

# Seminario PYTHON - MCAF (Sesión III)

Francisco Gárate Santiago

UCM, 2021

# Objetivos de la sesión

Probabilidad y estadística en Python:

- ▶ Introducción a la librería estadística SciPy.stats
- ▶ Manejo de las distribuciones de probabilidad más conocidas
- ▶ Modelos estadísticos.

Introducción al Machine Learning y a los principales algoritmos estadísticos de aprendizaje automático:

- ▶ Familiarizarse y conocer la librería de aprendizaje automático Scikit-learn.
- ▶ Conocer el análisis de componentes y estimación de un valor con `scikit-learn.LinearRegression`
- ▶ Introducción a la metodología básica del machine learning (model fitting, predicting, cross-validation, etc.)
- ▶ Overfitting y Cross-Validation: Técnica de entrenamiento y validación.

La librería **SciPy** (*Scientific computing*, pronunciado “Sigh Pie”) posee diferentes módulos (**stats**, **linalg**, **interpolate...**) orientados a diversas áreas científicas. De momento, el módulo/paquete que nos interesa para estadística es **scipy.stats**.

- ▶ Sitio: <https://www.scipy.org/scipylib/index.html>
- ▶ Documentacion:  
<https://docs.scipy.org/doc/scipy/reference/tutorial/stats.html>
- ▶ Licencia open-source propia.



- **scipy.stats** aporta algunas funciones que pueden complementar a NumPy:

```
1 import numpy as np
2 import scipy.stats as st
3 np.random.seed(999)
4 x = np.random.normal(size=1000)
5 print(np.median(x))
6 print(st.scoreatpercentile(x, 50)) # mediana
7 print(st.scoreatpercentile(x, 95))
```

```
0.007576375024717093
```

```
0.007576375024717093
```

```
1.6906947688452745
```

# Distribuciones de probabilidad

scipy.stats

Sin embargo, donde destaca **SciPy** es en el manejo de **distribuciones de probabilidad**:

- ▶ En estadística, toda variable aleatoria se distribuye en base a una función de probabilidad.
- ▶ Dependiendo del evento aleatorio, las distribuciones pueden ser **discretas** (toma ciertos valores aislados -puntos- dentro de un intervalo) o **continuas** (puede tomar cualquier valor real del rango).

# Distribuciones de probabilidad

scipy.stats

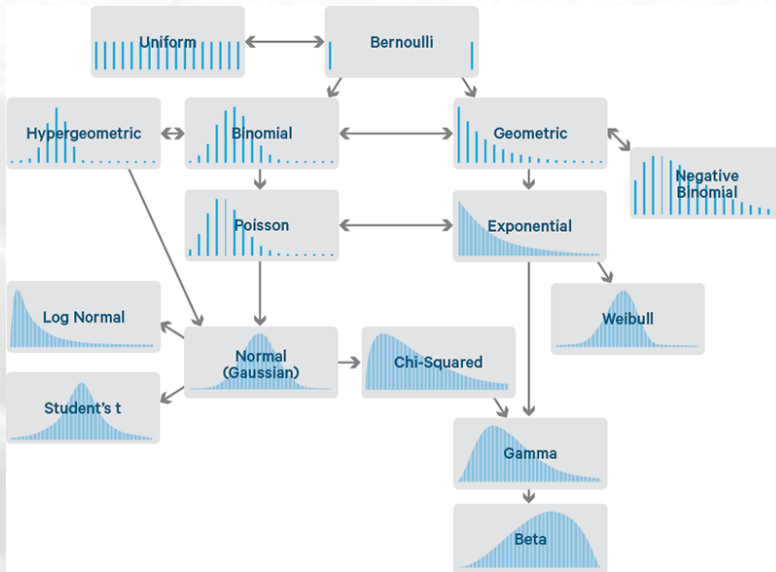
La versión 1.6 de SciPy tiene un catálogo de 88 distribuciones continuas y 15 discretas:

```
1 import scipy.stats as st
2 print(dir(st))
```

- ▶ **.norm()**: Normal (Gausiana)
- ▶ **.lognorm()**: Lognormal
- ▶ **.expon()**: Exponencial
- ▶ **.t()**: Student's t
- ▶ **.chi2()**: Chi-cuadrado
- ▶ **.gamma()**: Gamma
- ▶ **.beta()**: Beta
- ▶ **.bernoulli()**: Bernoulli
- ▶ **.binom()**: Binomial
- ▶ **.poisson()**: Poisson

<https://docs.scipy.org/doc/scipy/reference/tutorial/stats/continuous.html>

# Distribuciones de probabilidad



# Distribuciones de probabilidad

scipy.stats

Función de densidad:

- ▶ Cada distribución puede ilustrarse con su **función de densidad de probabilidad** (pdf), es decir, el eje horizontal es el conjunto de posibles resultados numéricos y el eje vertical describe la probabilidad de resultados.
- ▶ En la función de densidad de probabilidad, la **suma** de las alturas de las líneas (distrib. discretas) o las áreas bajo las curvas (distrib. continuas) son **siempre 1**.



# Distribuciones de probabilidad

scipy.stats

Tomaremos como ejemplo la distribución normal (gaussiana) como la función de distribución de nuestra variable a estudiar. La distr. normal es ampliamente usada en inferencia estadística, ya que con muestras grandes los errores *se aproximan bien* a una distribución normal.

La **función de densidad de probabilidad** normal para un número real  $x$  es:

$$f(x) = \frac{\exp(-x^2/2)}{\sqrt{2\pi}}$$

**norm.pdf(x, loc=0, scale=1)** para el valor  $x$ , siendo:

- **loc**: media de la distribución (por defecto 0)
- **scale**: desviación estándar de la distribución (por defecto 1).

```
1 import scipy.stats as st
2 a = st.norm.pdf(0.5)
3 print(a)
```

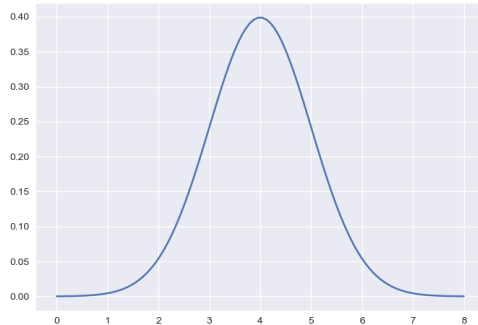
```
0.3520653267642995
```

# Distribuciones de probabilidad

scipy.stats

Ejemplo usando ***nombre\_distribución.pdf(x)*** con media 4 y desviación estándar 1:

```
1 import numpy as np
2 import scipy.stats as st
3 import matplotlib.pyplot as plt
4 gaussian = st.norm(loc=4.0, scale=1.0)
5 x = np.linspace(0.0, 8.0, 100)
6 y = gaussian.pdf(x)
7 plt.style.use('seaborn')
8 plt.plot(x, y)
9 plt.show()
```



# Distribuciones de probabilidad

scipy.stats

Adicionalmente, disponemos de la función de distribución acumulativa y la función de punto porcentual (la inversa de cdf):

- ▶ **.cdf(x, loc=0, scale=1)**: Función de distribución acumulativa (*Cumulative distribution function*)
- ▶ **.ppf(q, loc=0, scale=1)**: Función de punto porcentual (*Percent point function*) (la inversa de cdf, equivalente a los percentiles).

```
1 import scipy.stats as st
2 gaussian = st.norm(loc=4.0, scale=1.0)
3 a = st.norm.cdf(1.65)
4 b = st.norm.ppf(0.95) #percentil
5 print(a, b)
```

```
0.9505285319663519
```

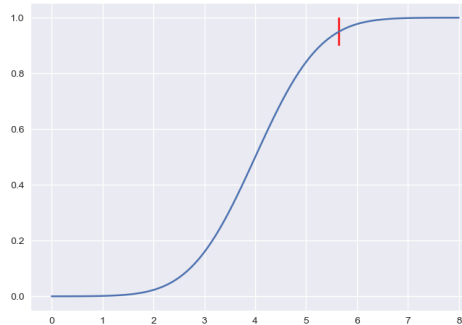
```
1.6448536269514722
```

# Distribuciones de probabilidad

scipy.stats

Ejemplo usando **.cdf(x)** y **.ppf(0.95)** para calcular el percentil 95

```
1 import numpy as np
2 import scipy.stats as st
3 import matplotlib.pyplot as plt
4 gaussian = st.norm(loc=4.0, scale=1.0)
5 x = np.linspace(0.0, 8.0, 100)
6 y = gaussian.cdf(x)
7 plt.style.use('seaborn')
8 plt.plot(x, y)
9 plt.vlines(gaussian.ppf(0.95), 0.9, 1, 'r')
10 plt.show()
```



# Distribuciones de probabilidad

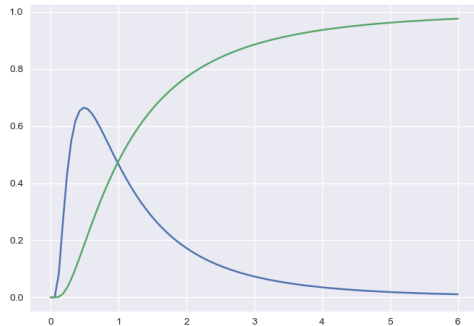
Ejemplo con la función de lognormalidad con **st.lognorm(s=sigma, scale=exp(mu))**

s = sigma std, mu = mean

sigma = std / la media de  $\ln(x)$

std = desviación estandar de  $\ln(x)$

```
1 import numpy as np
2 import scipy.stats as st
3 import matplotlib.pyplot as plt
4 # std normal
5 sigma = 0.859455801705594
6 # mean normal
7 mu = 0.418749176686875
8 lognormal = st.lognorm(s=sigma,
9                          scale=np.exp(mu))
9 x = np.linspace(0, 8.0, 100)
10 plt.style.use('seaborn')
11 plt.plot(x, lognormal.pdf(x))
12 plt.plot(x, lognormal.cdf(x))
13 plt.show()
```



# Distribuciones de probabilidad

**st.gamma(a=alpha, scale=1/lambda)**

a: alpha

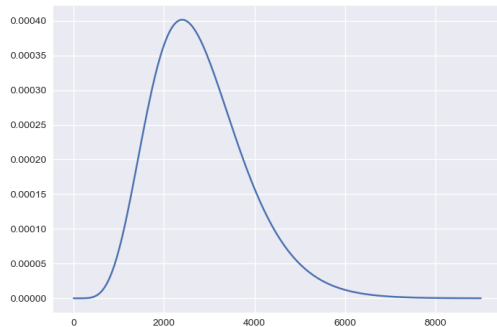
scale: theta = 1 / lambda (por defecto = 1)

**st.gamma.pdf(x, a, scale)**

x: valor para evaluar la función

*Cuando a es un entero, gamma se reduce a la distribución Erlang, y cuando a=1 a la distribución exponencial.*

```
1 import numpy as np
2 import scipy.stats as st
3 from matplotlib import pyplot as plt
4
5 alpha = 7.
6 lamdba = 0.0025
7 x = np.linspace(0, 9000, 200)
8 y = st.gamma.pdf(x, a=alpha, scale
9     =1/lamdba)
10
11 plt.style.use('seaborn')
12 plt.plot(x, y)
13 plt.show()
```



# Distribuciones de probabilidad

scipy.stats

**`mi_distribución.stats(loc=0, scale=1, moments='mvsk')`**: Devuelve los momentos que especifiquemos:

- ▶ **m**: Media (*mean*)
- ▶ **v**: Varianza (*variance*)
- ▶ **s**: Sesgo (*skew*)
- ▶ **k**: kurtosis

```
1 import scipy.stats as st
2 gaussian = st.norm(loc=4.0, scale=1.0) #mu=4 #sigma=1
3 mean, var, skew, kurt = gaussian.stats(moments='mvsk')
4 print(mean, var, skew, kurt)
```

```
4.0 1.0 0.0 0.0
```

# Modelos de Ajuste a una distribución

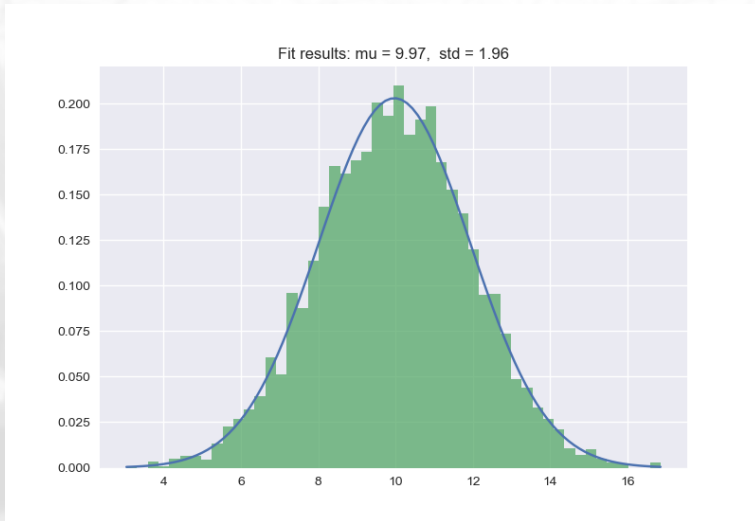
## SciPy.Stats

- ▶ Para ajustar datos a una distribución, es decir, determinar la distribución que sigue un vector de datos, todas las distribuciones continuas de **SciPy.Stats** disponen la función **.fit()**, que nos permite estimar parámetros partiendo de los datos disponibles.
- ▶ El método utilizado es el de máxima verosimilitud.
- ▶ El output de fit dependerá de los parámetros de la distribución a ajustar. Por ejemplo, en las siguientes dos distribuciones el output es distinto:
  - ▶ Normal: **st.norm.fit(data)**, devuelve media y std
  - ▶ Lognormal: **st.lognorm.fit(data)**, devuelve sigma, mu y std.
- ▶ Importante: **.fit()** no da ningún valor sobre la **bondad del ajuste**. Para ello veremos otras funciones.



# Modelos de Ajuste a una distribución

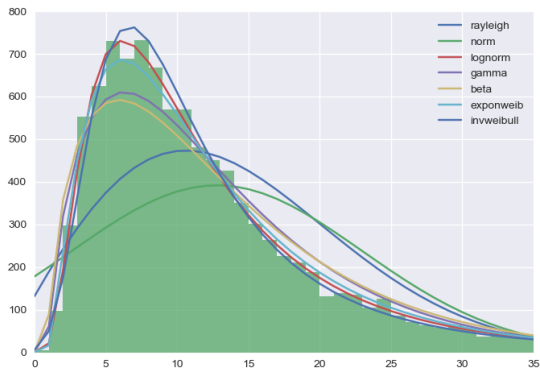
SciPy.Stats + matplotlib



# Modelos de Ajuste a una distribución

SciPy.Stats + matplotlib

Para un mismo set de datos puede realizarse para tantas distribuciones como queramos, y elegir la que mejor se adapte:



# Modelos de Ajuste a una distribución

SciPy.Stats

El código del anterior gráfico está disponible en el fichero **Ejemplo\_Ajuste\_distribucion.ipynb**.

A continuación, un extracto del mismo:

```
1 size = 10000
2 x = np.arange(size)
3 y = np.random.lognormal(np.log(10), np.log(2), size)
4
5 distrs = ['rayleigh', 'norm', 'lognorm', 'gamma', 'beta', 'exponweib', '
    invweibull']
6 for i in distrs:
7     dist = getattr(st, i)
8     param = dist.fit(y)
9     pdf_fitted = dist.pdf(x, *param[:-2], loc=param[-2], scale=param[-1])
        * size
10    plt.plot(pdf_fitted, label=i)
11    plt.xlim(0,25)
12 plt.hist(y, bins=range(50), alpha=0.75)
13 plt.show()
```

# Modelos estadísticos

## Modelos de ajustes a los datos

Muchos procedimientos estadísticos suponen que los datos siguen algún tipo de **modelo matemático que se define mediante una ecuación**, donde al desconocerse alguno de sus parámetros, pueden estimarse a partir de la información disponible.

Existen principalmente dos procedimientos para estimar los coeficientes de un modelo de regresión, o para estimar los parámetros de una distribución de probabilidad:

- ▶ **Mínimos cuadrados ordinarios (MCO)**, *ordinary least squares* (OLS) o mínimos cuadrados lineales son los nombres que recibe el método para encontrar los parámetros poblacionales en un modelo de regresión lineal obtenidos minimizando los residuos observados (la suma de los cuadrados de las diferencias entre la variable dependiente observada).
- ▶ **Estimación por máxima verosimilitud (EMV)**, *maximum likelihood estimation* (MLE), la cual requiere maximizar la función de verosimilitud, o función de probabilidad conjunta de los valores muestrales.

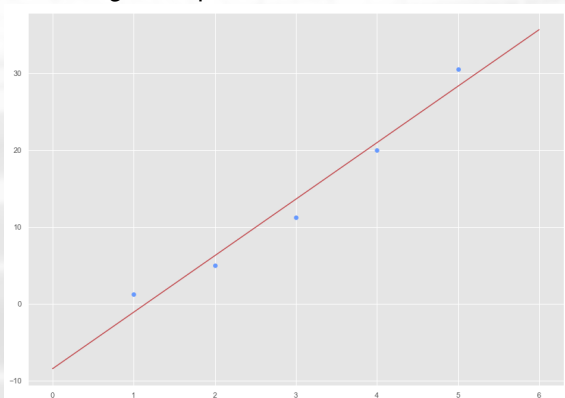
Bajo la hipótesis de distribución normal, las estimaciones por ambos criterios coinciden.

# Regresión lineal

## Mínimos cuadrados ordinarios

Gráfico de puntos y recta estimada de regresión por mínimos cuadrados:

	x	y
2014	1	1.25
2015	2	5
2016	3	11.25
2017	4	20
2018	5	30.5



```
1 df = pd.DataFrame(index=range(2014,2019))
2 df['x'] = [1, 2, 3, 4, 5]
3 df['y'] = [1.25, 5, 11.25, 20, 30.5]
```

# Regresión lineal

## Modelos de ajustes a los datos

En el modelo de regresión lineal, conocido también como modelo de regresión simple, o recta poblacional, es un modelo matemático usado para aproximar la relación de dependencia entre una variable dependiente  $y$ , las variables independientes  $x$  y un término aleatorio  $e$ :

$$y = \beta_1 + \beta_2 x + e$$

Siendo:

- ▶ **y**: la variable endógena o dependiente (variable explicada o regresando)
- ▶ **x**: la variable exógena o independiente (variable explicativa, regresor, covariable o variable de control)
- ▶ **e**: el error o perturbación aleatoria. Lo que no explica el modelo de ajuste.
- ▶  $\beta_1$ : es la ordenada al origen (indica el valor de  $y$  cuando  $x=0$ ).
- ▶  $\beta_2$  es la pendiente de la recta (cuanto cambia  $y$  por cada incremento de  $x$ ).

El objetivo principal del modelo de regresión es  $\beta_1$  y  $\beta_2$  (parámetros fijos y desconocidos) a partir de una muestra dada.

# Regresión lineal

## Mínimos cuadrados ordinarios

En base a una muestra de tamaño  $n$  es posible estimar los parámetros del modelo. Si una recta pasa exactamente por los puntos, el error será cero. El programa de minimización es el siguiente:

$$\min \sum_{i=1}^n (y_i - (\beta_0 + \beta_1 x_i))^2$$

En Python existen funciones específicas cuya finalidad es ajustar los datos tanto a una recta como a una distribución de probabilidad.

Así, para determinar la distribución que sigue un set de datos contamos en Python principalmente con dos librerías:

- ▶ Scipy: **scipy.stats**
- ▶ Statsmodels: **statsmodels.api**

# Regresión lineal

SciPy.stats

**SciPy.stats.linregress(x, y)** donde la salida devuelve los siguientes parámetros:

- ▶ **slope**: pendiente de la línea de regresión
- ▶ **intercept**: intercepción de la línea de regresión
- ▶ **r\_value**: coeficiente de correlación
- ▶ **p\_value**: p-value cuya hipótesis nula es que la pendiente es cero.
- ▶ **std err**: Error estándar de la estimación

```
1 import scipy.stats as st
2 x = (1., 2., 3., 4., 5.)
3 y = (1.25, 5, 11.25, 20, 30.5)
4 print(st.linregress(x,y))
```

```
LinregressResult(slope=7.35, intercept=-8.45, rvalue=0.983437109,
pvalue=0.00255244, stderr=0.782091213)
```



# Scikit-learn

## Machine learning

**Scikit-learn** es una biblioteca de aprendizaje automático (machine learning) de código abierto que soporta el aprendizaje supervisado y no supervisado. También proporciona varias herramientas para el ajuste de modelos, el preprocesamiento de datos, la selección y evaluación de modelos y muchas otras utilidades.

- ▶ Sitio: <https://scikit-learn.org/>
- ▶ Documentacion: [https://scikit-learn.org/stable/user\\_guide.html](https://scikit-learn.org/stable/user_guide.html)
- ▶ Licencia BSD open-source.



[Install](#) [User Guide](#) [API](#) [Examples](#) [More ▾](#) [Go](#)

# scikit-learn

Machine Learning in Python

[Getting Started](#)[Release Highlights for 0.24](#)[GitHub](#)

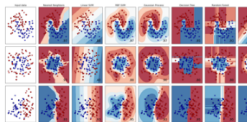
- Simple and efficient tools for predictive data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

## Classification

Identifying which category an object belongs to.

**Applications:** Spam detection, image recognition.

**Algorithms:** SVM, nearest neighbors, random forest, and more...

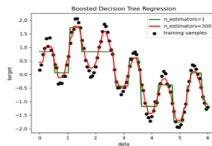
[Examples](#)

## Regression

Predicting a continuous-valued attribute associated with an object.

**Applications:** Drug response, Stock prices.

**Algorithms:** SVR, nearest neighbors, random forest, and more...

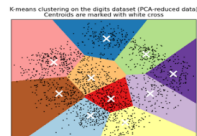
[Examples](#)

## Clustering

Automatic grouping of similar objects into sets.

**Applications:** Customer segmentation, Grouping experiment outcomes

**Algorithms:** k-Means, spectral clustering, mean-shift, and more...

[Examples](#)

## Dimensionality reduction

Reducing the number of random variables to consider

## Model selection

Comparing, validating and choosing parameters and models

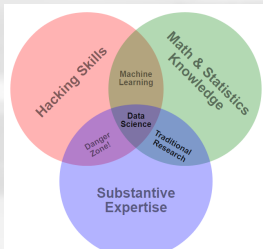
## Preprocessing

Feature extraction and normalization.

# Machine Learning

## Aprendizaje automático

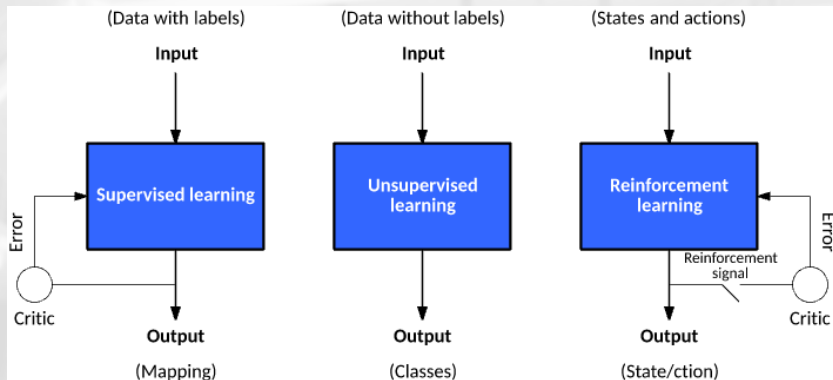
- ▶ El aprendizaje automático es un método de análisis de datos que automatiza la construcción de modelos analíticos.
- ▶ Es una rama de la inteligencia artificial basada en la idea de que los sistemas pueden aprender de los datos, identificar patrones y tomar decisiones con una mínima intervención humana.
- ▶ Su reciente éxito se debe a la posibilidad de utilización de recursos de hardware de bajo coste para resolver problemas estadísticos que hasta ahora dependían del poder del cerebro humano.
- ▶ En un primer momento, el Machine learning empezó llamándose “Statistical Learning” o estadística computacional.



# Machine Learning

## Aprendizaje automático

- Para entender el término de aprendizaje automático tenemos que hablar primero de sus categorías: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.



# Machine Learning

## Aprendizaje automático

x	y
2	4
3	9
4	16
8	?
9	81

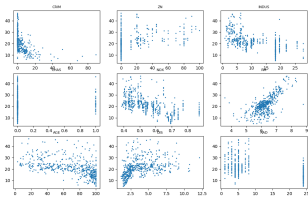
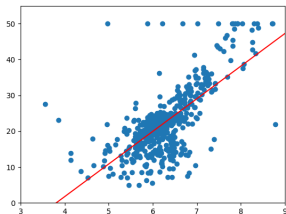
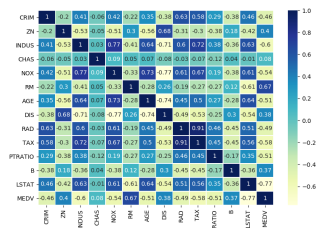
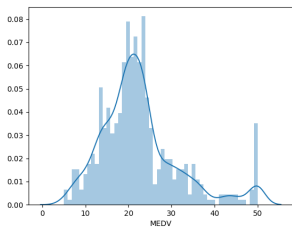
- ▶ ¿Cómo llega la máquina a devolver 64? La máquina resuelve la ecuación “aprendiendo de la experiencia”.
- ▶ Los algoritmos de aprendizaje automático utilizan métodos computacionales para “aprender” información directamente de los datos sin depender de una ecuación predeterminada como modelo.

## Ejemplo\_Boston\_LR\_GLM.ipynb

Veamos una aproximación al aprendizaje automático con el set de datos de Boston (boston.xlsx), en un primer lugar bajo un enfoque de modelos predictivos clásicos (regresión lineal y GLM, sin datos de entrenamiento ni test):

- ▶ Mostrar los datos a estimar (precio de la vivienda) a través de un histograma.
- ▶ Visualización de las correlaciones entre las variables.
- ▶ Realizar un análisis visual uni-variante
- ▶ Estimar el precio de la vivienda usando:
  - ▶ un modelo de regresión lineal con stasmodels: Hallar la recta de regresión entre la variable número de habitación (RM) y el valor de las viviendas (MEDV).
  - ▶ un modelo GLM con stasmodels utilizando el resto de variables cuantitativas.

## Ejemplo\_Boston\_LR\_GLM.ipynb - Gráficos:



- Comparamos los resultados:

Función de regresión	RMSE
Regresión lineal	7.17
GLM	5.99
Regresión de crestas (Ridge)	3.88

- La regresión Ridge posee con diferencia un menor error de previsión, y ofrece mejores resultados que las regresiones anteriores (lineal clásica y GLM).
- ¿Es posible bajar más el error? Si, scikit posee otras funciones con regresiones SGD (Stochastic Gradient Descent) o polinominales (varios grados de linealidad) muy eficaces que pueden reducir considerablemente error (en este ejemplo incluso con valores RMSE cercanos a 1), aunque no conseguiremos fórmulas de estimación muy fiables ya que existe el riesgo que caer sobreajustes o overfitting.
- Para validar dichas regresiones es útil la **validación cruzada**, es decir, entrenar el modelo con p.ej. el 75 % de los datos, y ver cómo de fiable es estimando el 25 % restante (muestra de contraste), técnica conocida como Leave-One-Out (Dejar uno fuera) o Cross-Validation.



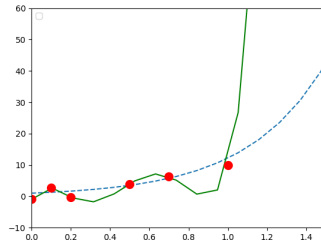
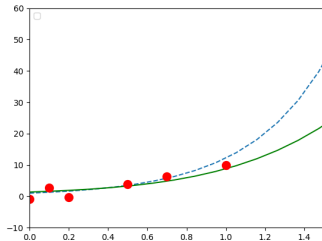
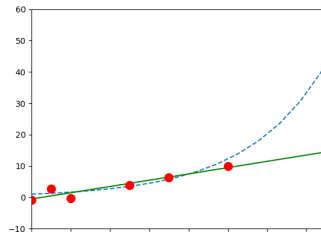
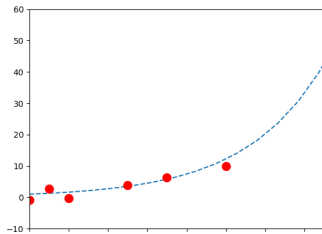
## Sobreajuste: overfitting

Veamos con otro ejemplo lo que significa el **overfitting** usando solo una variable explicativa para poder visualizarlo en una gráfica 2D.

- ▶ Seleccionamos una serie de puntos (en rojo) creados con una función exponencial ( $y = e^{2,5x}$ ). Los puntos se calculan sobre esta curva (se muestra en todas las gráficas en línea azul discontinua) añadiéndose algo de ruido.
- ▶ Es decir, los puntos rojos equivaldrían a los puntos de entrenamiento del modelo, y la curva en azul discontinua sería la regla exacta que rige los datos (fórmula exponencial).
- ▶ Se estiman diferentes modelos bajo una regresión lineal (2), una regresión Ridge (3) y una regresión polinomial (4). Los residuos, aunque no se muestran, pueden apreciarse visualmente.
- ▶ El código de este ejemplo puede encontrarse en el fichero `Ejemplo_overfitting.ipynb`
- ▶ **Conclusión:** Mejores residuos no significa mejor modelo. Los modelos deben validarse con datos de entrenamiento y de test (validación cruzada).

# Sobreajuste: overfitting

- (1) Datos en rojo y la fórmula exponencial para su creación en azul.  
(2) Modelo de regresión lineal (3) Ridge regression (4) Polynomial regression



# Cross-Validation

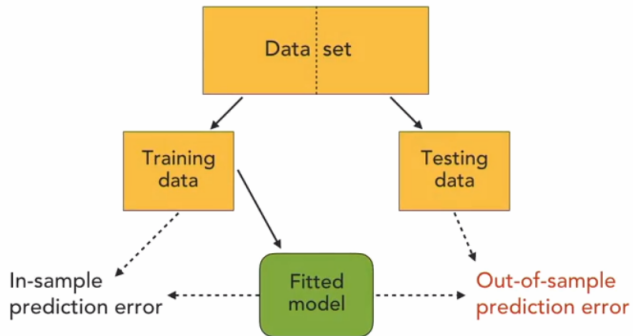
## Validación cruzada

- ▶ La **validación cruzada** es el método estadístico utilizado en los modelos de *machine learning* para medir su capacidad predictiva.
  - ▶ Se divide los datos aleatoriamente en dos grupos: **datos de entrenamiento** y **datos de test**.
  - ▶ Ajusta un modelo en el conjunto de entrenamiento y lo evalúa en el conjunto de test.
  - ▶ Se obtiene el error de previsión para el modelo estimado (p.ej: MSE o Error cuadrático medio) y se compara con otros posibles modelos.
- ▶ Según el tipo de modelo y los datos disponibles, es practica habitual dividir 75 %-25 % nuestros datos (aunque algún autor trabaja con 50 %-50 %).
- ▶ Hay muchas formas de dividir el conjunto de datos inicial en dos partes (training, testing). Una posibilidad es extraer una muestra que forma el conjunto de training y colocarla en el conjunto de pruebas (testing). A esto se le llama validación cruzada (Leave-One-Out o Dejar uno fuera).
- ▶ Con N muestras, obtenemos N sets de training y pruebas.

# Cross-Validation

Validación cruzada

## Cross Validation



# Cross-Validation

## Crear datos de entrenamiento

- ▶ En **Scikit-Learn** se dispone de la función `.model_selection.train_test_split(test_size)` (por defecto `test_size=0.25`)

- ▶ Ejemplo de división con scikit-learn:

```
1 from sklearn.model_selection import train_test_split
2 data = pd.read_csv('datos.csv')
3 train, test = train_test_split(data)
```

- ▶ Ejemplo de división con pandas:

```
1 import pandas as pd
2 df = pd.read_csv('desempleo.csv', sep=';')
3 sampling = df.sample(len(df))
4
5 training = sampling.iloc[:len(df)//2]
6 testing = sampling.iloc[len(df)//2:]
```

# Cross-Validation

## Ejemplo

```
1 from sklearn.model_selection import train_test_split
2 from sklearn import datasets, linear_model
3 data = datasets.load_boston()
4 X_train, X_test, y_train, y_test = train_test_split(data.data, data.
    target)
5
6 lm = linear_model.LinearRegression()
7 model = lm.fit(X_train, y_train)
8 predictions = lm.predict(X_test)
9
10 print(lm.score(X_train, y_train))
11 print(lm.score(X_test, y_test))
12 #print(lm.coef_, lm.intercept_)
```

```
0.7513466212833374
0.7040640680312542
```

## Ejemplo\_Boston\_ML\_CV.ipynb

Con los mismos datos del ejercicio 1, el set de datos de Boston (BostonHousing.csv), se pide estimar el precio de la vivienda aplicando el enfoque propio de machine learning:

- ▶ Dividir datos en datos en entrenamiento y test.
- ▶ Entrenar el modelo utilizando las siguientes funciones de regresión<sup>1</sup>:
  - ▶ LinearRegression()
  - ▶ RidgeCV()
  - ▶ SGDRegressor()
  - ▶ LassoCV()
- ▶ Validar el ajuste con los datos de test.

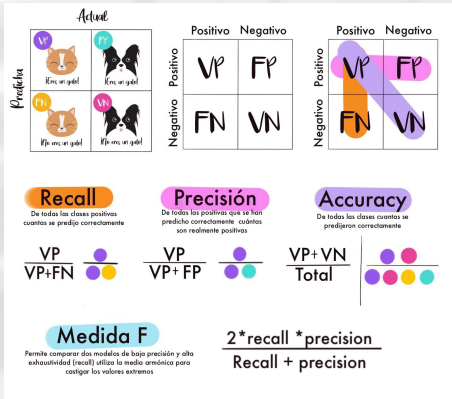
*(Es decir, se pide replicar el ejemplo anterior modificando la función de regresión, y comparar los resultados).*

---

<sup>1</sup>[https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear\\_model](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)

# Matriz de confusión

- ▶ La **matriz de confusión** permite la visualización del desempeño de un algoritmo que se emplea en aprendizaje supervisado.
- ▶ Se compara lo observado con la predicción del modelo para concluir que algoritmo que logra más aciertos.
- ▶ los errores pueden ponderarse de manera distinta, si van en uno u otro sentido.





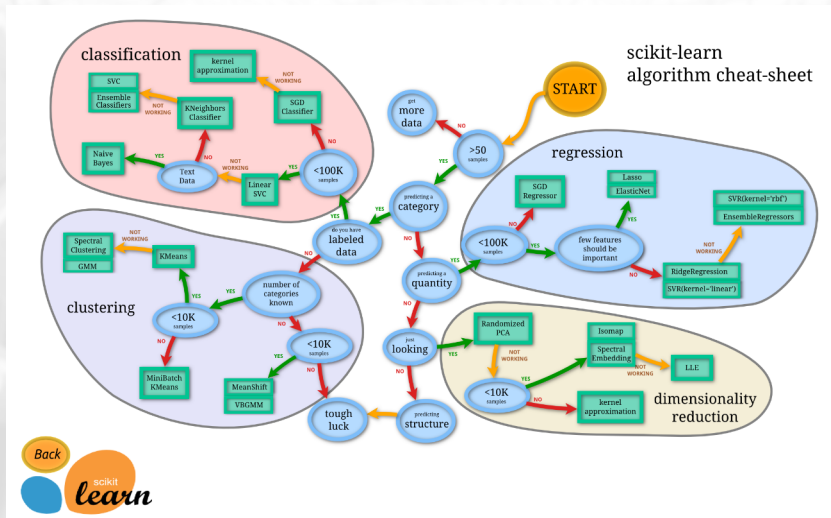
# ¿Que algoritmo elegir?

Supervisado, no supervisado o reforzado

- ▶ Los **métodos** del machine learning se clasifican en tres grupos:
  - ▶ Aprendizaje supervisado (Clasificación o Regresión)
  - ▶ Aprendizaje no supervisado
  - ▶ Aprendizaje reforzado.
- ▶ La elección del algoritmo no es fácil.
- ▶ Para acotar las técnicas disponibles, Scikit-learn y Microsoft Azure ML han diseñado sus propios **árboles de decisión**.

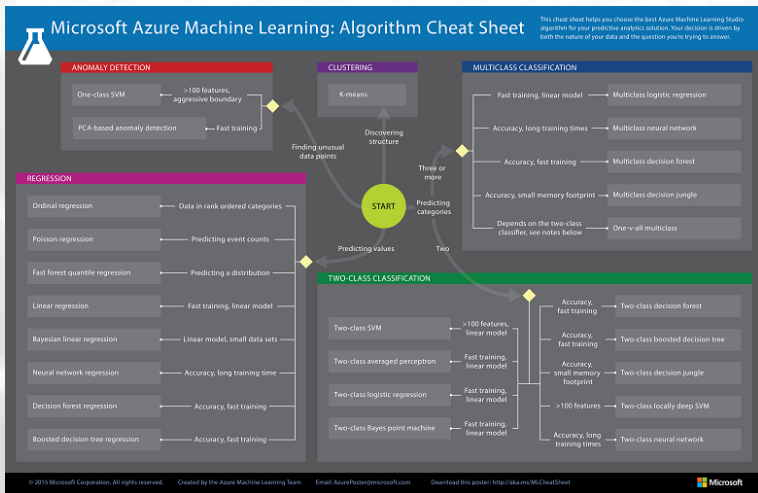
# Scikit-learn: árbol de decisión

[https://scikit-learn.org/stable/tutorial/machine\\_learning\\_map/](https://scikit-learn.org/stable/tutorial/machine_learning_map/)



# Microsoft Azure Machine Learning: árbol de decisión

<https://docs.microsoft.com/en-us/azure/machine-learning/studio/algorithm-cheat-sheet>



# Aprendizaje supervisado

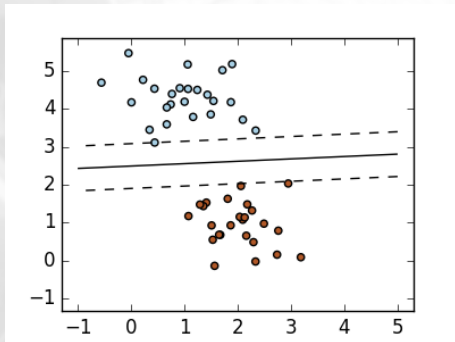
## El **aprendizaje supervisado**:

- ▶ Encuentra patrones (y desarrolla modelos predictivos) usando tanto datos de entrada como de salida.
- ▶ Los algoritmos trabajan con datos “etiquetados”: input - algoritmo - output predeterminado
- ▶ Todas las técnicas de Aprendizaje Supervisado son formas de Clasificación o de Regresión.
- ▶ Clasificación: Asignar categoría, detección de fraude, etc..
- ▶ Regresión: Las tareas de regresión se utilizan cuando lo que se quiere averiguar es un valor numérico de una variable continua. Ejemplo: el precio de activos financieros.

AVISO: No deberíamos realizar clasificación pura (asignando etiquetas Verdadero o Falso), sino proporcionar en la medida de lo posible probabilidad y, si fuera viable, distribuciones de probabilidad.

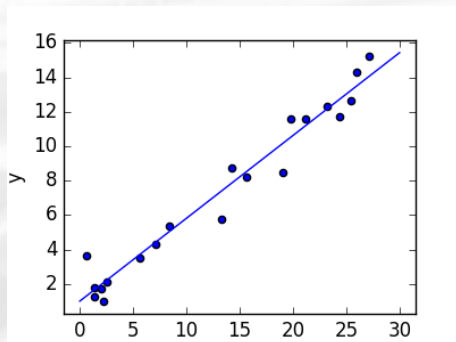
# Aprendizaje supervisado

Clasificación:



Ejemplo típico: Iris dataset de Fischer

Regresión:



Ejemplo típico: Boston House Prices

# Regresiones en Machine Learning

## Sckit-learn

- ▶ Las técnicas tanto estadísticas como computacionales en que se basa el Machine Learning han puesto fácilmente al alcance de cualquiera el uso de una familia de regresiones avanzadas. Las más usadas:
  - ▶ **Im.LinearRegression()**: admite como parámetro de la función **.predict()** el grado de la ecuación parabólica a ajustar.
  - ▶ **Im.Ridge()**: Ridge Regression o regularización de Tikhonov. Este estimador tiene soporte incorporado para regresión multivariante.
  - ▶ **Im.Lasso()**: Modelo lineal de Lasso con ajuste iterativo con regularización.
  - ▶ **Im.SGDRegressor()**: Stochastic Gradient Descent.
- ▶ Todas las funciones de sklearn de regresiones acabadas en CV, es que admiten *Cross-Validation*.
- ▶ En la documentación de scikit-learn se pueden consultar todas las posibilidades:  
[https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear\\_model](https://scikit-learn.org/stable/modules/classes.html#module-sklearn.linear_model)

# Regresiones múltiples

sklearn.linear\_model

**sklearn.linear\_model.LinearRegression()** calcula un modelo de regresión lineal basado en la estimación por mínimos cuadrados (OLS). Scikit-learn, en cambio, infiere la formula de la regresión múltiple, que pueden ser consultados con **.predict()**.

- ▶ **lm.fit()**: Ajuste a un modelo lineal
- ▶ **lm.predict()**: Predice Y usando el modelo lineal con coeficientes estimados
- ▶ **lm.score()**: Devuelve el coeficiente de determinación ( $R^2$ ): como de bien los resultados observados son replicados por el modelo, como la proporción de la variación total de los resultados explicada por el modelo.

Para obtener la formula de la regresión:

- ▶ **lm.coef\_**: Parametros de la regresión
- ▶ **lm.intercept\_**: Valor independiente

# Aprendizaje no supervisado

## El aprendizaje no supervisado:

- ▶ Encuentra patrones basados solo en datos de entrada. Es decir, no se dispone de datos etiquetados para el entrenamiento, solo conocemos los datos de entrada pero no existen datos de salida.
- ▶ Esta técnica puede ser útil cuando no estás seguro de qué buscar.
- ▶ A menudo se utiliza para el análisis exploratorio de datos en bruto.
- ▶ La mayoría de las técnicas de aprendizaje no supervisado son una forma de análisis de cluster (*agrupamiento o escasillamiento* basado en similitudes):
  - ▶ Clustering: agrupar individuos por su similitud o co-ocurrencia.
  - ▶ Modelado Causal: detectar la influencia de unos hechos sobre otros
- ▶ Un estudio puede abordarse con varios algoritmos, por ejemplo, clustering o regresión.



# Aprendizaje no supervisado

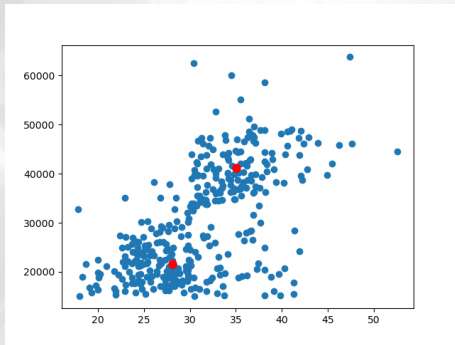
## K-medias o *k*-means clustering

- ▶ K-medias o **k-means clustering** es un método de cuantificación de vectores, que tiene por objeto dividir  $n$  observaciones en  $k$  clusters (celdas de Voronoi) en los que cada observación pertenece al cluster con la media más cercana (centros de cluster o centroides).
- ▶ Se utiliza la **Curva de Elbow** para la selección del número óptimo de clusters. El objetivo es buscar el  $n$  óptimo de celdas para evitar sobreajuste del modelo:

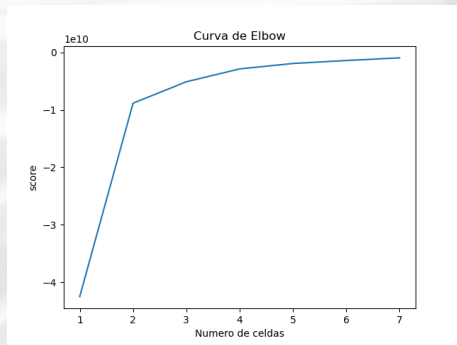
# Aprendizaje no supervisado

## K-medias o $k$ -means clustering

K-means:



Curva de Elbow:



Ver ejemplo en el fichero: **Ejemplo\_kmeans.py**