

Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación



# IIC2115 - Programación como Herramienta para la Ingeniería

Modelos predictivos con Machine Learning

**Profesora:** Francesca Lucchini  
**Prof. Coordinador:** Hans Löbel

# ¿Qué es el análisis de datos (en Python)?

- Desde un punto de vista práctico, consiste principalmente en utilizar herramientas para:
  - Limpiar y transformar los datos
  - Explorar distintas dimensiones de los datos
  - Calcular estadísticas de los datos
  - Visualizar los datos
  - Construir modelos predictivos
- Para todo esto (y más), está **Pandas** y **scikit-learn**



En esta segunda parte nos centraremos en **scikit-learn**

- Implementa gran cantidad de algoritmos predictivos y de procesamiento de datos.
- Permite una fácil integración con Pandas y numpy.



Antes de revisar *scikit-learn*, necesitamos una breve introducción al **Aprendizaje de Máquina (Machine Learning)**

¿Cómo puedo saber si en una foto hay un perro o un gato?  
(usando un computador, lógicamente)

¿



?



Cat



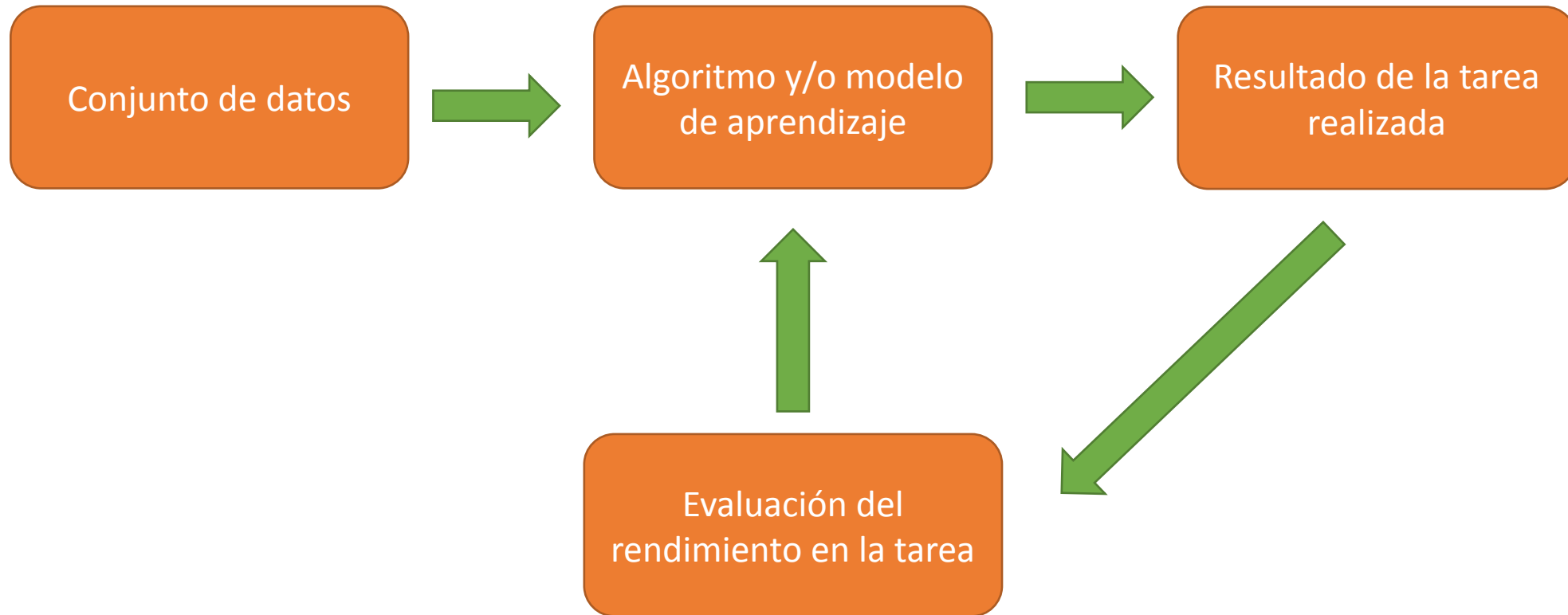
Dog

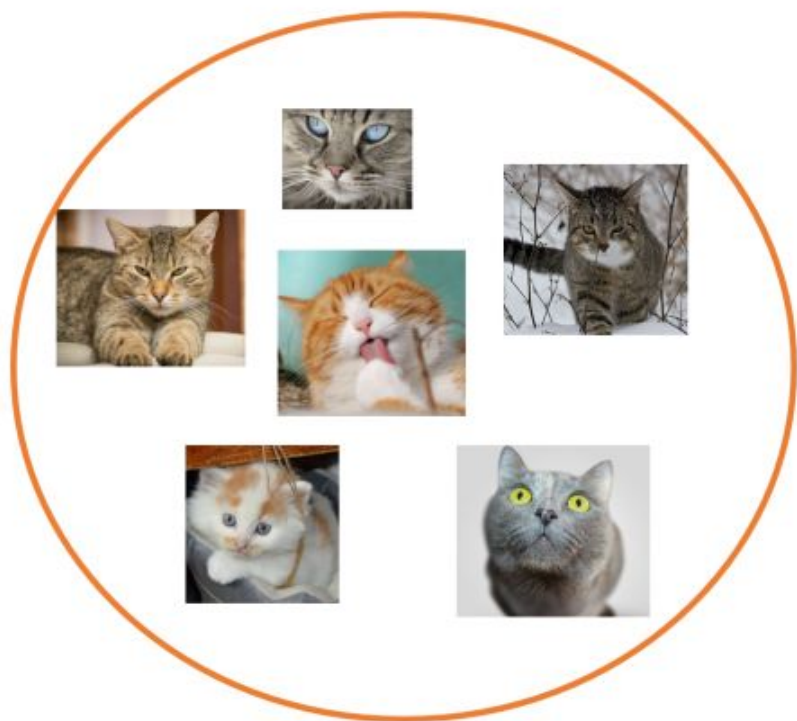
# ¿Qué es Machine Learning (ML)?

- En simple, se trata de algoritmos que procesan **datos** para realizar una **tarea** (predicción, clasificación, clustering, etc.)
- Más específicamente, se centra en el estudio de algoritmos que mejoran su **rendimiento** en una **tarea**, a través de la experiencia (**aprendizaje** desde los **datos**).
- Buscan resolver la **tarea** con la mayor precisión posible, más que entender el fenómeno subyacente.



(casi) Todas las técnicas de ML usan el mismo esquema de procesamiento





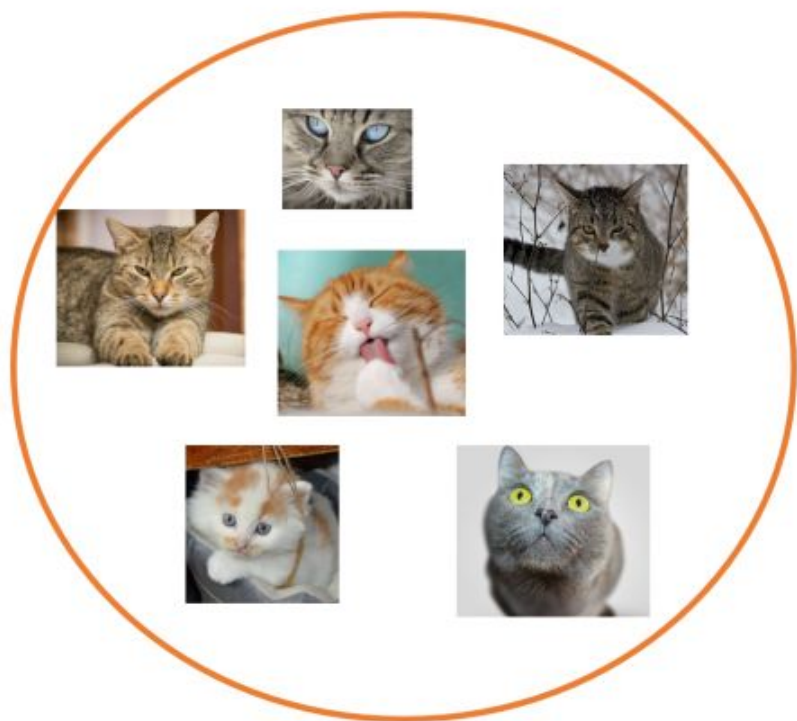
Cat



Dog







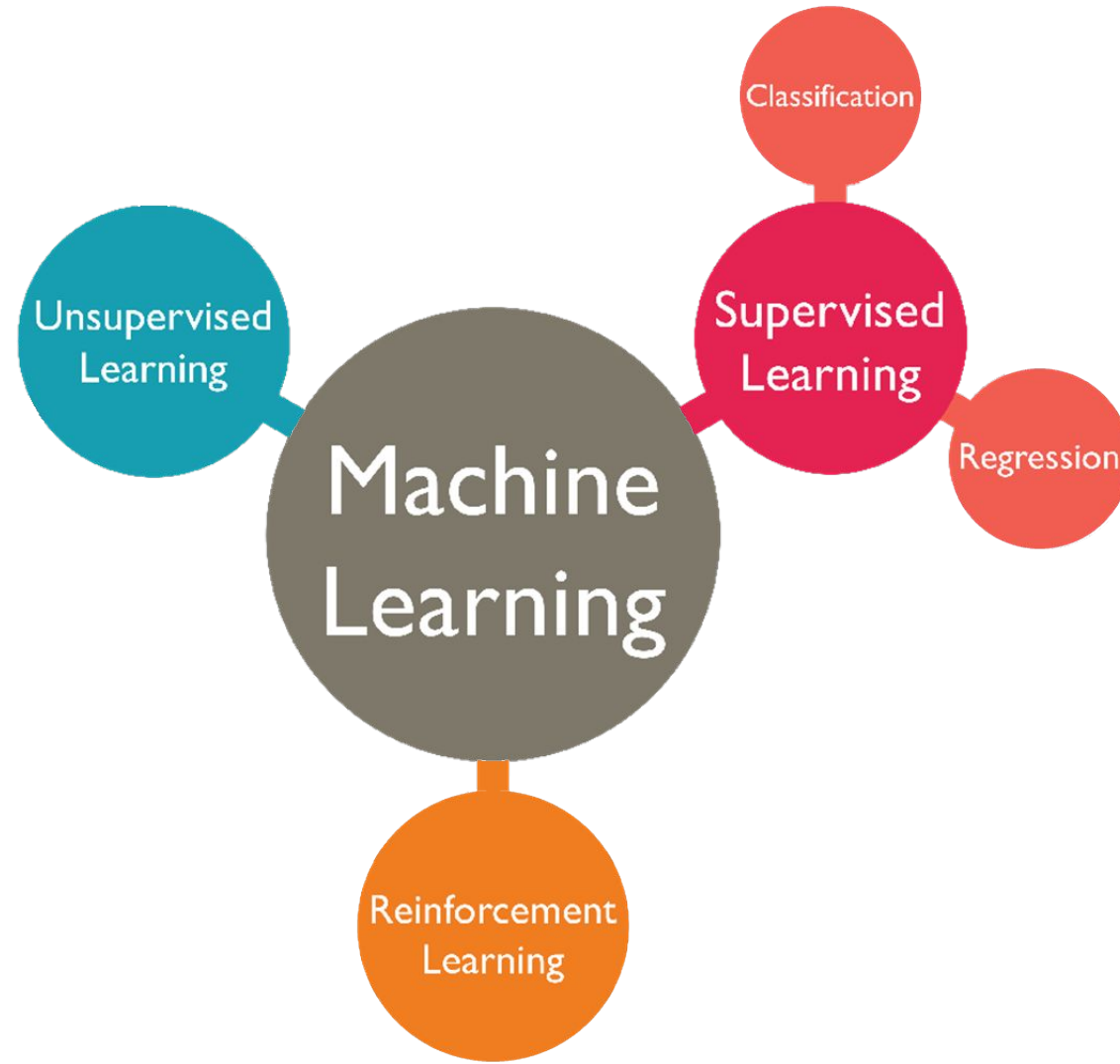
Cat



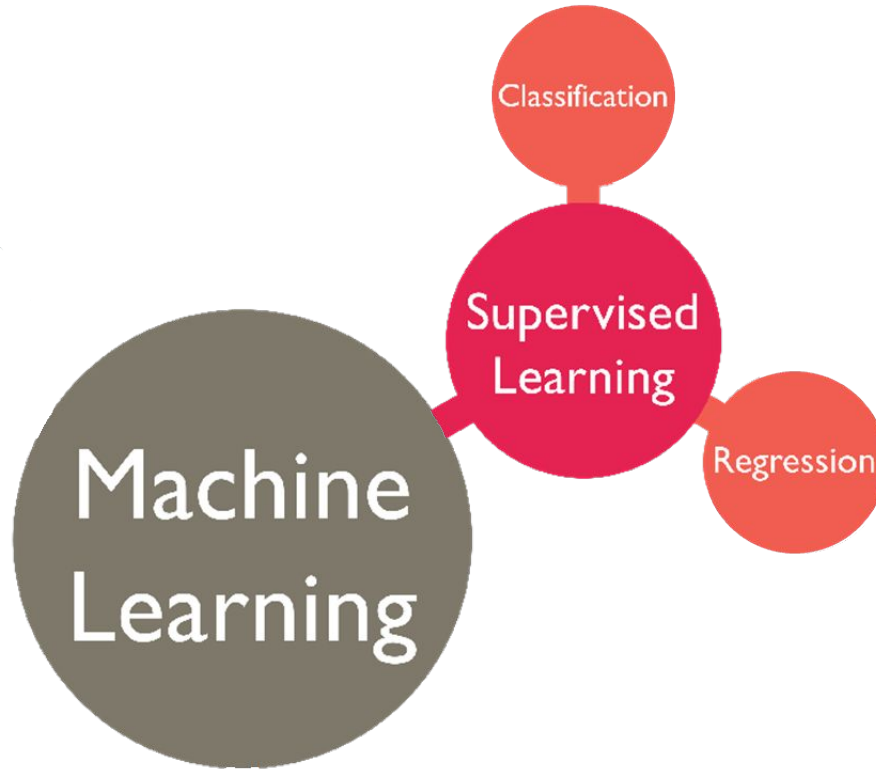
Dog



# Tipos de Algoritmos de ML



## Tipos de Algoritmos de ML que conoceremos...



## ¿Cómo se manejan los **datos**?

Técnicas de ML trabajan sobre datos **multidimensionales**

- Cada dato está caracterizado por una serie de **características** = **mediciones** = **atributos** = **variables**.
- La cantidad de características define la **dimensionalidad** del dato.
- El espacio donde viven los datos se conoce como **espacio de características** (*feature space*).


| Distance from the eye<br>of the storm (km) | Wind speed at site<br>(m/s) | Pressure deficit<br>at site (hPa) | Forward speed of the<br>eye of the storm (km/h) | Storm surge<br>(cm) |
|--|-----------------------------|-----------------------------------|---|---------------------|
| 96.0                                       | 20.7                        | 20.6                              | 27.6  | 47.4                |
| 108.5                                      | 15.4                        | 11.0                              | 58.9  | 24.5                |
| 181.2                                      | 8.1                         | 1.7                               | 40.1  | 7.9                 |
| 245.3                                      | 5.7                         | 6.4                               | 29.6  | 5.5                 |
| 117.5                                      | 23.3                        | 22.0                              | 46.6  | 61.7                |
| 231.4                                      | 13.3                        | 11.5                              | 38.1  | 20.8                |
| 293.6                                      | 4.0                         | 7.2                               | 35.4  | 5.6                 |
| 0.6  | 8.5                         | 7.0                               | 32.2  | 8.7                 |
| 227.6                                      | 10.0                        | 10.4                              | 19.3  | 16.0                |
| 257.3                                      | 11.5                        | 15.0                              | 44.1  | 10.8                |

Cada columna puede verse como un eje en el espacio de características



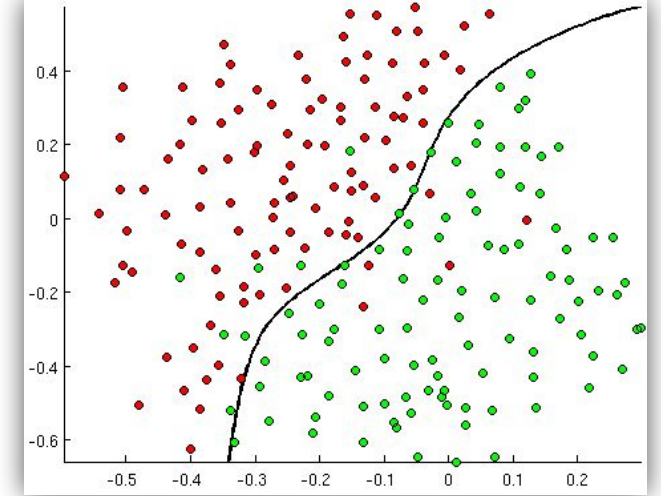
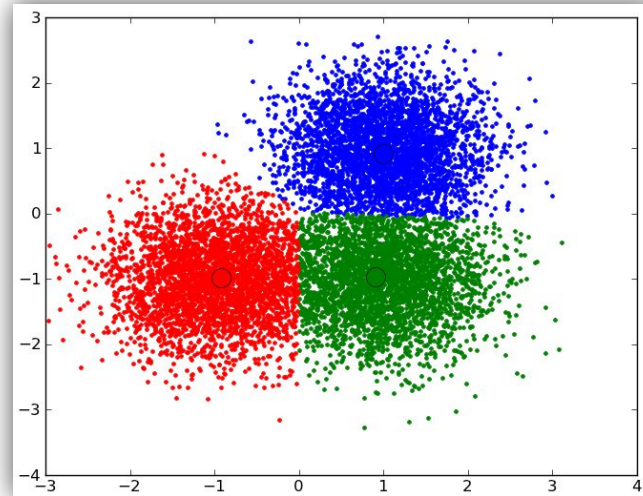
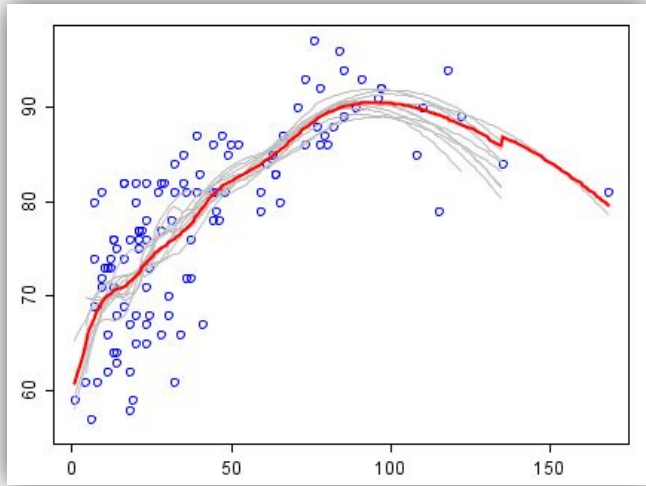
| Distance from the eye<br>of the storm (km) | Wind speed at site<br>(m/s) | Pressure deficit<br>at site (hPa) | Forward speed of the<br>eye of the storm (km/h) | Storm surge<br>(cm) |
|--|-----------------------------|-----------------------------------|---|---------------------|
| 96.0                                       | 20.7                        | 20.6                              | 27.6  | 47.4                |
| 108.5                                      | 15.4                        | 11.0                              | 58.9  | 24.5                |
| 181.2                                      | 8.1                         | 1.7                               | 40.1  | 7.9                 |
| 245.3                                      | 5.7                         | 6.4                               | 29.6  | 5.5                 |
| 117.5                                      | 23.3                        | 22.0                              | 46.6  | 61.7                |
| 231.4                                      | 13.3                        | 11.5                              | 38.1  | 20.8                |
| 293.6                                      | 4.0                         | 7.2                               | 35.4  | 5.6                 |
| 0.6  | 8.5                         | 7.0                               | 32.2  | 8.7                 |
| 227.6                                      | 10.0                        | 10.4                              | 19.3  | 16.0                |
| 257.3                                      | 11.5                        | 15.0                              | 44.1  | 10.8                |

Cada dato puede verse como un vector/punto en el espacio de características



| Distance from the eye<br>of the storm (km) | Wind speed at site<br>(m/s) | Pressure deficit<br>at site (hPa) | Forward speed of the<br>eye of the storm (km/h) | Storm surge<br>(cm) |
|--|-----------------------------|-----------------------------------|---|---------------------|
| 96.0                                       | 20.7                        | 20.6                              | 27.6  | 47.4                |
| 108.5                                      | 15.4                        | 11.0                              | 58.9  | 24.5                |
| 181.2                                      | 8.1                         | 1.7                               | 40.1  | 7.9                 |
| 245.3                                      | 5.7                         | 6.4                               | 29.6  | 5.5                 |
| 117.5                                      | 23.3                        | 22.0                              | 46.6  | 61.7                |
| 231.4                                      | 13.3                        | 11.5                              | 38.1  | 20.8                |
| 293.6                                      | 4.0                         | 7.2                               | 35.4  | 5.6                 |
| 0.6  | 8.5                         | 7.0                               | 32.2  | 8.7                 |
| 227.6                                      | 10.0                        | 10.4                              | 19.3  | 16.0                |
| 257.3                                      | 11.5                        | 15.0                              | 44.1  | 10.8                |

Cada dato puede verse como un **vector/punto** en el espacio de características



Cada columna puede verse como un **eje** en el espacio de características

¿Cómo **aprende** un modelo de ML?  
Pasan por una etapa de **entrenamiento** y una de **prueba**

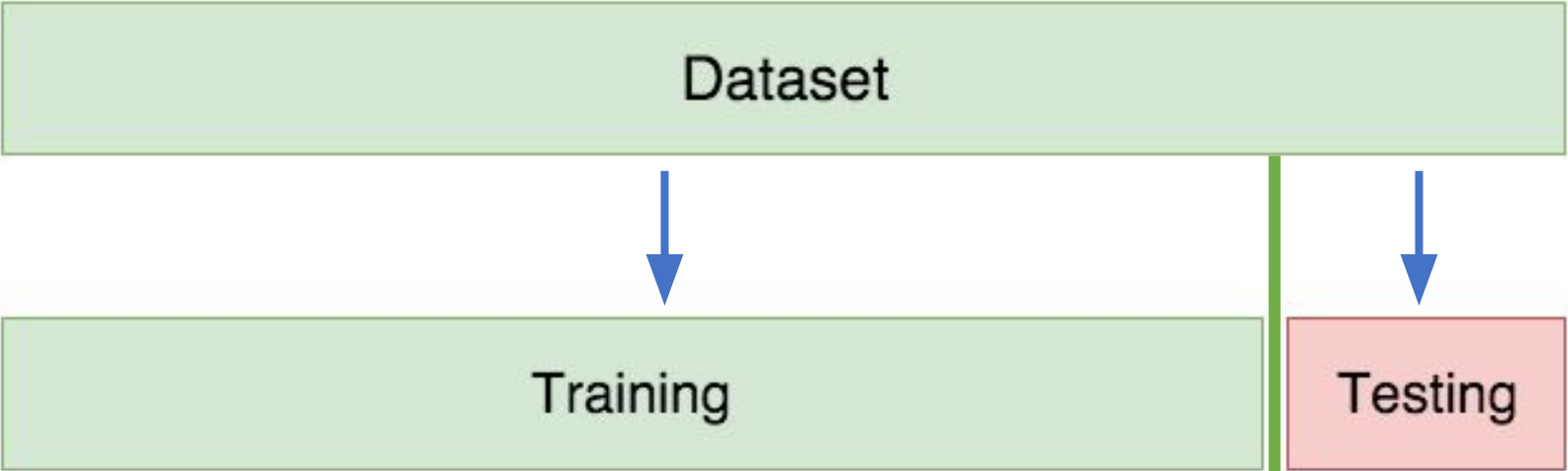


Para **entrenar** = ajustar = calibrar un modelo, se utiliza un **set de entrenamiento**

|               | Input vectors                           |                          |                                |  | Response vector  |
|---------------|---|--------------------------|--------------------------------|--|------------------|
|               | Distance from the eye of the storm (km) | Wind speed at site (m/s) | Pressure deficit at site (hPa) | Forward speed of the eye of the storm (km/h) | Storm surge (cm) |
| Entrenamiento | 96.0                                    | 20.7                     | 20.6                           | 27.6   | 47.4             |
|               | 108.5                                   | 15.4                     | 11.0                           | 58.9   | 24.5             |
|               | 181.2                                   | 8.1                      | 1.7                            | 40.1   | 7.9              |
|               | 245.3                                   | 5.7                      | 6.4                            | 29.6   | 5.5              |
|               | 117.5                                   | 23.3                     | 22.0                           | 46.6   | 61.7             |
|               | 231.4                                   | 13.3                     | 11.5                           | 38.1   | 20.8             |
|               | 293.6                                   | 4.0                      | 7.2                            | 35.4   | 5.6              |
|               | 0.6                                     | 8.5                      | 7.0                            | 32.2   | 8.7              |
|               | 227.6                                   | 10.0                     | 10.4                           | 19.3   | 16.0             |
|               | 257.3                                   | 11.5                     | 15.0                           | 44.1   | 10.8             |
| Test          | 290.6                                   | 9.5                      | 13.6                           | 46.9   | ?                |
|               | 245.3                                   | 10.6                     | 14.2                           | 77.6   |                  |
|               | 227.0                                   | 4.4                      | 7.9                            | 20.8   |                  |
|               | 279.1                                   | 4.4                      | 7.8                            | 29.5   |                  |
|               | 266.3                                   | 8.7                      | 8.8                            | 32.9   |                  |
|               | 165.6                                   | 19.2                     | 16.4                           | 45.6   |                  |
|               | 136.5                                   | 10.7                     | 12.2                           | 4.6  |                  |
|               | 207.9                                   | 4.4                      | 8.0                            | 14.1   |                  |

**Set de test** es útil para evaluar la capacidad de generalización del modelo

Una forma clara de ver esto es con conjuntos de datos disjuntos



Teniendo las bases cubiertas, podemos revisar *scikit-learn*

## En este curso usaremos scikit-learn

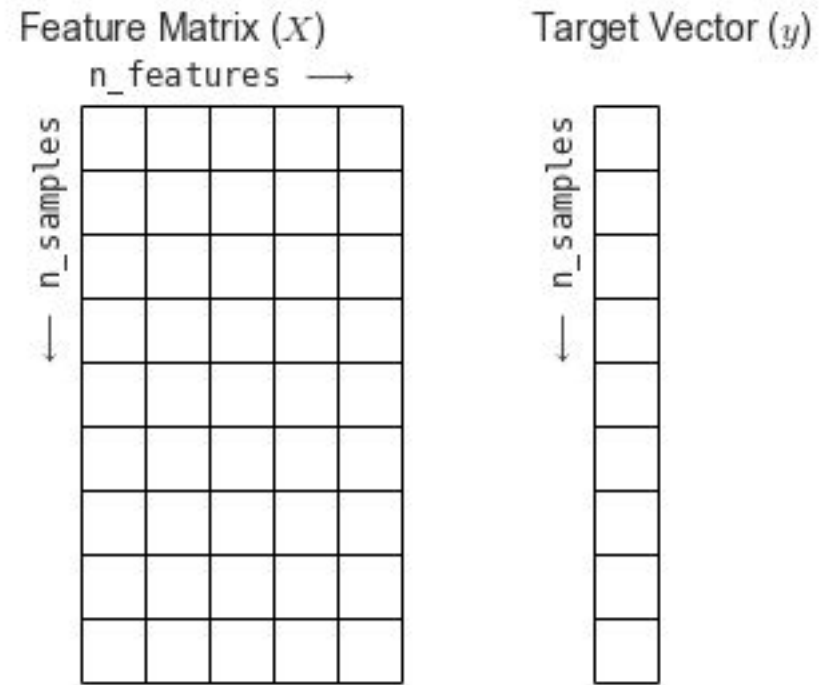
- scikit-learn es el módulo para ML más conocido y utilizado en Python.
- Su principal atractivo es una interfaz limpia, uniforme y simple, que facilita la exploración y permite la integración con otros paquetes, como Pandas.
- Posee además de una completa documentación en línea (<https://scikit-learn.org/>).



## Esquema de datos es similar a Pandas

- Los datos son representados por una *matriz de features* y un *vector objetivo* (etiquetas de los datos)
- Las características de los ejemplos se almacenan en una matriz de *features* (**X**), de tamaño [n\_samples, n\_features] (esta matriz puede ser un **DataFrame**).
- El vector objetivo (**y**) contiene el valor a predecir para cada ejemplo y tiene tamaño [n\_samples, 1] (este vector puede ser una *Series*).
- Y eso es todo...

Esquema de datos es similar a Pandas



# Interfaz para usar modelos/algoritmos

- La interfaz de scikit-learn se basa en los siguientes conceptos principales:
  - Consistente: todos los modelos comparten una **interfaz** con unas pocas funciones.
  - Sucinta: solo usa clases propias para los algoritmos. Para todo el resto utiliza formatos estándares (datos en DataFrame por ejemplo).
  - Útil: los parámetros por defecto son útiles para estimar adecuadamente los modelos.
- En resumen, requiere muy poco esfuerzo utilizarla y obtener resultados rápidamente.

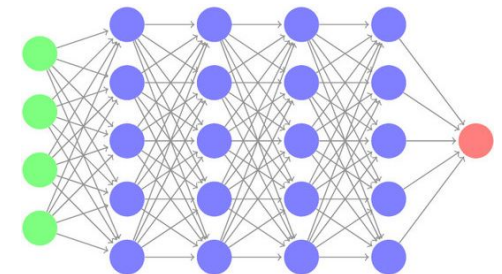
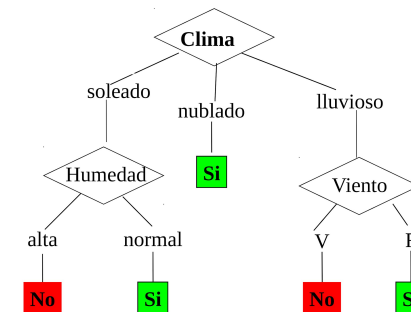
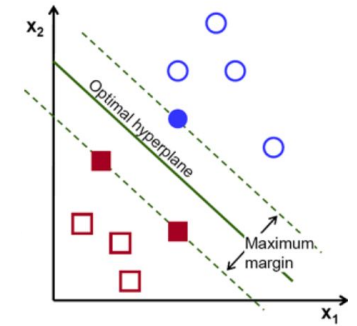
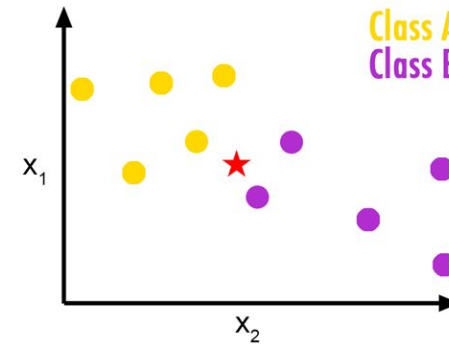
# Interfaz para usar modelos

- En general, un caso de uso típico en Scikit-learn es como el siguiente:
  1. Elegir el modelo adecuado, importando la clase correspondiente desde *sklearn*.
  2. Obtener o generar matriz *X* y vector *y*.
  3. Entrenar el modelo llamando al *fit(X, y)*.
  4. Aplicar el modelo al set de test, usando el método *predict()*.
- Al igual que para los datos, se requiere muy poco esfuerzo para obtener resultados rápidamente.



# Podrán utilizar múltiples modelos/algoritmos en este capítulo

- k-NN
- Regresiones (lineal, logística, polinomial)
- SVM
- Árboles de decisión
- Ensamblados
- Redes neuronales
- y más...

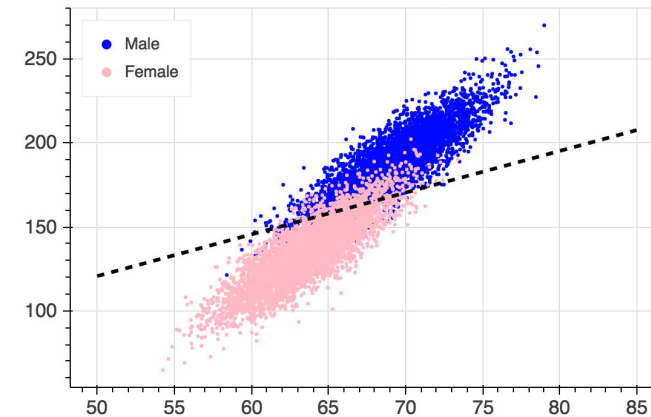
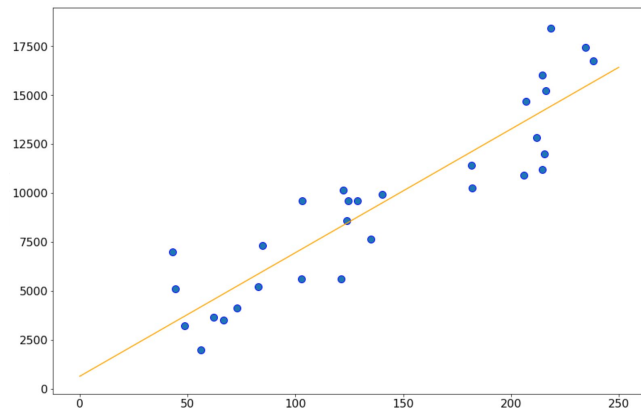


# Ejemplo: Regresión lineal y logística

- Se encuentran en el módulo `sklearn.linear_model`
- Para instanciarlas, utilizamos los siguientes comandos:

```
model = linear_model.LinearRegression()
```

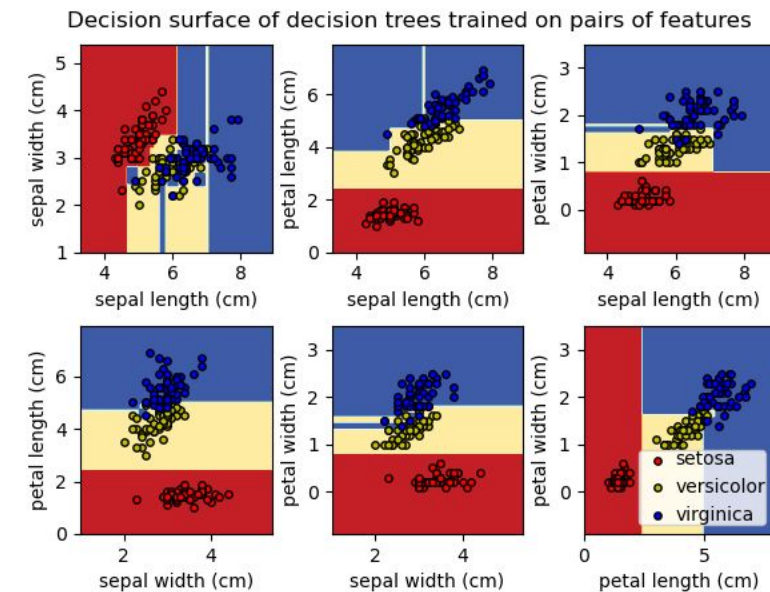
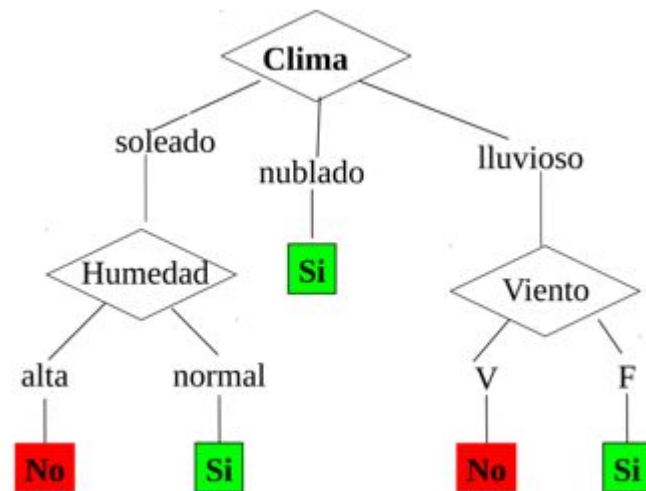
```
model = linear_model.LogisticRegression()
```



# Ejemplo: Árboles de decisión y regresión

- Se construye un árbol que en base a **análisis de cada característica**, genera **reglas de decisión**
- Se encuentran en el módulo **sklearn.tree**
- Para instanciarlo, utilizamos el siguiente comando:

```
model = tree.DecisionTreeClassifier()
```



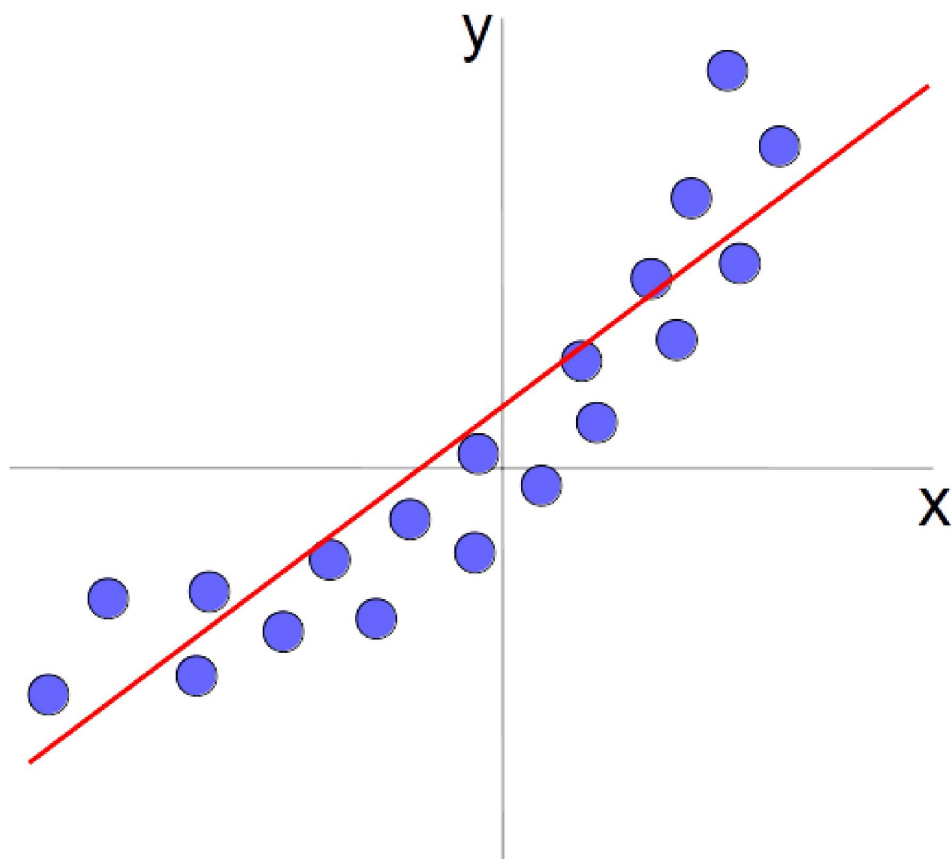
## ¿Cómo elegimos el mejor modelo para cada tarea?

- El primer paso consiste en analizar y explorar los datos.
- En base a esto, se eligen algunos modelos candidatos y se evalúa su rendimiento.
- scikit-Learn entrega una gran cantidad de métricas de rendimiento para distintos tipos de problema.
- Se encuentran en el módulo `sklearn.metrics`
- En la práctica, las más usadas son *accuracy*, *precision*, *recall*, error cuadrático medio y matriz de confusión.

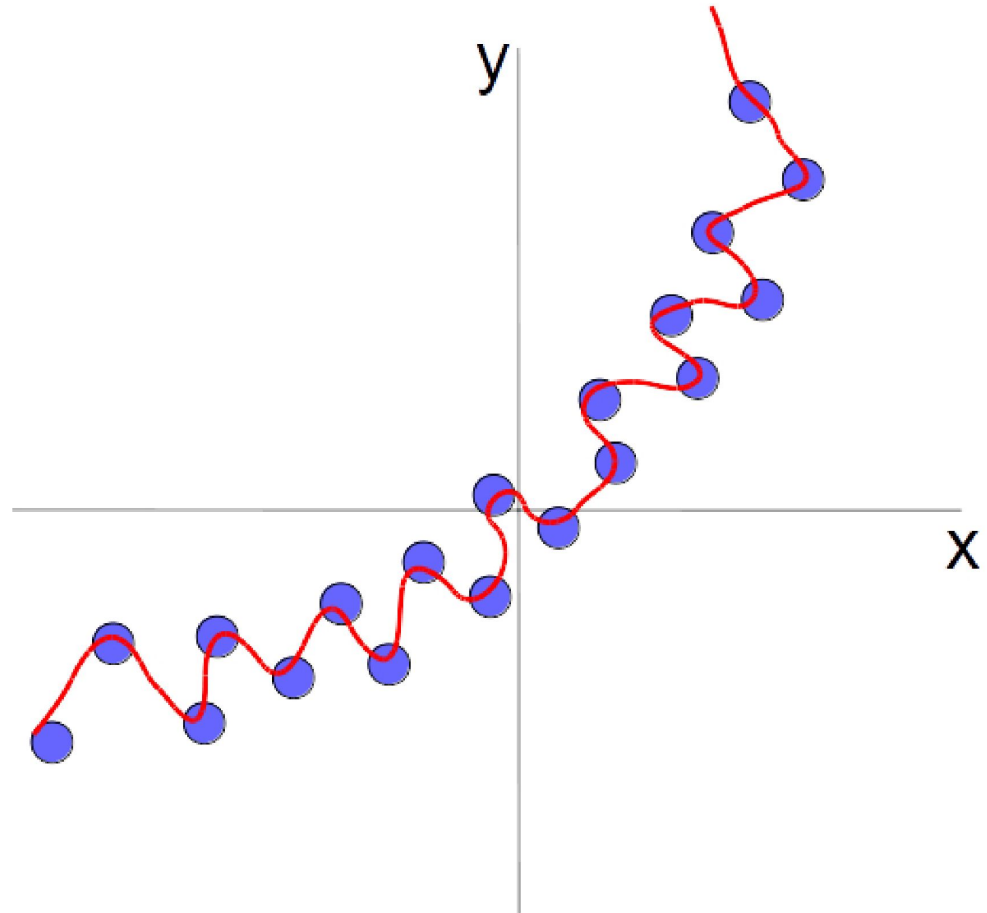
A pesar de ser clave, el **set de entrenamiento** no lo es todo

- En general, los algoritmos de aprendizaje viven y mueren por el set de entrenamiento.
- Lamentablemente, tener un buen set de entrenamiento, **no asegura tener buena generalización**.
- La complejidad del modelo (cuánto puede aprender) pasa a ser un tema central.
- Si no tenemos cuidado, podemos toparnos con los siguientes problemas...

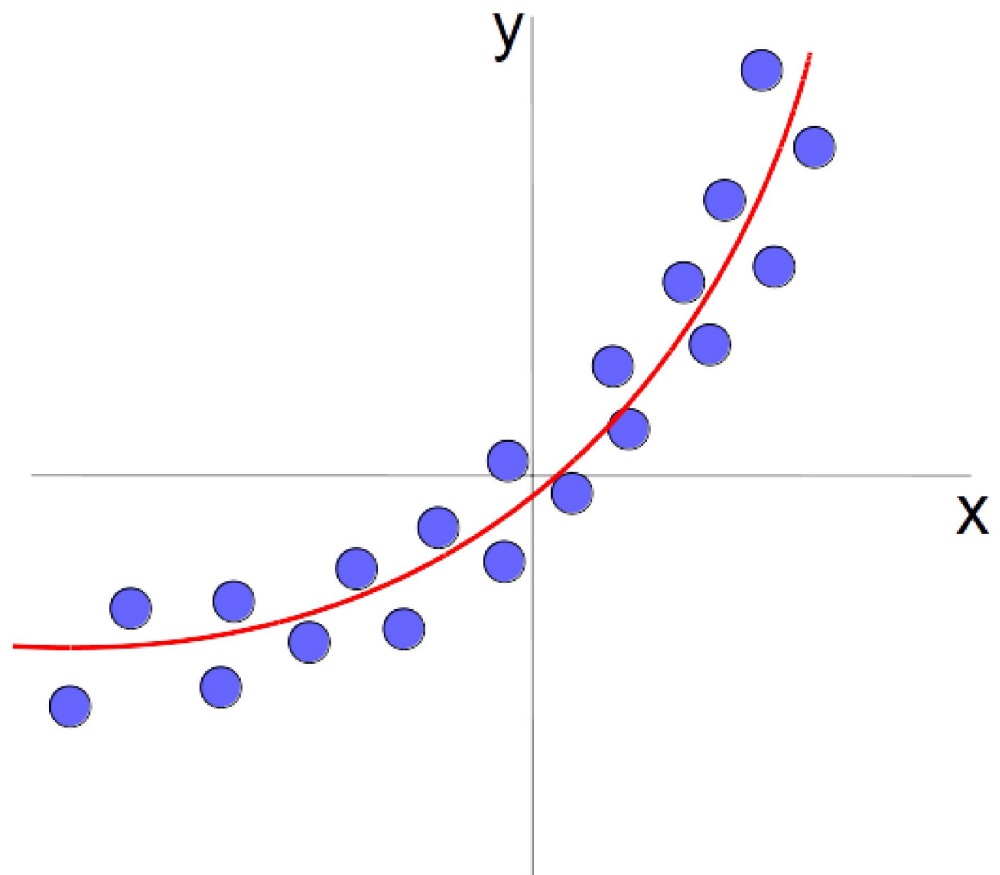
## Subentrenamiento (o subajuste, o *underfitting*)



## Sobreentrenamiento (o sobreajuste, u overfitting)



## Complejidad correcta del modelo

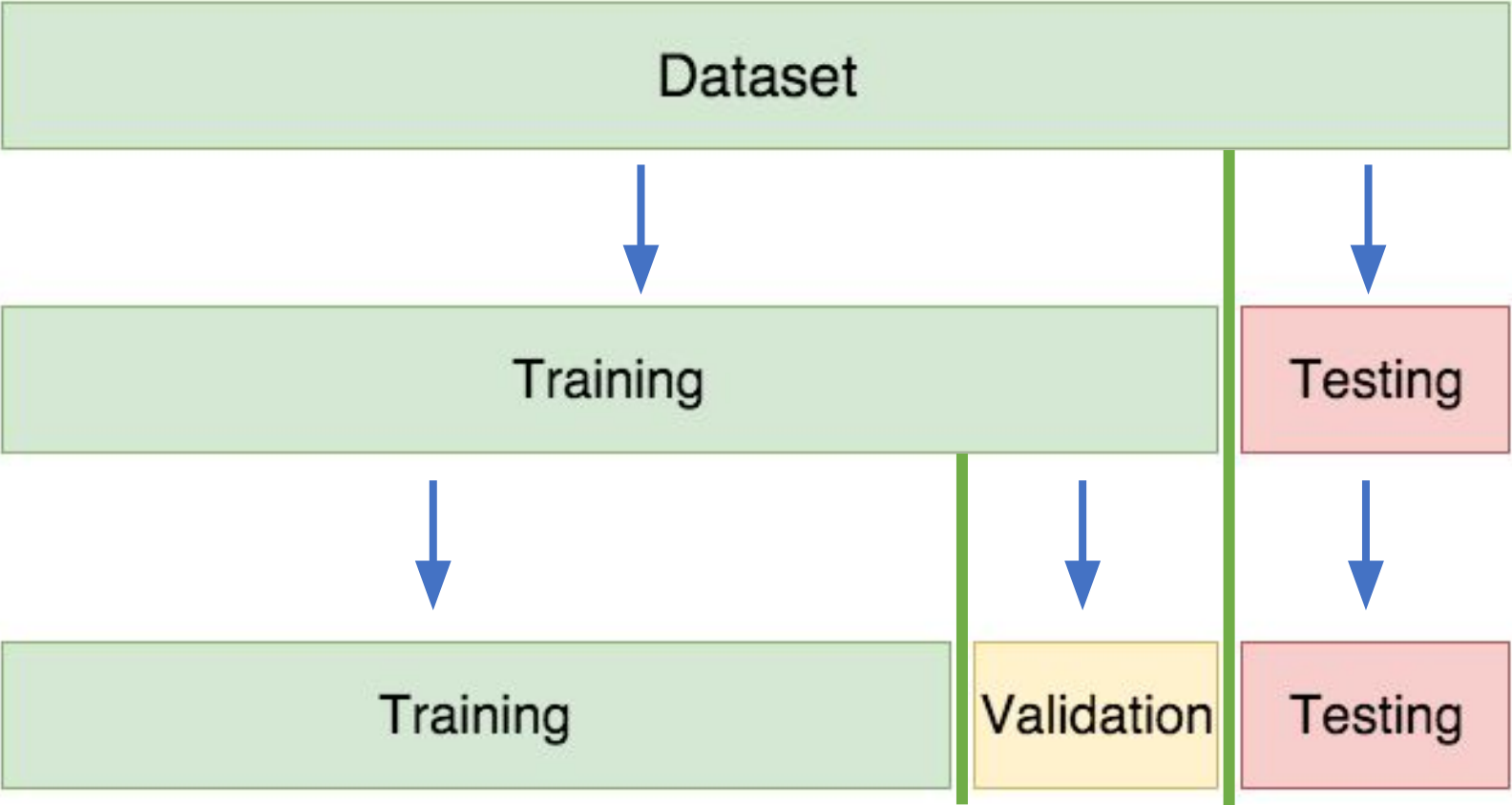




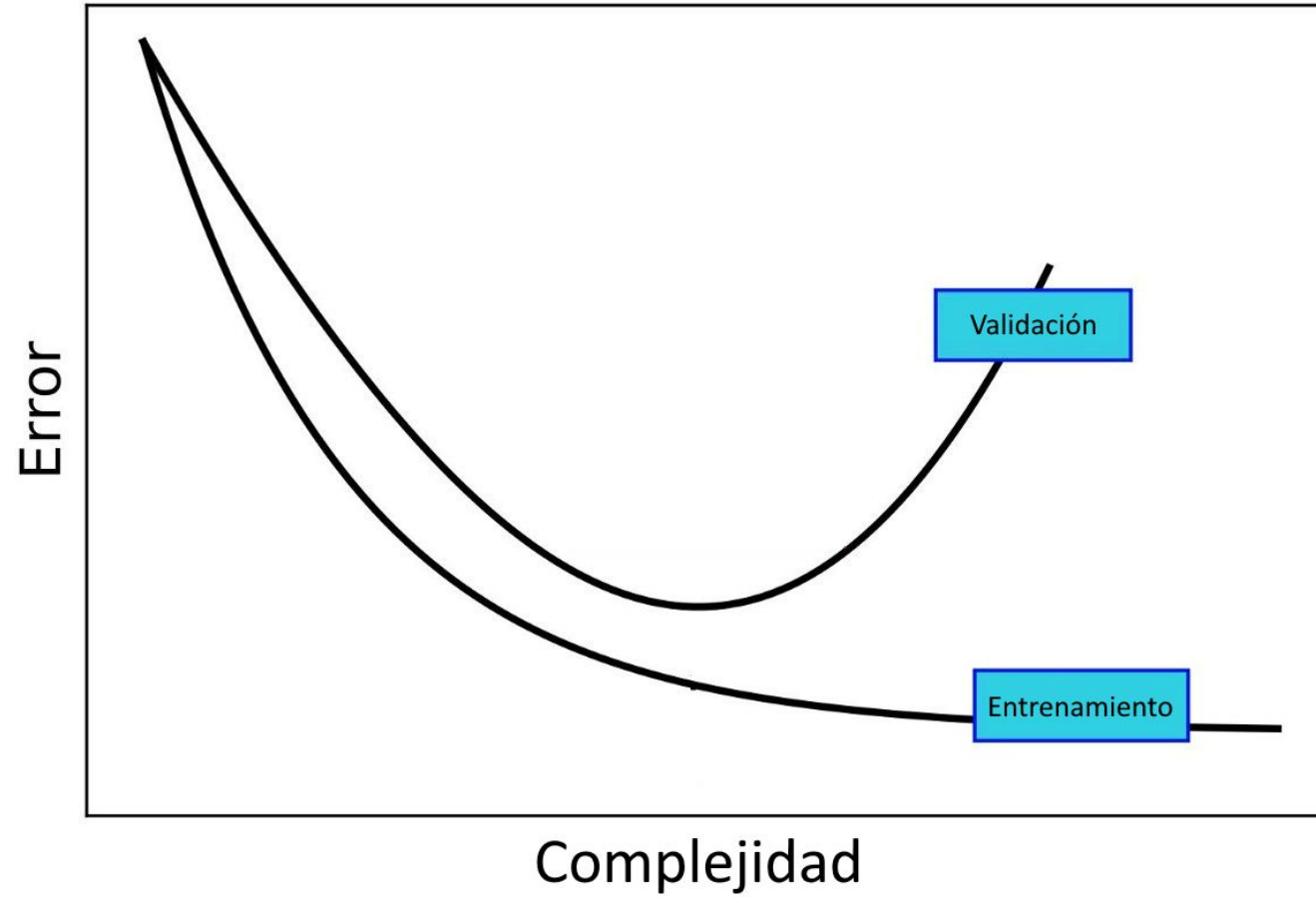
## Cómo podemos controlar esto

- Un mecanismo típico es utilizar un **set de validación** para evaluar el rendimiento.
- El set de validación es una pequeña parte del set de entrenamiento, que no se usa para entrenar inicialmente.
- Se entrenan distintos modelos en el nuevo set de entrenamiento y se evalúan en el de validación.

Una forma clara de ver esto es con conjuntos de datos disjuntos



En general, error en validación baja y luego sube



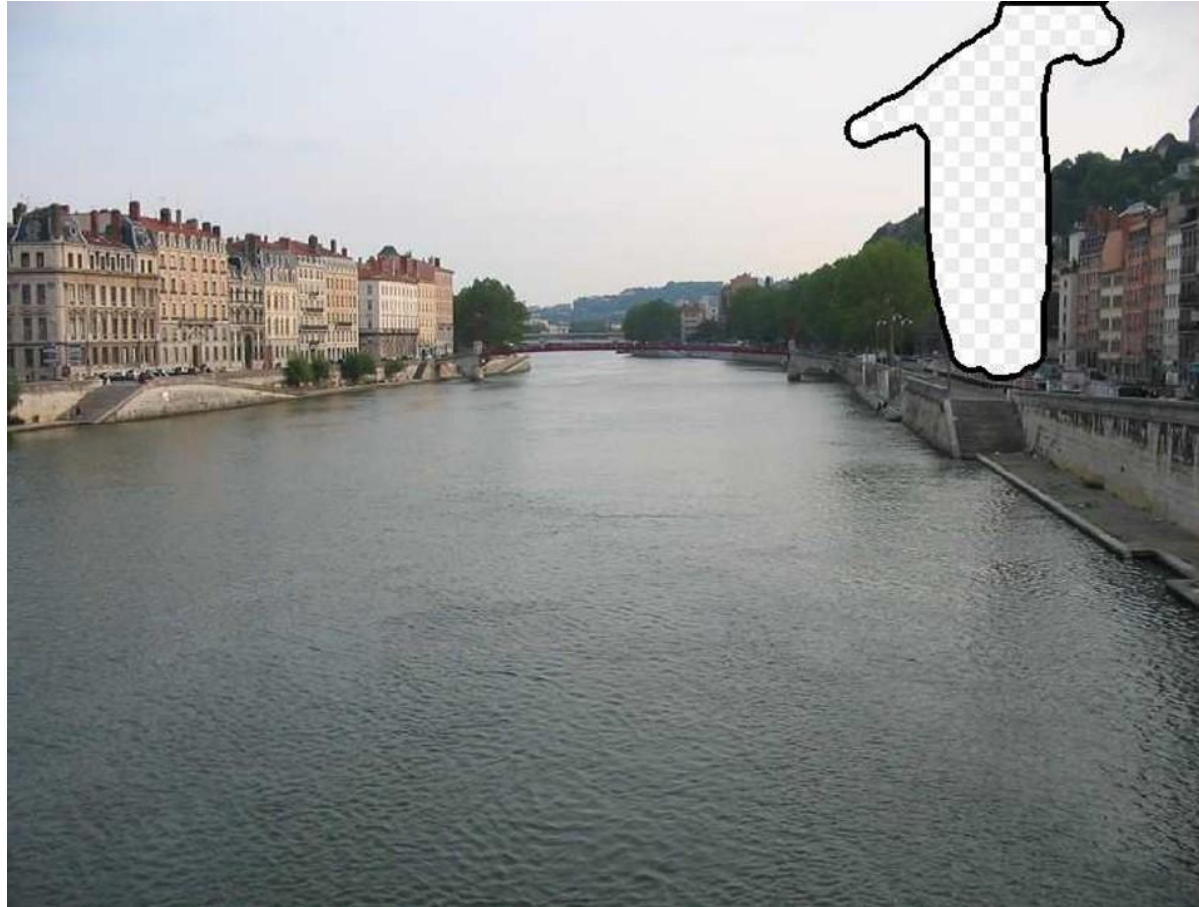
Cerremos con un caso de estudio más avanzado (e interesante)..

## Reconstrucción de Imágenes con IA



Cerremos con un caso de estudio más avanzado (e interesante)..

## Reconstrucción de Imágenes con IA

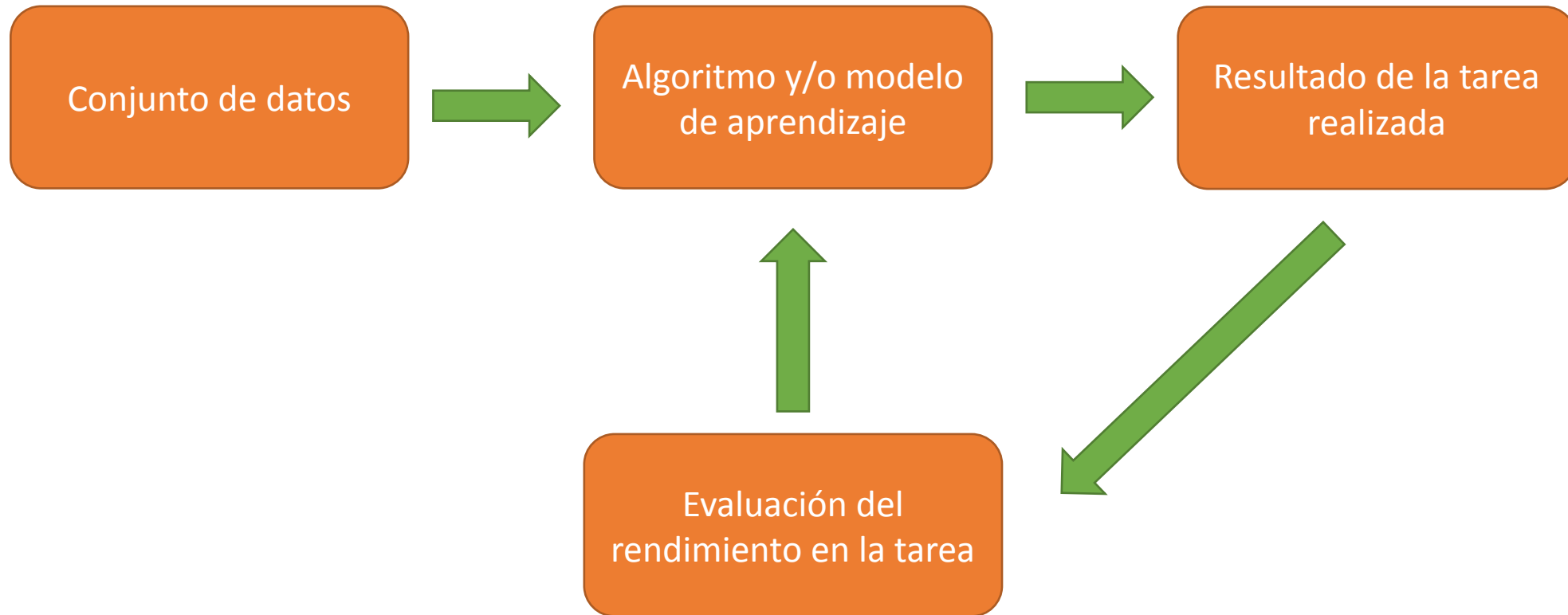


Cerremos con un caso de estudio más avanzado (e interesante)..

## Reconstrucción de Imágenes con IA

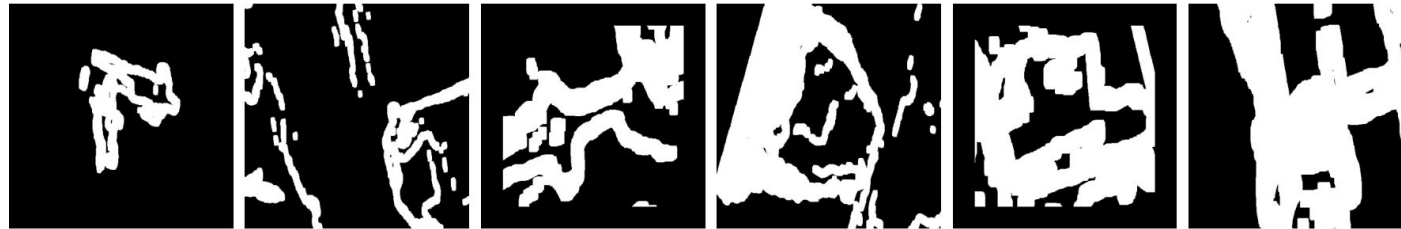


Recordemos que (casi) todas las técnicas de ML usan el mismo esquema de procesamiento

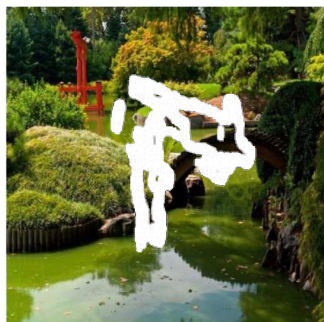




¿Qué es lo primero que necesitamos?







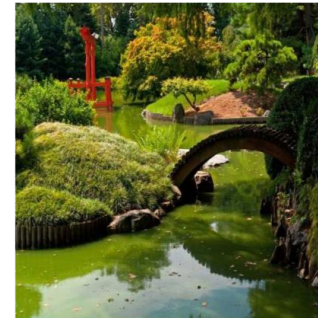
Algoritmo de  
aprendizaje  
supervisado



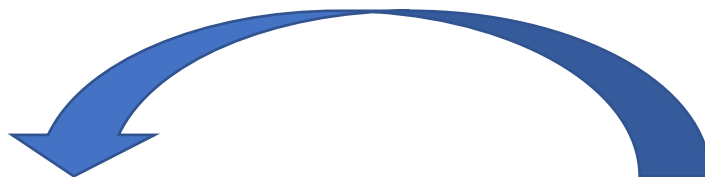
¿



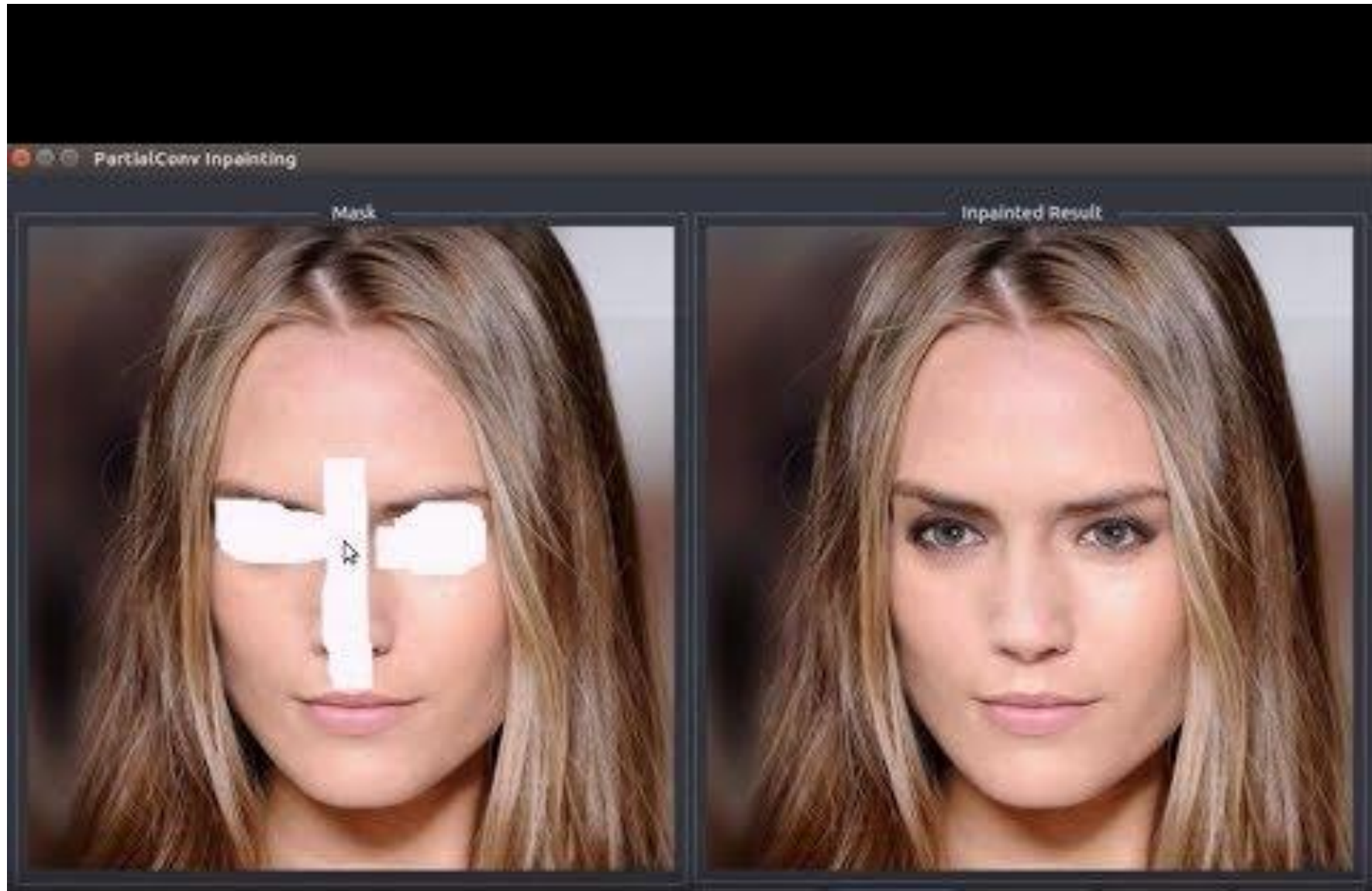
=



?



## Veamos cómo funciona el sistema en la práctica



## En resumen...

- ML se centra en algoritmos/modelos que aprenden de los datos para resolver una tarea
- scikit-learn permite hacer ML en Python de manera práctica y rápida
- (casi) Todas las técnicas de ML funcionan de la misma manera
- Qué técnica usar dependerá de la tarea y los datos disponibles, pero es posible elegir en base a métricas de rendimiento
- Todo esto y mucho más en el curso **ICT3115 – Sistemas Urbanos Inteligentes**

Pontificia Universidad Católica de Chile  
Escuela de Ingeniería  
Departamento de Ciencia de la Computación



# IIC2115 - Programación como Herramienta para la Ingeniería

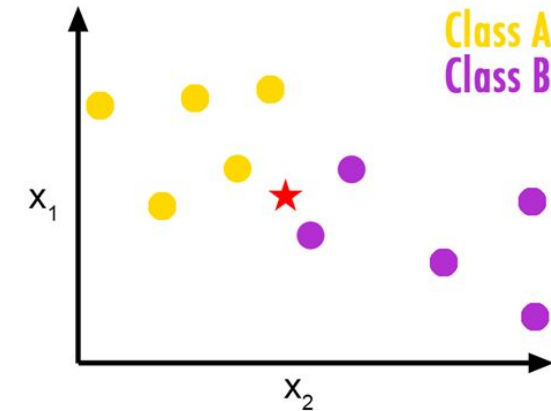
Modelos predictivos con Machine Learning

**Profesora:** Francesca Lucchini  
**Prof. Coordinador:** Hans Löbel

Veamos algunos modelos

# K-NN es la simpleza hecha algoritmo

- k-NN es el algoritmo más intuitivo y simple en ML.
- La inferencia sobre un nuevo ejemplo se basa directamente en la información de **ejemplos similares conocidos**.
- Se encuentra en el módulo **`sklearn.neighbors`**
- Para instanciarlo, utilizamos el siguiente comando:  
**`model = neighbors.KNeighborsClassifier()`**

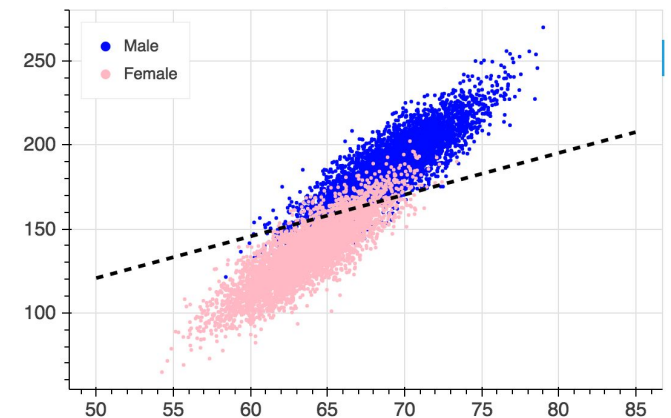
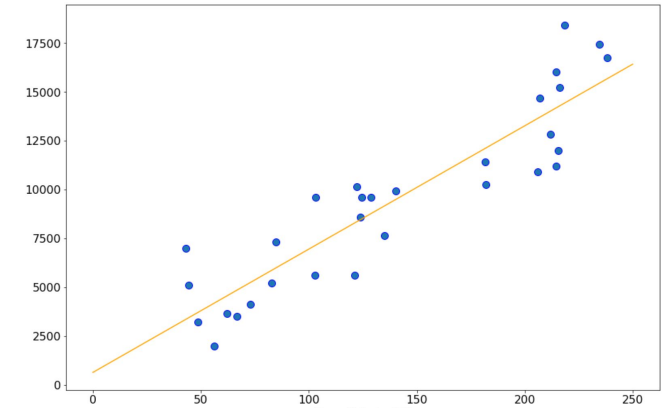


# Regresión lineal y logística

- Permiten estimar una función (reg. lineal) o clasificar (reg. logística) en base a una combinación lineal de las características.
- Ampliamente usadas en la práctica debido a su sencillez e interpretabilidad.
- Se encuentran en el módulo `sklearn.linear_model`
- Para instanciarlas, utilizamos los siguientes comandos:

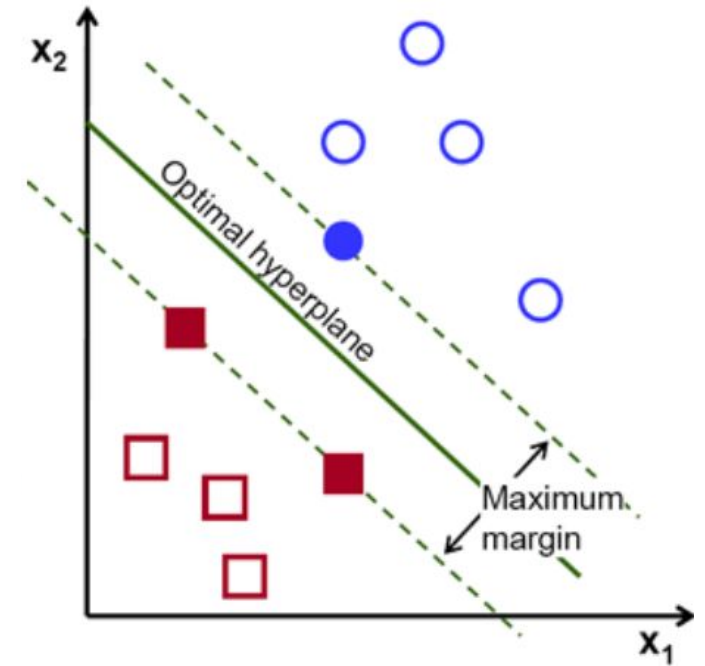
```
model = linear_model.LinearRegression()
```

```
model = linear_model.LogisticRegression()
```



# Support Vector Machine (SVM)

- Permite construir clasificadores que maximizan la distancia entre las clases.
- Excelente rendimiento y muy rápido de entrenar.
- Se encuentra en el módulo `sklearn.svm`
- Para instanciarlo, utilizamos el siguiente comando:  
`model = svm.SVC()`

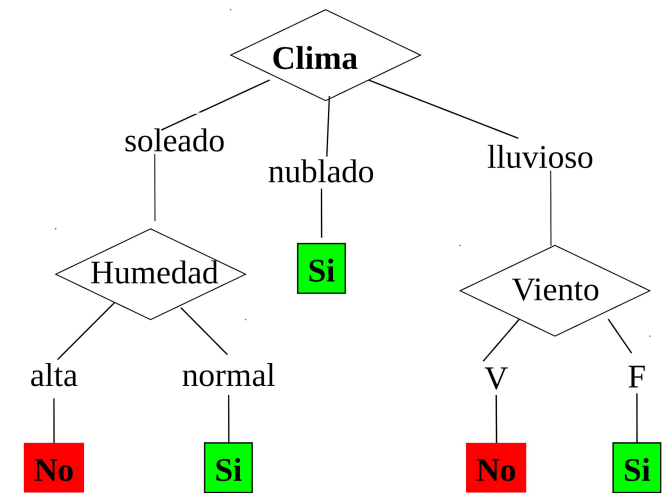




# Árboles de Decisión

- Técnica simple que funciona con cualquier tipo de dato.
- Construye una estructura de árbol en base a tests sobre las características.
- Rendimiento regular, pero altamente interpretable.
- Se encuentra en el módulo `sklearn.tree`
- Para instanciarlo, utilizamos el siguiente comando:

```
model = tree.DecisionTreeClassifier()
```

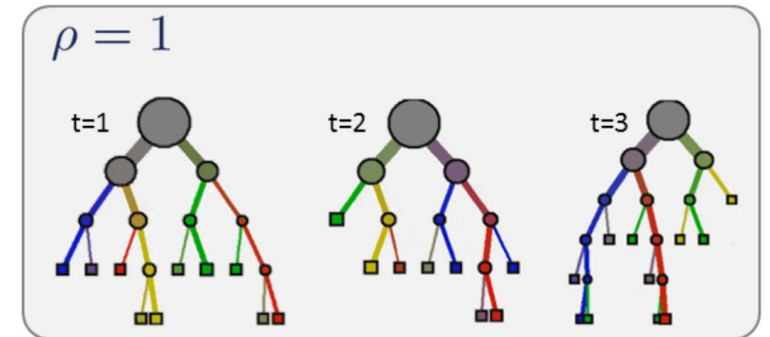


# Ensambles

- Técnicas que combinan múltiples clasificadores (generalmente árboles) para generar una predicción.
- Menor interpretabilidad que un árbol, pero obtienen rendimiento muy altos.
- Se encuentran en el módulo `sklearn.ensemble`
- Para instanciarlos, utilizamos los siguientes comandos:

```
model = ensemble.RandomForestClassifier()
```

```
model = ensemble.GradientBoostingClassifier()
```



# Red Neuronal

- Técnica altamente general y compleja para estimar funciones de todo tipo.
- Procesan los datos a través de varias capas, lo que les permite aprender cualquier cosa.
- En la actualidad, si se tienen muchos datos, son las que mejor funcionan.
- Se encuentran en el módulo `sklearn.neural_network`
- Para instanciarla, utilizamos el siguiente comando:

```
model = neural_network.MLPClassifier()
```

