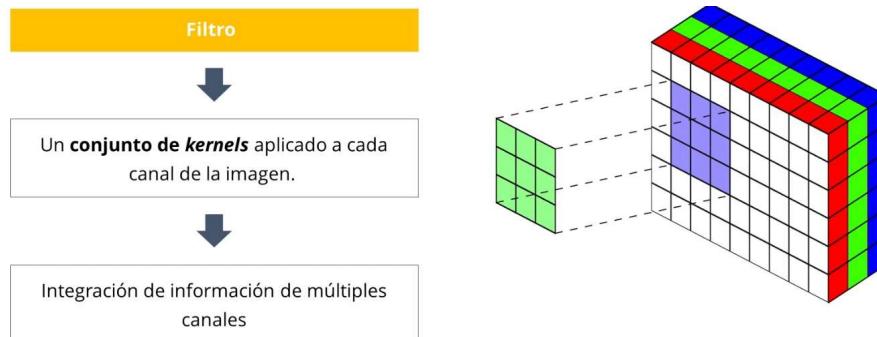
- viii. La convolución se debe mover a través de la imagen para extraer la información en todas las regiones de ella. Para esto se define **Stride** o paso.



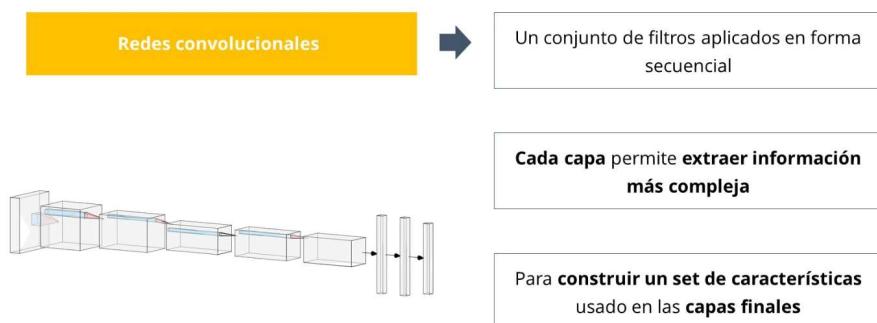
- ix. En general, dado el tamaño del kernel y el stride habrá problemas en los bordes de la imagen dado que la operación no está definida para datos faltantes. Para resolver esto se introdujo el **padding** que permite llenar los espacios faltantes con una constante.



## x. Filtros:



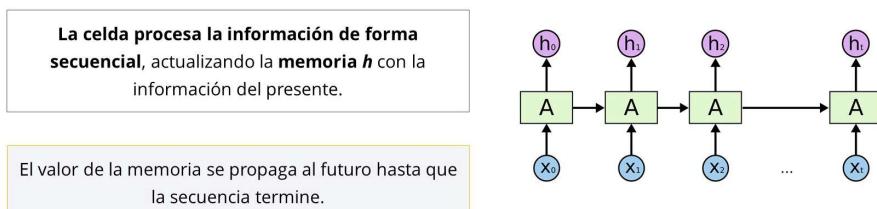
## b. Redes convolucionales



- i. Para extraer un conjunto de características que la red puede usar para realizar una predicción, el resultado de la última capa convolucional se transforma en un vector.

## 22. Redes neuronales recurrentes

- a. Una red neuronal convolucional no puede aprender del pasado.
- b. Si tenemos acceso a la información del pasado, podremos lograr un mejor resultado.
- c. La unidad fundamental de estas redes es la celda que procesa la información de un paso en la secuencia.



La información del paso anterior se combina con la información del paso actual.

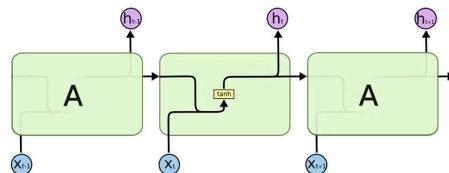
$$h_t = \tanh(\mathbf{W} \cdot h_{t-1} + \mathbf{U} \cdot x_t + b)$$

La información del paso anterior se combina con la información del paso actual.

$$h_t = \tanh(\mathbf{W} \cdot h_{t-1} + \mathbf{U} \cdot x_t + b)$$

El proceso se repite hasta que la secuencia termina.

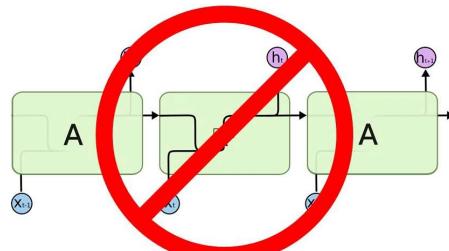
Permite extraer características de la secuencia.



#### d. Problemas:

En secuencias largas no hay memoria de largo plazo.

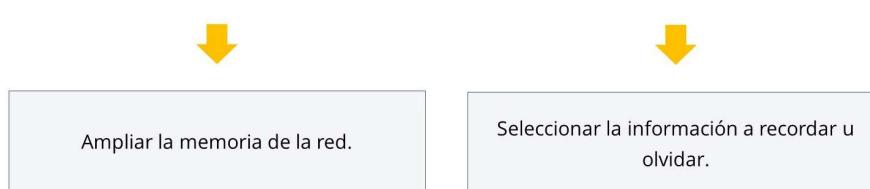
En su forma básica, las redes recurrentes presentan inestabilidades numéricas que dificultan su entrenamiento.  
**(Desaparición del gradiente)**



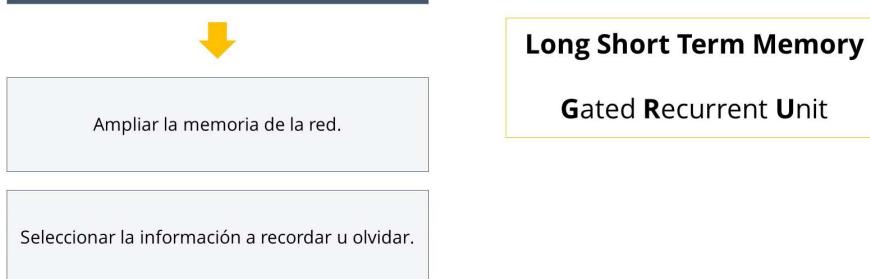
e. Para resolver el problema, se han propuesto modificaciones a las celdas recurrentes (**LSTM** y **GRU**).

f. Celdas recurrentes con compuertas:

### Celdas que introducen operaciones multiplicativas adicionales



### Celdas que introducen operaciones multiplicativas adicionales



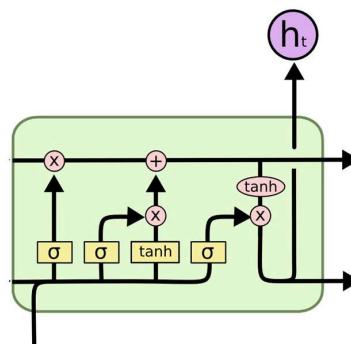
### Long Short Term Memory

#### Gated Recurrent Unit

### Celda LSTM

#### Operaciones

1. Olvidar
2. Actualizar
3. Definir la salida de la celda

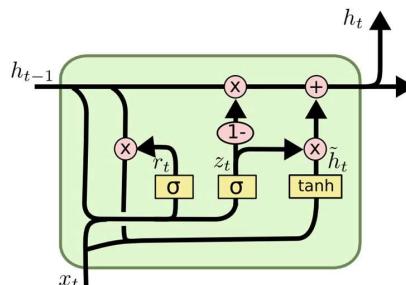


### Celda GRU

Una simplificación a la LSTM

Reemplaza las compuertas de recordar y actualizar en una sola.

Posee menos parámetros que la LSTM y su rendimiento es ligeramente inferior.



## g. Redes neuronales recurrentes profundas:

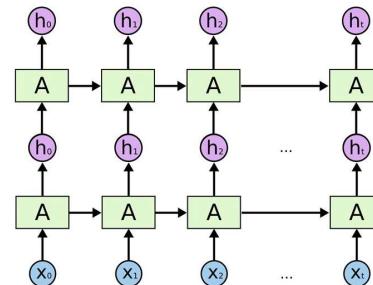
- i. Con dos o tres capas es suficiente, agregar más, no mejora los resultados, acentúa las inestabilidades numéricas, y hacen el proceso de entrenamiento más lento.

## GRU

Una capa recurrente puede extraer patrones de la secuencia.

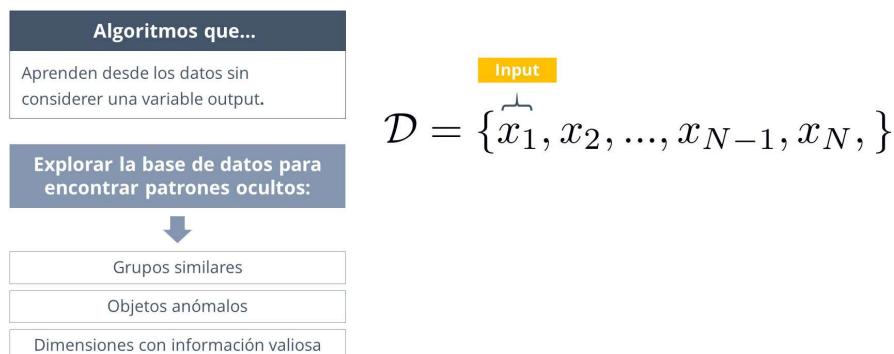
Dos o tres capas pueden extraer patrones más complejos.

La entrada de la capa siguiente son los estados ocultos de la capa recurrente anterior.



## 23. Aprendizaje no supervisado

- a. Aprende desde los datos sin considerar el input.

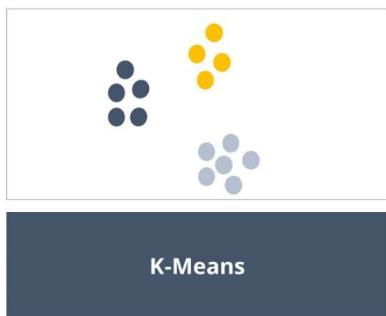


- b. Se pueden realizar herramientas como para:

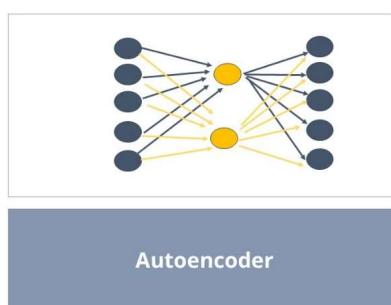


c. Estrategias estudiadas en el curso:

i. Clustering:



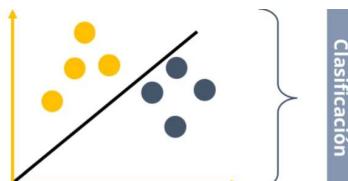
ii. Reducción de dimensionalidad:



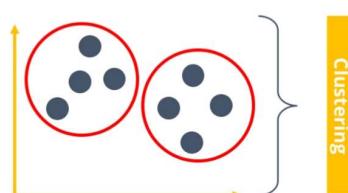
## 24. K-Means

### Clustering

1 No tenemos una etiqueta para los datos.



2 Se busca un agrupamiento de objetos parecidos.



Segmentación de clientes



Optimización de despachos



Análisis de documentos



Detección de fraudes

### a. K-Means

- i. De los más utilizados.
- ii. Descriptores numéricos.
- iii. Número de clúster debe entregarse como input.
- iv. Cada elemento pertenece solo a un clúster.

1 Se debe aplicar en dominios numéricos.

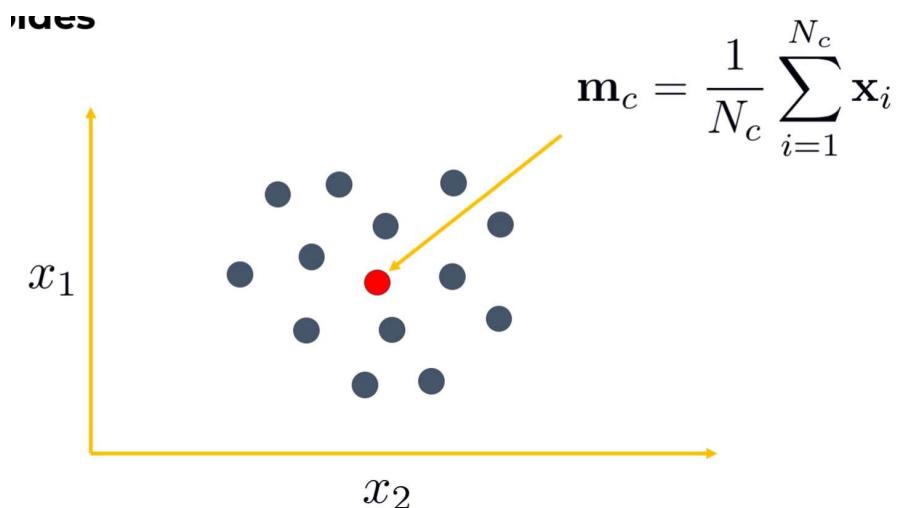
2 Genera como resultado conjuntos disjuntos de objetos.

3 El número de segmentos debe ser entregado como un *input*.

4 Algoritmo simple que ha sido utilizado por décadas.

#### v. Centroide:

- 1. Cada clúster es representado por su centroide.



vi. ¿Cómo creamos los clústeres usando K-Means?



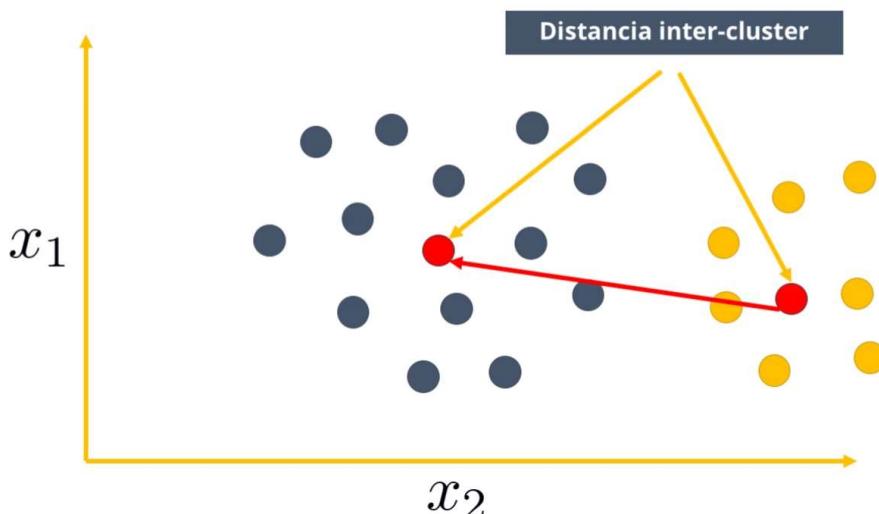
Paso 1	Paso 2
Elegir aleatoriamente K centroides.	<b>While (criterio = False):</b> <ul style="list-style-type: none"> <li>▪ Asignar cada objeto de la base de datos a su centroide más cercano.</li> <li>▪ Con las nuevas asignaciones recalcular centroides.</li> </ul>

### vii. Criterios de tención

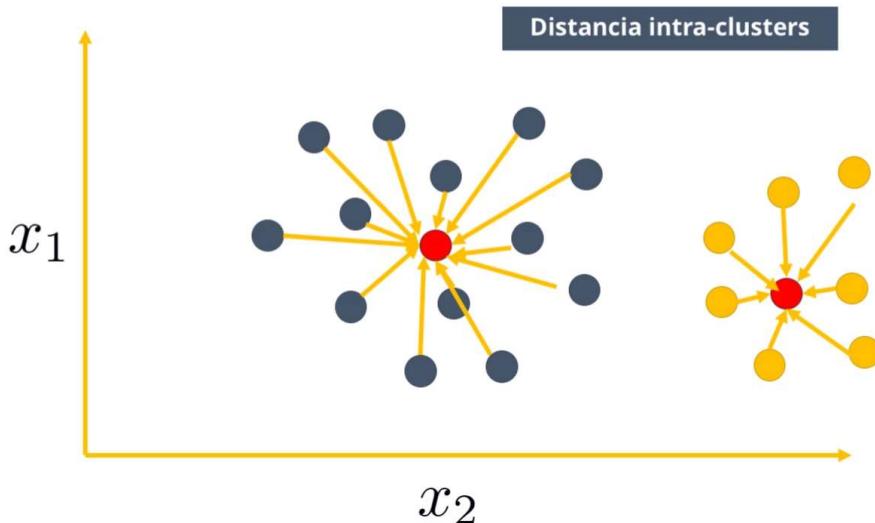
- 1** Los objetos no cambian de segmento entre una iteración y otra.
- 2** El porcentaje de objetos que varía es menor a valor predefinido.
- 3** Los centroides no cambian o cambian menos de un valor predefinido.

### viii. Análisis de clústeres:

1. Distancia inter-cluster: distancia entre los centroides de los clústeres, se asume una asignación buena de clústeres cuando existe una distancia grande entre los distintos grupos.



2. Distancia intra-cluster: lo mismo, pero la distancia intra clusteres es pequeña.



- ix. Las métricas de distancia son relevantes.
- x. Cuidado en la normalización de los descriptores, pues no queremos que la magnitud original de algunos descriptores haga que estos sean más relevantes que otros.

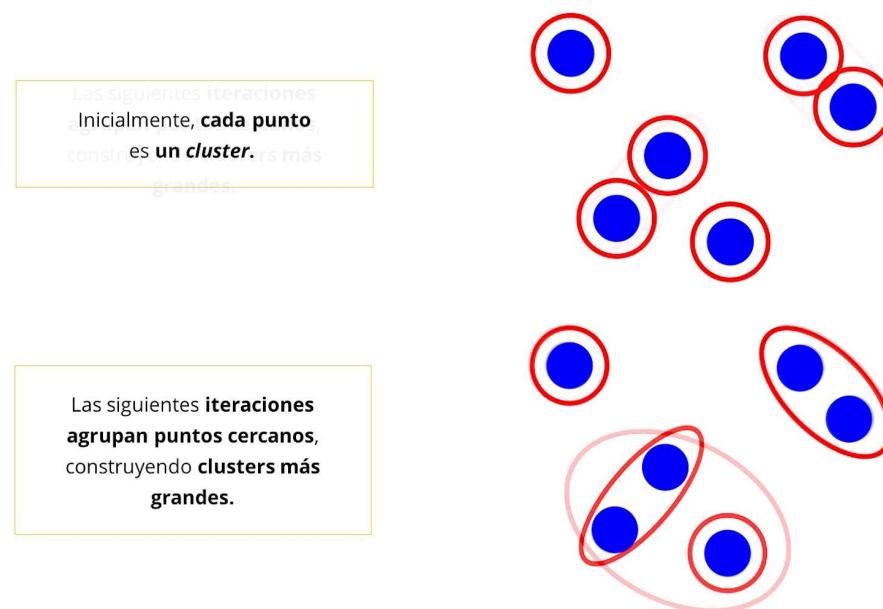
## 25. Clúster jerárquico

- Podemos separar los distintos algoritmos de *clustering* entre divisivos y aglomerativos.

- Algoritmos de *clustering* como K-means funcionan de arriba hacia abajo.
- Trabajan con una cantidad determinada de *clusters*.

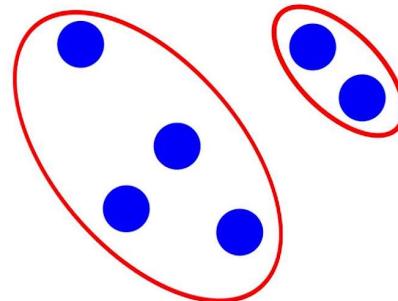
- Los métodos jerárquicos trabajan en sentido contrario.
- Buscan *clusters* asumiendo inicialmente que cada punto es uno.
- Más costosos, tanto en recursos como en tiempo.
- Menos transparentes que algoritmos como K-Means.

### a. Clustering jerárquico

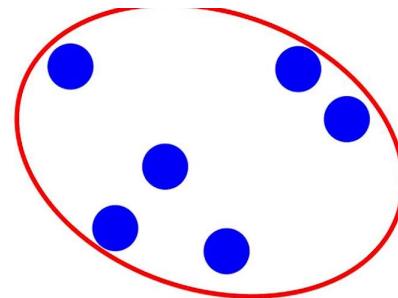


- i. Los clústeres que vayan formando tendrán formas irregulares ya que seguirán la distribución espacial de los distintos puntos.

A medida que el **proceso avanza**, **reducción** en la cantidad de *clusters*.



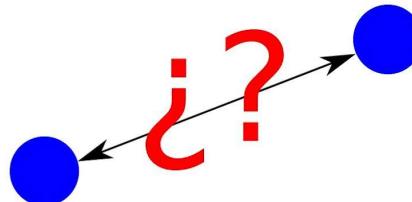
El **proceso termina** al quedar un único *cluster*.



## b. Métricas de distancia

Todo algoritmo de clustering tiene como **parámetro de entrada** una función que define la **distancia entre los puntos**.

La **selección de la métrica** es clave en el resultado.



Distintas formas de medir distancia pueden entregar resultados muy distintos.

**Euclideana**

$$\sqrt{\sum_{i=1}^N (p_i - q_i)^2} = \sqrt{\mathbf{p} \cdot \mathbf{q}}$$

**Minkowski**

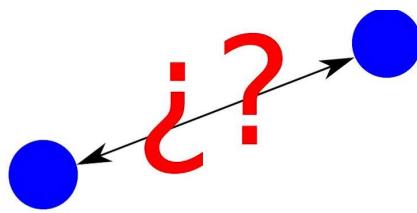
$$\left( \sum_{i=1}^N |p_i - q_i|^p \right)^{\frac{1}{p}}$$

**Coseno**

$$\cos \theta = \frac{\mathbf{p} \cdot \mathbf{q}}{\|\mathbf{p}\| \|\mathbf{q}\|}$$

**¡Cuidado!**

A altas dimensionalidades,  
la métrica de distancia pierde  
sentido.

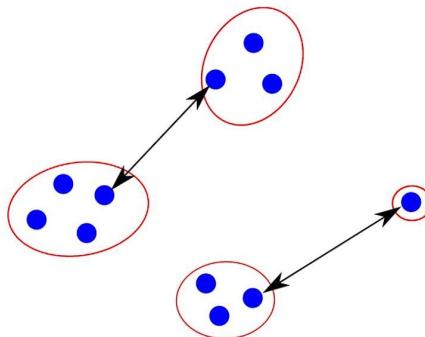


- i. Al perder sentido todos los puntos terminarán en una distancia similar. Por lo tanto, antes de hacer cualquier clústeres debemos seleccionar la menor cantidad de dimensiones posible, ya sea por medio de selección de variables o PCA.

### c. Modos de unión

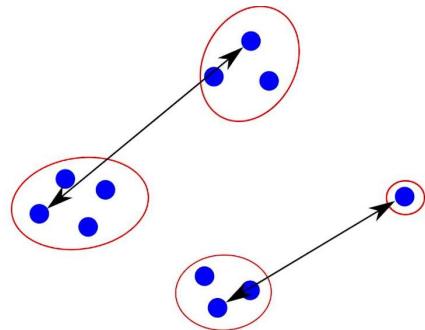
- i. Simple:

Distancia entre los elementos  
más cercanos de dos *clusters*.

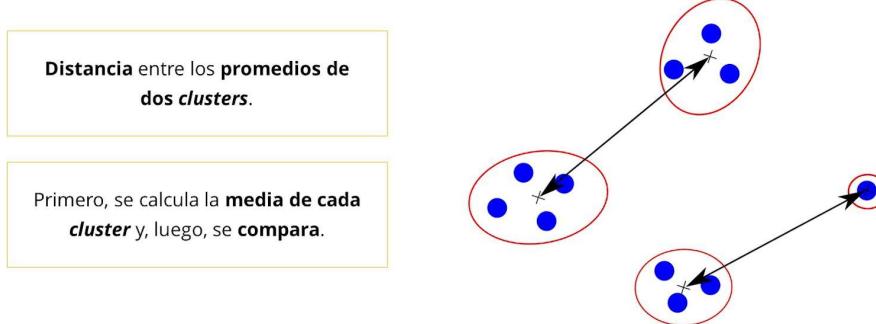


- ii. Completa:

Distancia entre los elementos  
más lejanos de dos *clusters*.

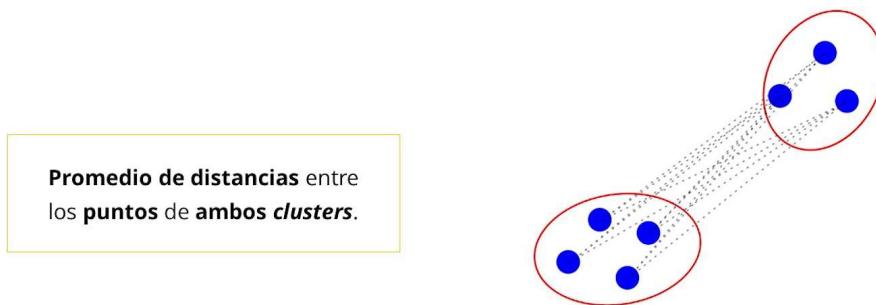


- iii. Promedio:



1. Toma en cuenta todos los puntos, pero es menos sensibles a formas irregulares.

#### iv. Promedio entre pares:



1. De esta forma se toma en cuenta todos los puntos representando mejor la forma de cada clase, pero ocupa más recursos.

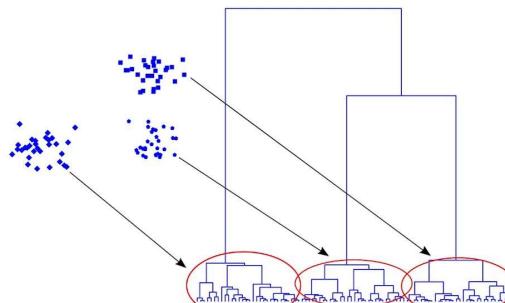
#### v. ¿Cuál usar?

1. Usar la menor cantidad de datos y dimensiones posibles.
2. Tener en cuenta la cantidad total de datos.
3. Los recursos computacionales posibles.
4. El tiempo disponible.
5. Lo que más importa es el objetivo del modelo.

#### d. Dendograma

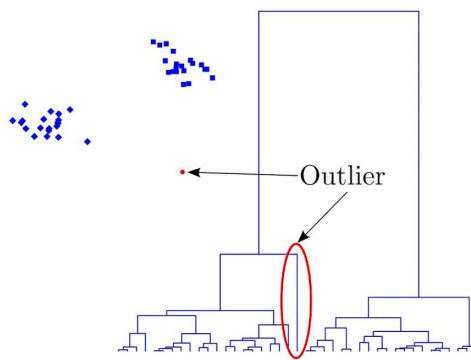
El **dendograma** es una forma de visualizar el resultado del algoritmo en función de la escala de similaridad.

En el **eje Y** se muestra la distancia y en el **eje X** se **ordenan los datos**.



Los elementos que se integran a algún cluster a un nivel de similaridad alto en comparación al resto de los puntos, probablemente es un **outlier**.

Disminución del efecto a **dimensionalidad alta**, donde la medida de distancia pierde sentido.



### 26. Evaluación de Clústeres

#### a. Algunas características del problema:

1

No hay etiqueta

2

Métricas de distancia mejoran a medida que aumenta el número de *clusters*

3

En alta dimensionalidad es complejo interpretar o visualizar

#### b. Métricas para evaluación de clústeres usando Python:

- i. Índice Davies-Bouldin: busca que los datos no sean parecidos entre sí. Usa distancia intra-cluster e intra-cluster

$$\text{DB index} = \frac{1}{n} \sum_{i=1}^n \max_{j \neq i} \frac{(\sigma_i + \sigma_j)}{d(c_i, c_j)}$$

$n$  = Número de *clusters*

$\sigma_i$  = Distancia intra-*cluster* del *cluster*  $i$

$d(c_i, c_j)$  = Distancia entre los centroides de los *clusters*  $i$  y  $j$

```

1 from sklearn import datasets
2 diabetes = datasets.load_diabetes()
3 X = diabetes.data
4
5 from sklearn.cluster import KMeans
6 from sklearn.metrics import davies_bouldin_score
7
8 for k in range (2, 10):
9     kmeans = KMeans(n_clusters=k, random_state=1).fit(X)
10    labels = kmeans.labels_
11    print(str(k) + ' clusters')
12    index = davies_bouldin_score(X, labels)
13    print('David Bouldin score = ' + str(index))

```

## ii. Coherencias:

<b>Número que indica coherencia del objeto <math>i</math> dentro del cluster que fue asignado.</b>	<b>Distancia promedio más pequeña del objeto <math>i</math> al resto de los puntos en otros clusters.</b>	<b>Distancia promedio del objeto <math>i</math> al resto de los puntos del cluster.</b>
--	---	---

$$\{ S(i) = \frac{\overbrace{b(i)} - \overbrace{a(i)}}{\max\{a(i), b(i)\}}$$

$$-1 \leq S(i) \leq 1$$

## iii. Silueta:

```

1 #!pip install yellowbrick
2
3
4
5
6
7
8
9
10

```

```

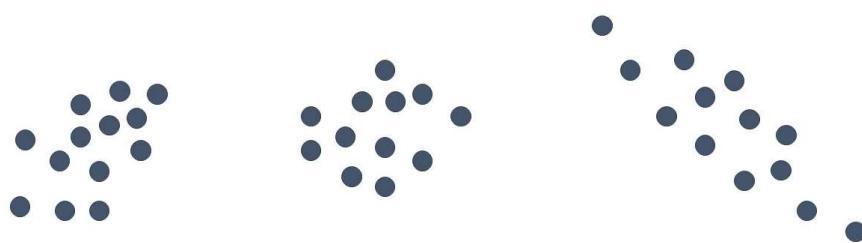
1 from sklearn.cluster import KMeans
2 from yellowbrick.cluster import SilhouetteVisualizer
3 from yellowbrick.datasets import load_nfl
4
5 X, y = load_nfl()
6 features = ['Red', 'Yds', 'TD', 'Fmb']
7 model = KMeans(6, random_state=42)
8 plot = SilhouetteVisualizer(model)
9 plot.fit(X)
10 plot.show()

```

## 27. Reducción de dimensionalidad

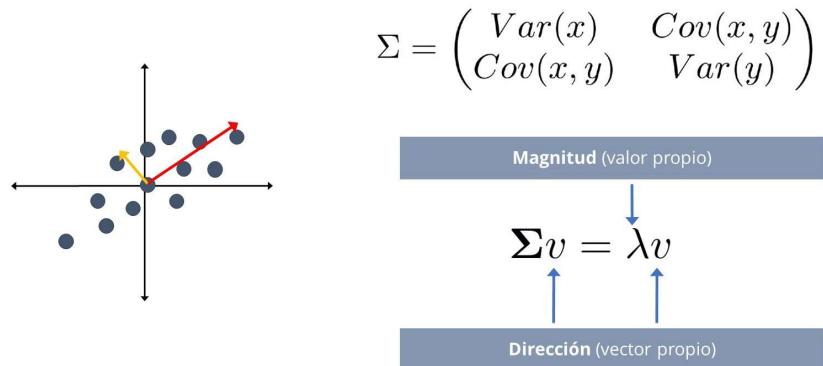
- a. Análisis de componentes principales:

### Algunas razones para reducir dimensionalidad

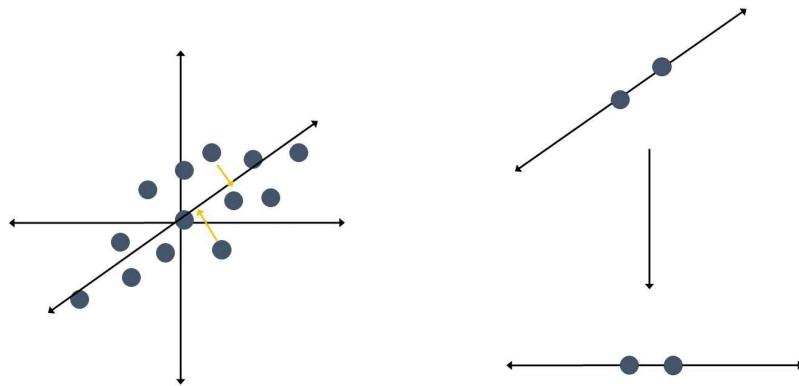


$$\Sigma = \begin{pmatrix} Var(x) & Cov(x, y) \\ Cov(x, y) & Var(y) \end{pmatrix}$$

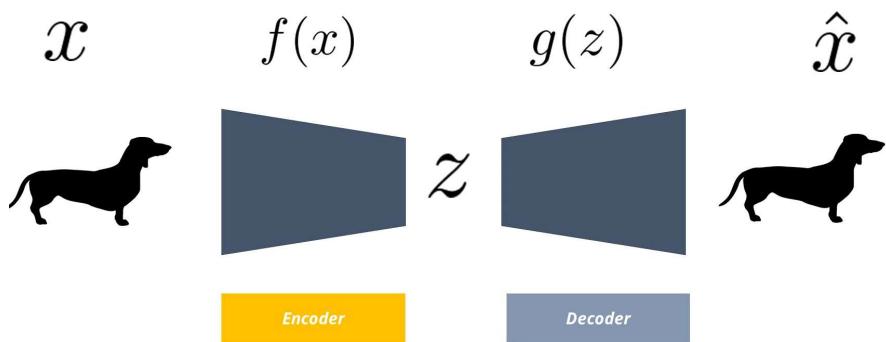
## Valores propios y vectores propios



## ¿Cuál componente aporta más información?



- i. Los pasos para reducir dimensionalidades son:
  1. Normalizar los datos (descriptores).
  2. Definir matriz de covarianza.
  3. Seleccionar el mayor valor propio y proyectar sobre él.
  
- b. Autoencoder:
  - i. Es una red neuronal entrenada para replicar el input en el output. Ejemplo,



## 28. Ayudantía 3

- a. Modelo probabilístico que supone que cada punto o cada conjunto de atributos pertenecen a una distribución gaussiana, busca una media en común y una covarianza.
- b. Utiliza la matriz de covarianza.