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

IIC2115 – Programación como herramienta para la Ingeniería

Capítulo 4 – Análisis Exploratorio de Datos

Profesores: Hans Löbel
Francisco Garrido

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

- Consiste principalmente en utilizar librerías para:
  - Limpiar y transformar los datos
  - Explorar distintas dimensiones de los datos
  - Calcular estadísticas de los datos
  - Visualizar los datos
  - Construir modelos predictivos preliminares

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

- Consiste principalmente en utilizar librerías para:
  - Limpiar y transformar los datos
  - Explorar distintas dimensiones de los datos
  - Calcular estadísticas de los datos
  - Visualizar los datos
  - Construir modelos predictivos preliminares
- Para todo esto (y más), está Pandas y scikit-learn

Introducción a Pandas

- Permite manipular, analizar y visualizar datos.
- Puede ser vista como una herramienta para trabajar datos almacenados en una estructura de tabla o de serie de tiempo.
- Se basa en, y generaliza a, la librería Numpy.
- 2 Estructuras principales
  - Series
  - DataFrame

DataFrame

	Comuna	Manzana	Predial	Línea de construcción	Material estructural	Calidad construcción	Año construcción
0	9201	1	1	1	E	4	1940
1	9201	1	1	2	E	4	1960
2	9201	1	2	1	E	4	1930
3	9201	1	3	1	E	4	1960
4	9201	1	4	1	E	3	1925

```python
import pandas as pd
import numpy as np


df = pd.read_csv("train.csv")
```

```python
display(df.describe())
```

	ApplicantIncome	CoapplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History
count	614.000000	614.000000	592.000000	600.00000	564.000000
mean	5403.459283	1621.245798	146.412162	342.00000	0.842199
std	6109.041673	2926.248369	85.587325	65.12041	0.364878
min	150.000000	0.000000	9.000000	12.00000	0.000000
25%	2877.500000	0.000000	100.000000	360.00000	1.000000
50%	3812.500000	1188.500000	128.000000	360.00000	1.000000
75%	5795.000000	2297.250000	168.000000	360.00000	1.000000
max	81000.000000	41667.000000	700.000000	480.00000	1.000000

```python
df['Property_Area'].value_counts()
```

```
Semiurban    233
Urban        202
Rural        179
Name: Property_Area, dtype: int64
```

```python
def conteo_nulo(x):
    return sum(x.isnull())

df.apply(conteo_nulo, axis = 0)
```

```
Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount          22
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

```python
df['LoanAmount'].fillna(df['LoanAmount'].mean(), inplace=True)
```

```python
df.apply(conteo_nulo, axis = 0)
```

```
Loan_ID              0
Gender              13
Married              3
Dependents          15
Education            0
Self_Employed       32
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term    14
Credit_History      50
Property_Area        0
Loan_Status          0
dtype: int64
```

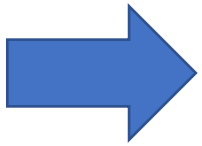
Otro problema típico al explorar son las múltiples fuentes

- Cuando todo está en un DataFrame, la cosa fluye…
- Pero la mayoría de las veces, tenemos más de uno
- Pandas entrega varios mecanismos para enfrentar esto

```python
def make_df(cols, ind):
    data = {c: [str(c) + str(i) for i in ind] for c in cols}
    return pd.DataFrame(data, ind)
```

```python
make_df('ABC', range(3))
```

	A	B	C
0	A0	B0	C0
1	A1	B1	C1
2	A2	B2	C2



```python
df1 = make_df('AB', [1, 2])
df2 = make_df('AB', [3, 4])
dfc = pd.concat([df1, df2], axis=1,)
display(df1, df2, dfc)
```

	A	B
1	A1	B1
2	A2	B2

	A	B
3	A3	B3
4	A4	B4

	A	B	A	B
1	A1	B1	NaN	NaN
2	A2	B2	NaN	NaN
3	NaN	NaN	A3	B3
4	NaN	NaN	A4	B4

Otro problema típico al explorar son las múltiples fuentes

```
df1 = pd.DataFrame({'employee': ['Bob', 'Jake', 'Lisa', 'Sue'],
                    'group': ['Accounting', 'Engineering', 'Engineering', 'HR']})
df2 = pd.DataFrame({'employee': ['Lisa', 'Bob', 'Jake', 'Sue'],
                    'hire_date': [2004, 2008, 2012, 2014]})
display(df1,df2)
```

	employee	group
0	Bob	Accounting
1	Jake	Engineering
2	Lisa	Engineering
3	Sue	HR

	employee	hire_date
0	Lisa	2004
1	Bob	2008
2	Jake	2012
3	Sue	2014

```
df3 = pd.merge(df1, df2)
df3
```

	employee	group	hire_date
0	Bob	Accounting	2008
1	Jake	Engineering	2012
2	Lisa	Engineering	2004
3	Sue	HR	2014

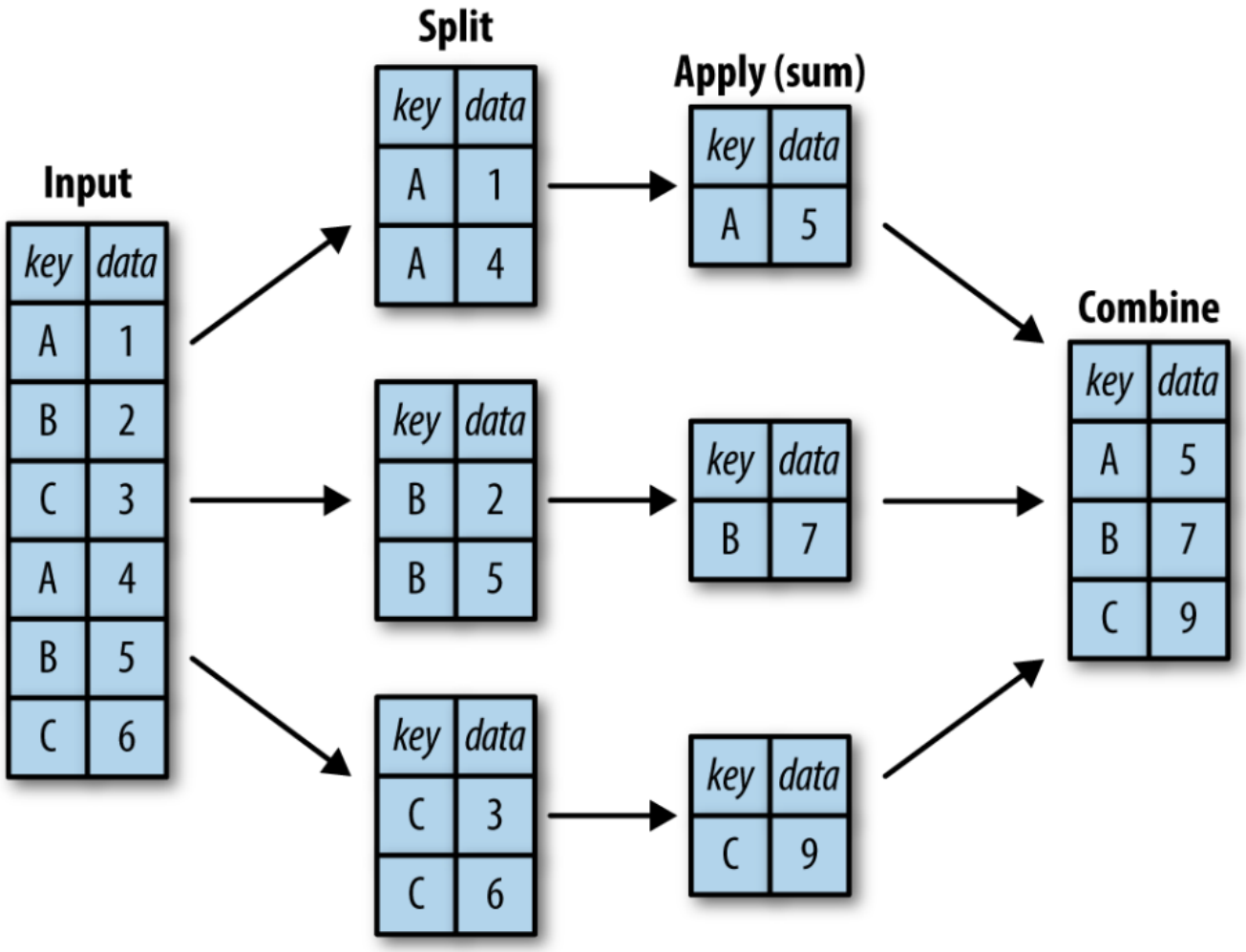
Agregación es la más común de las tareas exploratorias

- Analizar tendencias o buscar patrones se hace difícil si el análisis es individual
- Para evitar esto, datos generalmente se analizan de manera agregada
- Además de esto, la agregación suele ser a nivel grupal y no global
- Pandas permite enfrentar estos problemas con una serie de mecanismos que facilitan la exploración

Agregación es la más común de las tareas exploratorias

Aggregation	Description
`count()`	Total number of items
`first()`, `last()`	First and last item
`mean()`, `median()`	Mean and median
`min()`, `max()`	Minimum and maximum
`std()`, `var()`	Standard deviation and variance
`mad()`	Mean absolute deviation
`prod()`	Product of all items
`sum()`	Sum of all items

Función groupby permite combinar todo



```
import seaborn as sns
planets = sns.load_dataset('planets')
planets.head()
```

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

```
planets.dropna().describe()
```

	number	orbital_period	mass	distance	year
count	498.00000	498.000000	498.000000	498.000000	498.000000
mean	1.73494	835.778671	2.509320	52.068213	2007.377510
std	1.17572	1469.128259	3.636274	46.596041	4.167284
min	1.00000	1.328300	0.003600	1.350000	1989.000000
25%	1.00000	38.272250	0.212500	24.497500	2005.000000
50%	1.00000	357.000000	1.245000	39.940000	2009.000000
75%	2.00000	999.600000	2.867500	59.332500	2011.000000
max	6.00000	17337.500000	25.000000	354.000000	2014.000000

```
1 planets.groupby('method')['orbital_period'].median()
```

```
method
Astrometry                          631.180000
Eclipse Timing Variations          4343.500000
Imaging                           27500.000000
Microlensing                       3300.000000
Orbital Brightness Modulation         0.342887
Pulsar Timing                        66.541900
Pulsation Timing Variations        1170.000000
Radial Velocity                     360.200000
Transit                               5.714932
Transit Timing Variations            57.011000
Name: orbital_period, dtype: float64
```

```
1 planets.groupby('method')['year'].describe()
```

method	count	mean	std	min	25%	50%	75%	max
Astrometry	2.0	2011.500000	2.121320	2010.0	2010.75	2011.5	2012.25	2013.0
Eclipse Timing Variations	9.0	2010.000000	1.414214	2008.0	2009.00	2010.0	2011.00	2012.0
Imaging	38.0	2009.131579	2.781901	2004.0	2008.00	2009.0	2011.00	2013.0
Microlensing	23.0	2009.782609	2.859697	2004.0	2008.00	2010.0	2012.00	2013.0
Orbital Brightness Modulation	3.0	2011.666667	1.154701	2011.0	2011.00	2011.0	2012.00	2013.0
Pulsar Timing	5.0	1998.400000	8.384510	1992.0	1992.00	1994.0	2003.00	2011.0
Pulsation Timing Variations	1.0	2007.000000	NaN	2007.0	2007.00	2007.0	2007.00	2007.0
Radial Velocity	553.0	2007.518987	4.249052	1989.0	2005.00	2009.0	2011.00	2014.0
Transit	397.0	2011.236776	2.077867	2002.0	2010.00	2012.0	2013.00	2014.0
Transit Timing Variations	4.0	2012.500000	1.290994	2011.0	2011.75	2012.5	2013.25	2014.0

Tablas dinámicas son otra forma de agrupar

```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 titanic = sns.load_dataset('titanic')
5 titanic.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

Tablas dinámicas son otra forma de agrupar

```
1 titanic.groupby('sex')[['survived']].mean()
```

sex	survived
female	0.742038
male	0.188908

```
1 titanic.groupby(['sex', 'class'])['survived'].aggregate('mean')
```

```
sex     class
female  First     0.968085
        Second    0.921053
        Third     0.500000
male    First     0.368852
        Second    0.157407
        Third     0.135447
Name: survived, dtype: float64
```

Tablas dinámicas son otra forma de agrupar

```
titanic.pivot_table('survived', index='sex', columns='class')
```

class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

```
age = pd.cut(titanic['age'], [0, 18, 80])
titanic.pivot_table('survived', ['sex', age], 'class')
```

	class	First	Second	Third
sex	age			
female	(0, 18]	0.909091	1.000000	0.511628
	(18, 80]	0.972973	0.900000	0.423729
male	(0, 18]	0.800000	0.600000	0.215686
	(18, 80]	0.375000	0.071429	0.133663

Tablas dinámicas son otra forma de agrupar

```
fare = pd.qcut(titanic['fare'], 2)
titanic.pivot_table('survived', ['sex', age], [fare, 'class'])
```

	fare	(-0.001, 14.454]			(14.454, 512.329]		
	class	First	Second	Third	First	Second	Third
sex	age						
female	(0, 18]	NaN	1.000000	0.714286	0.909091	1.000000	0.318182
	(18, 80]	NaN	0.880000	0.444444	0.972973	0.914286	0.391304
male	(0, 18]	NaN	0.000000	0.260870	0.800000	0.818182	0.178571
	(18, 80]	0.0	0.098039	0.125000	0.391304	0.030303	0.192308

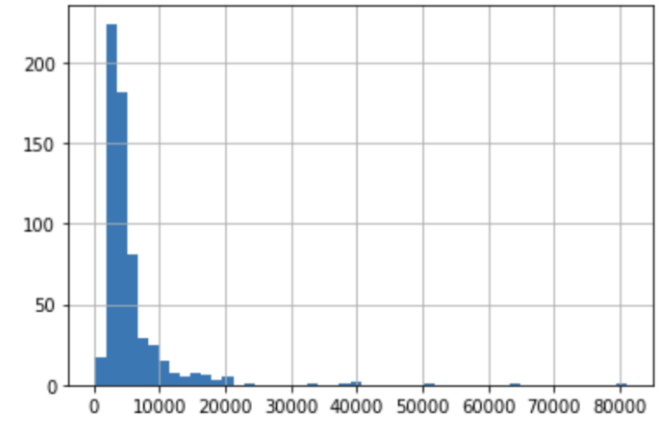
```
titanic.pivot_table(index='sex', columns='class',
                    aggfunc={'survived':sum, 'fare':'mean'})
```

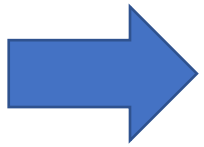
	fare			survived		
class	First	Second	Third	First	Second	Third
sex						
female	106.125798	21.970121	16.118810	91	70	72
male	67.226127	19.741782	12.661633	45	17	47

Cómo podemos presentar todo esto en Python

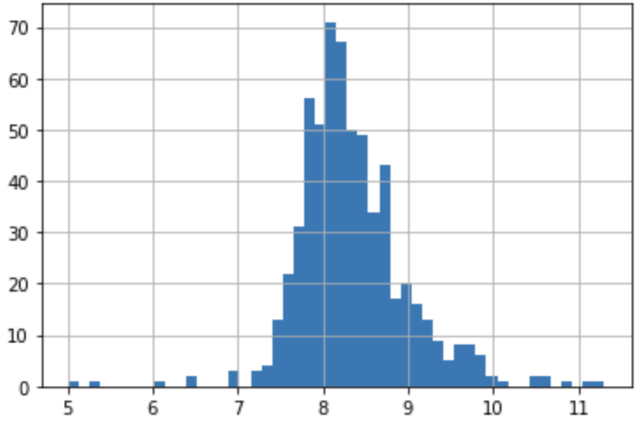
- Existen varias maneras en Python de presentar resultado gráficamente. Todas comparten la facilidad de uso y gran calidad de la presentación
- Con el fin de facilitar su uso, Pandas incorpora varias visualizaciones adecuadas a Series y DataFrame

```
1 df['ApplicantIncome'].hist(bins=50)
2 plt.show()
```





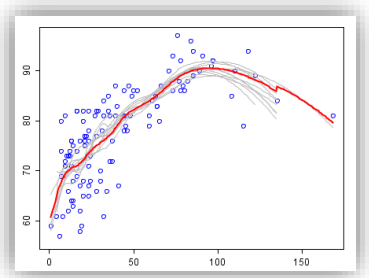
```
1 df['ApplicantIncome_log'] = np.log(df['ApplicantIncome'])
2 df['ApplicantIncome_log'].hist(bins=50)
3 plt.show()
```

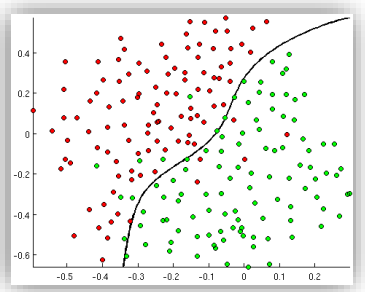


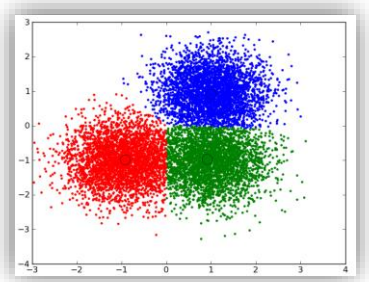
Antes de pasar a los modelos predictivos, necesitamos una breve introducción teórica al **Aprendizaje de Máquina (Machine Learning)**

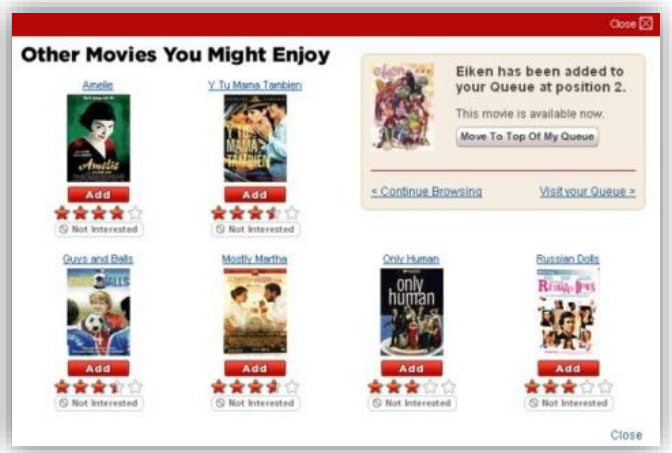
¿Qué es Machine Learning (ML)?

- ML se centra en el estudio de algoritmos que mejoran su rendimiento en una tarea, a través de la experiencia (mientras más datos mejor).
- Mejoran rendimiento con la experiencia (mientras más datos mejor*).
- Buscan aprendizaje más que modelamiento de datos (representaciones útiles del mundo)



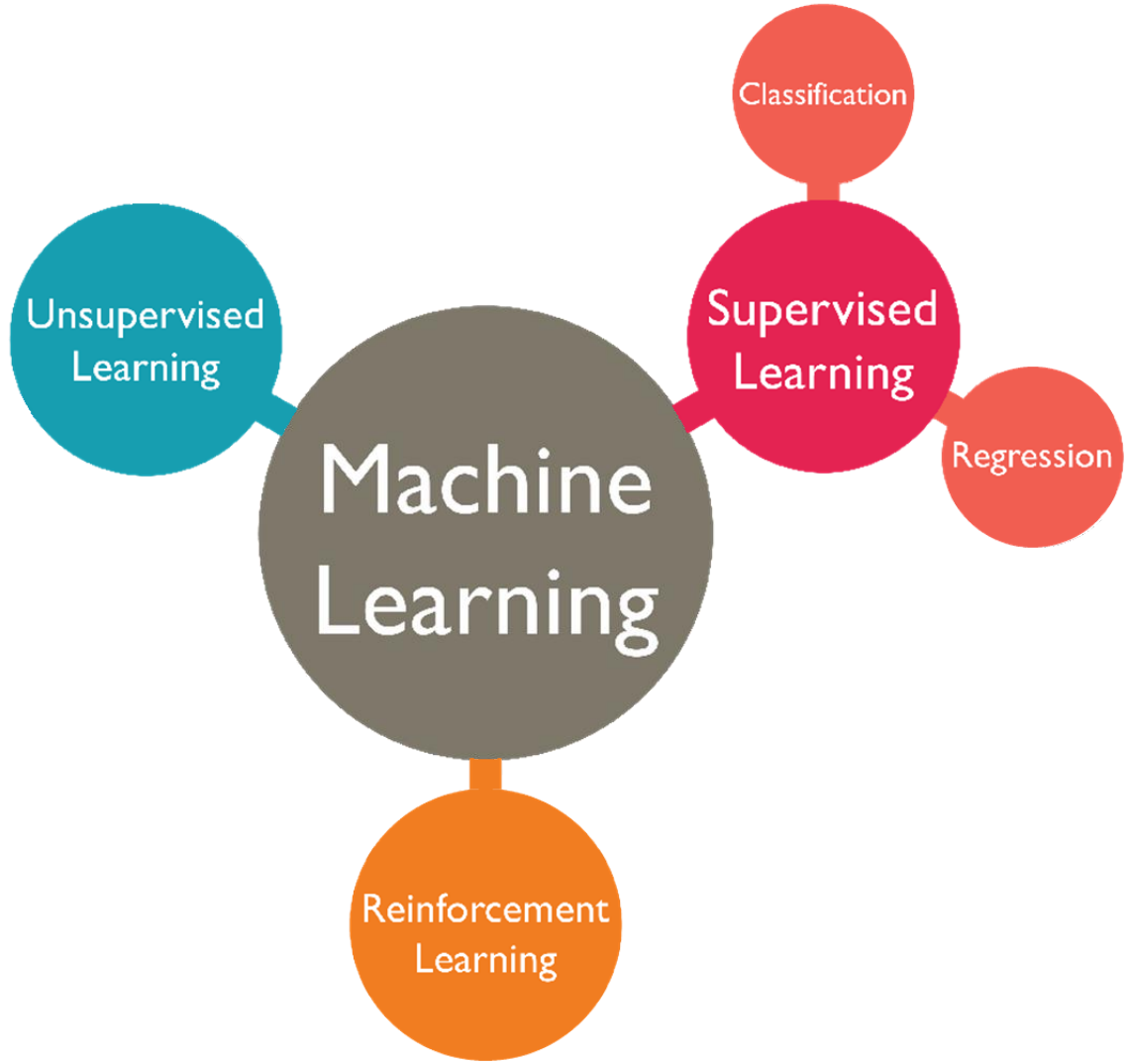





Close
Other Movies You Might Enjoy
Amelie
Y Tu Mama Tambien
Add
Not Interested
Eiken has been added to your Queue at position 2.
This movie is available now.
Move To Top Of My Queue
« Continue Browsing
Visit your Queue »
Guys and Balls
Mostly Martha
Only Human
Russian Dolls
Close


SPAM


CreditCard
SECURITY
CASH
VALID FROM


Classification
Unsupervised Learning
Supervised Learning
Machine Learning
Regression
Reinforcement Learning

Cat

¿  ?

Dog

Cat

¿  ?

Dog

Algoritmos de ML trabajan sobre datos multidimensionales

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

Wine Data Set

Download: Data Folder, Data Set Description

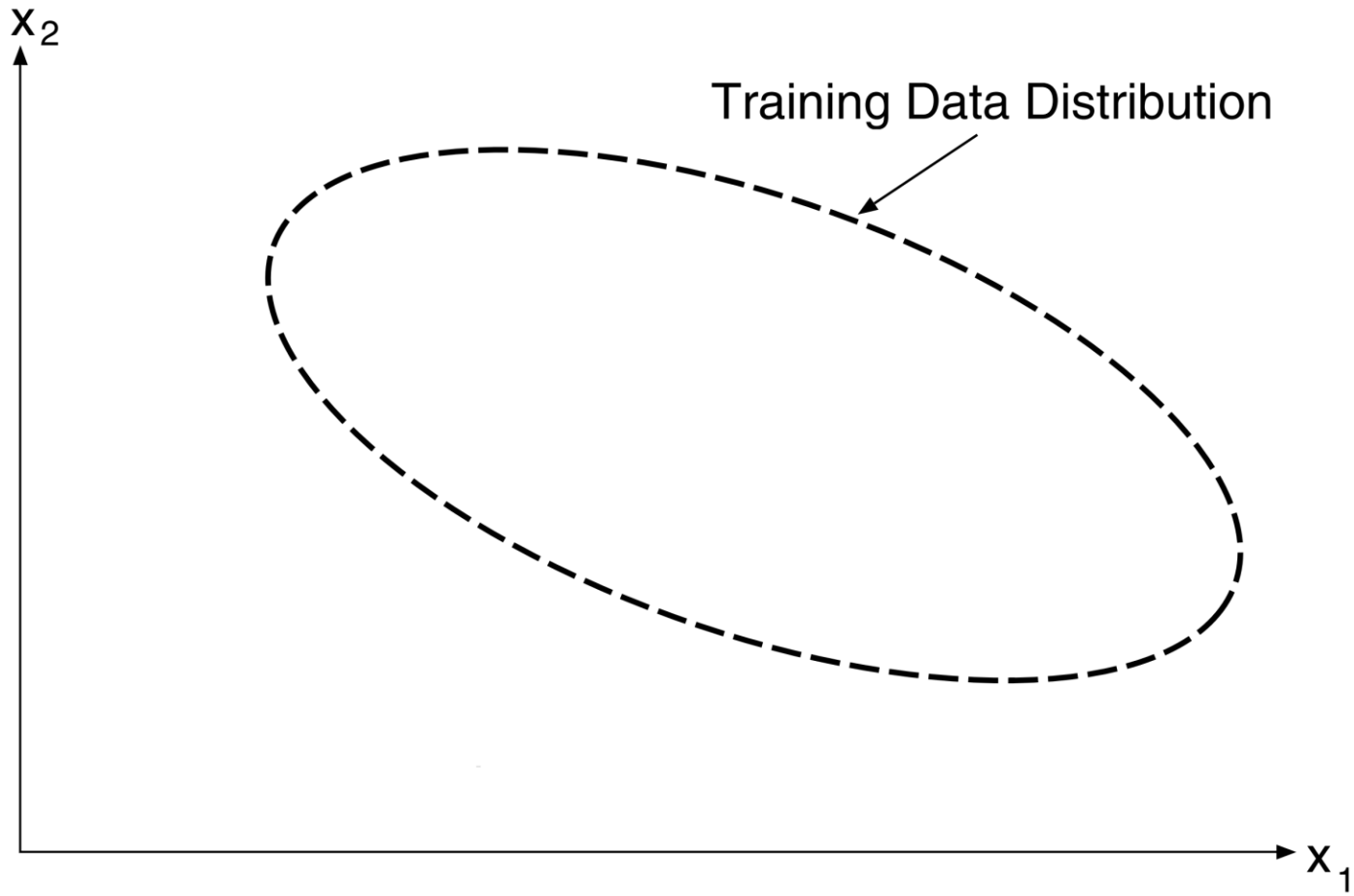
Abstract: Using chemical analysis determine the origin of wines

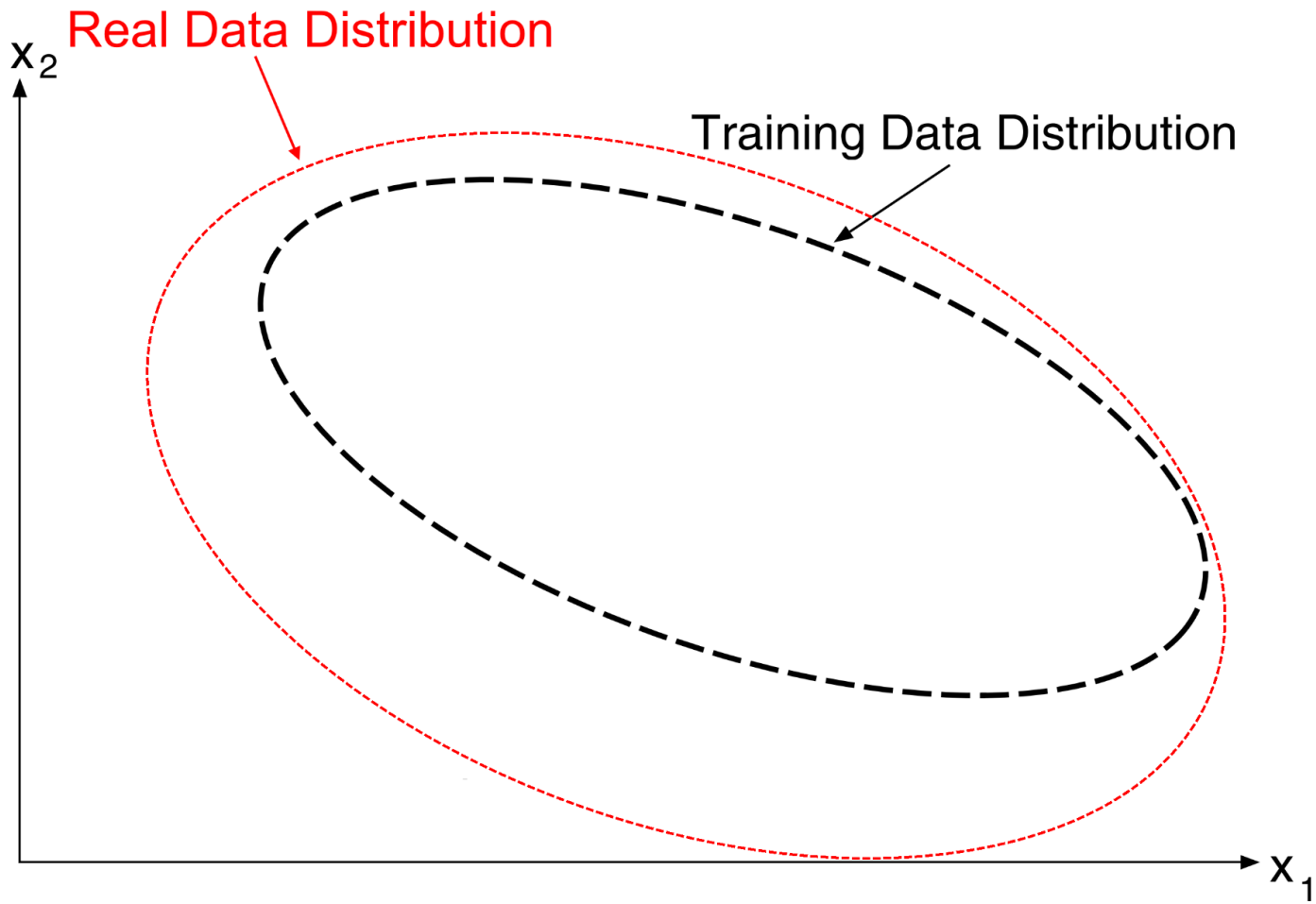
Data Set Characteristics:	Multivariate	**Number of Instances:**	178	**Area:**	Physical
Attribute Characteristics:	Integer, Real	**Number of Attributes:**	13	**Date Donated**	1991-07-01
Associated Tasks:	Classification	**Missing Values?**	No	**Number of Web Hits:**	573523

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

Typhoon number	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)
5111	96.0	20.7	20.6	27.6	47.4
5114	108.5	15.4	11.0	58.9	24.5
5201	181.2	8.1	1.7	40.1	7.9
5204	245.3	5.7	6.4	29.6	5.5
5209	117.5	23.3	22.0	46.6	61.7
5211	231.4	13.3	11.5	38.1	20.8
5309	293.6	4.0	7.2	35.4	5.6
5508	0.6	8.5	7.0	32.2	8.7
5512	227.6	10.0	10.4	19.3	16.0
5609	257.3	11.5	15.0	44.1	10.8

Cada dato (fila) del set de entrenamiento, puede considerarse como un vector en el espacio de características.

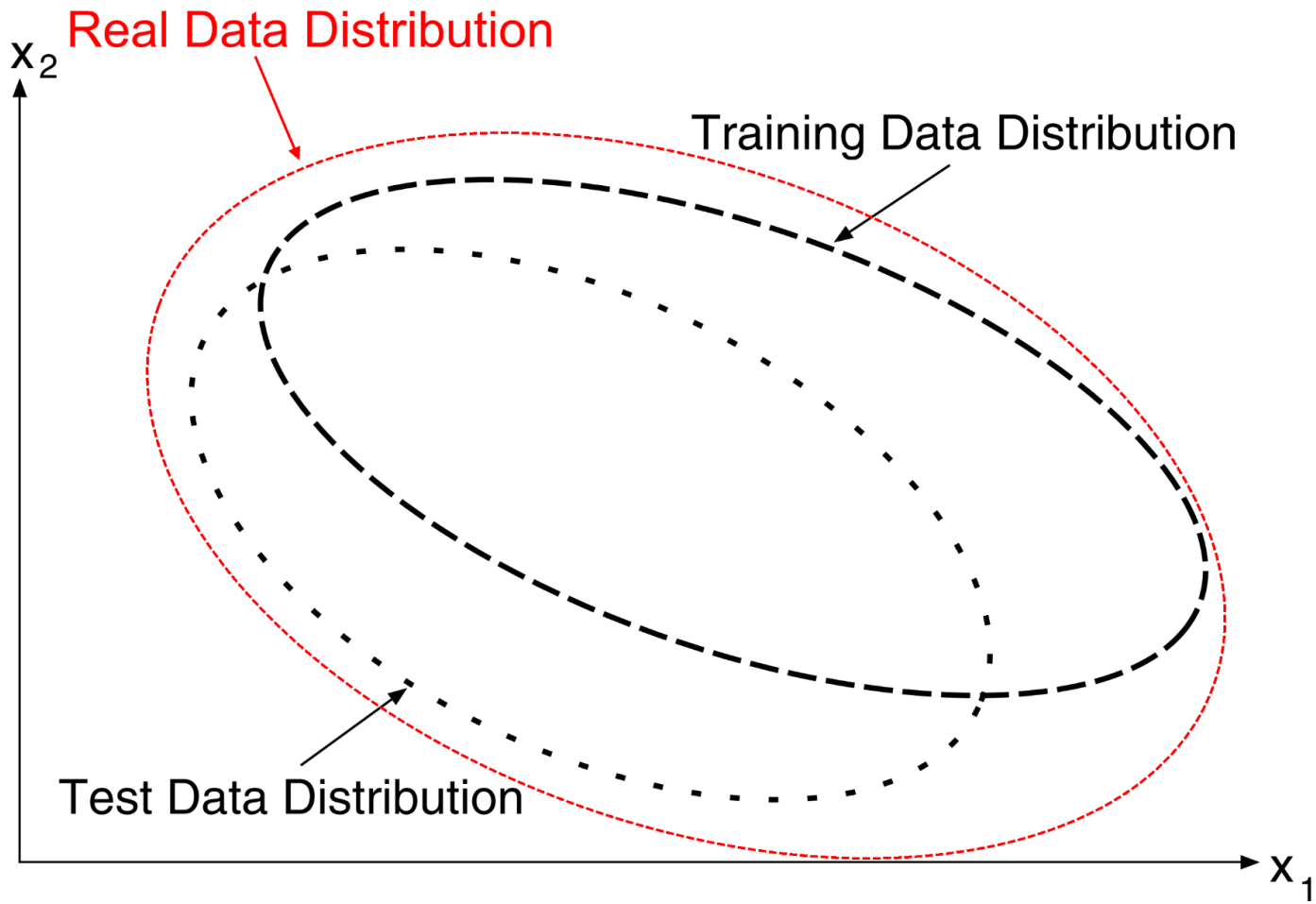

x 2
Training Data Distribution
x 1


Real Data Distribution
Training Data Distribution
x 2
x 1

Objetivo último es la generalización

	Typhoon number	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	5111	96.0	20.7	20.6	27.6	47.4
	5114	108.5	15.4	11.0	58.9	24.5
	5201	181.2	8.1	1.7	40.1	7.9
	5204	245.3	5.7	6.4	29.6	5.5
	5209	117.5	23.3	22.0	46.6	61.7
	5211	231.4	13.3	11.5	38.1	20.8
	5309	293.6	4.0	7.2	35.4	5.6
	5508	0.6	8.5	7.0	32.2	8.7
	5512	227.6	10.0	10.4	19.3	16.0
	5609	257.3	11.5	15.0	44.1	10.8
Test	0209	290.6	9.5	13.6	46.9	?
	0215	245.3	10.6	14.2	77.6	
	0306	227.0	4.4	7.9	20.8	
	0314	279.1	4.4	7.8	29.5	
	0415	266.3	8.7	8.8	32.9	
	0515	165.6	19.2	16.4	45.6	
	0601	136.5	10.7	12.2	4.6	
	0603	207.9	4.4	8.0	14.1	

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


Real Data Distribution
x_2
Training Data Distribution
Test Data Distribution
x_1

Existen algoritmos predictivos para todos los gustos

- Existen múltiples algoritmos de aprendizaje supervisado (demasiados en realidad)
- Cuál usar depende de los datos (cantidad y tipo), poder de cómputo disponible, tarea, etc.
- Es importante entender las diferencias para saber cuál utilizar…
- Para este curso, no es necesario comprender toda la matemática subyacente para poder usarlos (gracias a Python y scikit-learn)

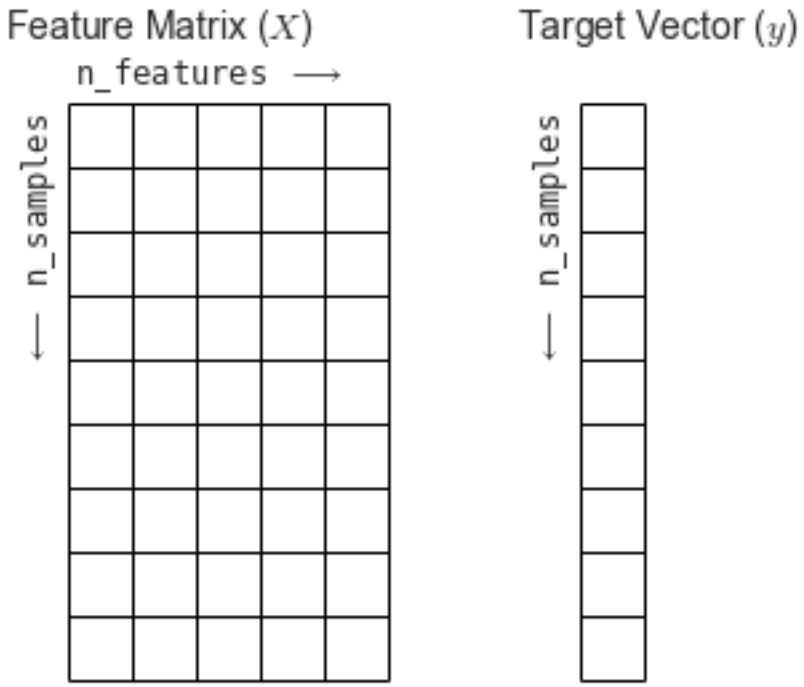
En este curso usaremos scikit-learn

- 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 otro paquetes, como Pandas.
- Posee además de una completa documentación en línea (https://scikit-learn.org/).

Esquema de datos es similar a Pandas

- Datos son representados por una matriz de *features* y un vector objetivo.
- 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

- 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

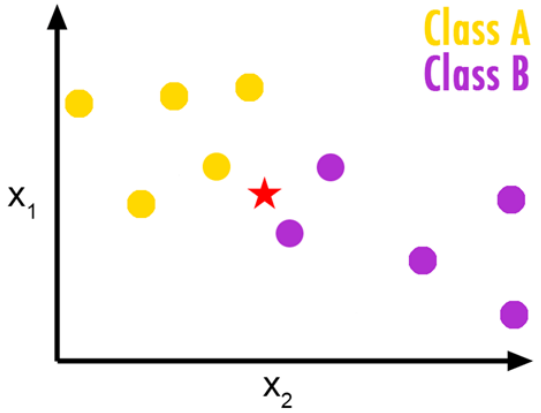
- En general, un caso de uso típico en Scikit-learn es como el siguiente:
  1. Elegir el modelo adecuado, importando la clase correspondiente desde *sklearn*.
  2. Obtener o generar matriz X y vector y.
  3. Entrenar el modelo llamando al fit(X, y).
  4. Aplicar el modelo al set de test, usando el método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

Veamos (por fin) 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()
```

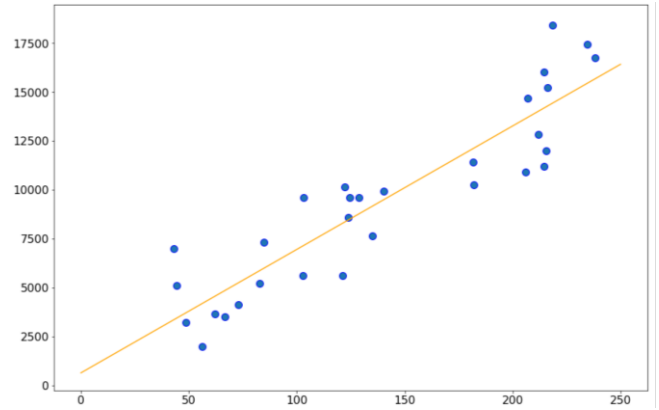


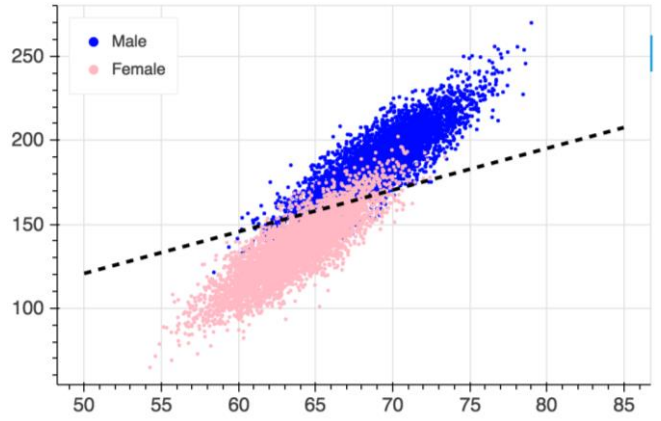
Regresión lineal y logística

- 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()
```

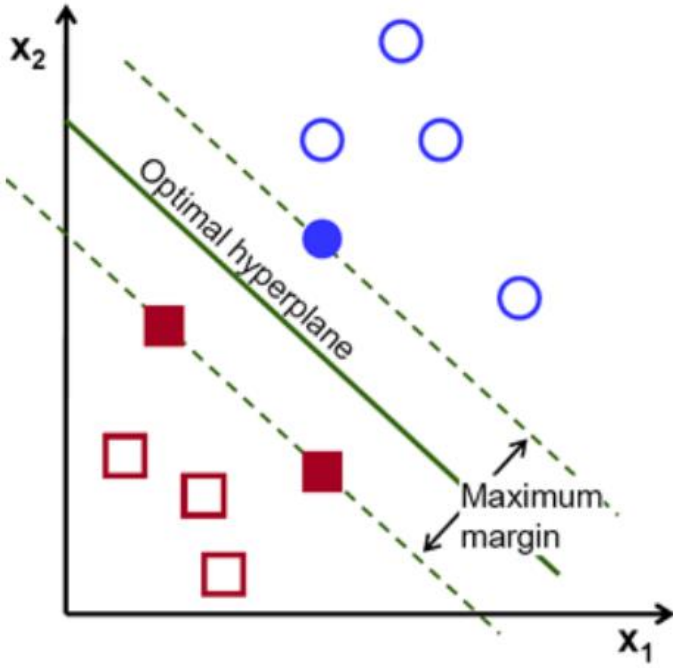




Support Vector Machine (SVM)

- 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:

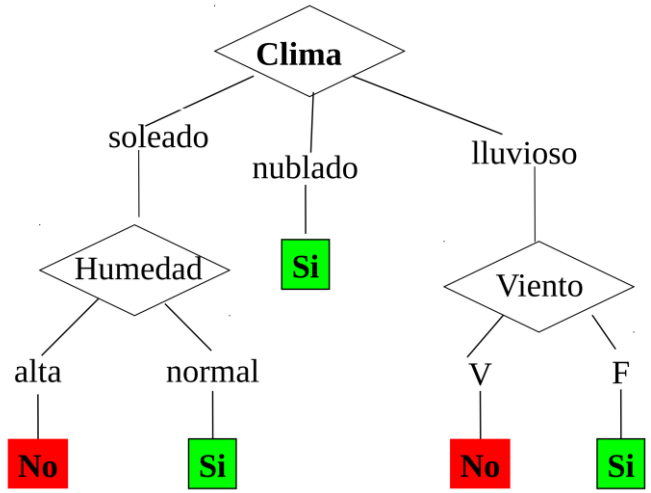
```
model = svm.SVC()
```



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:

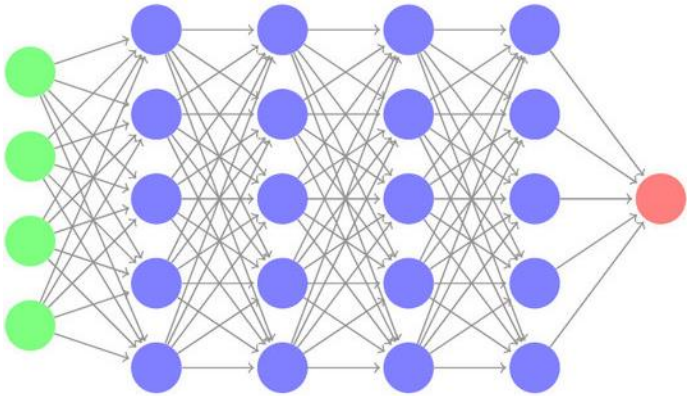
```
model = tree.DecisionTreeClassifier()
```



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()
```



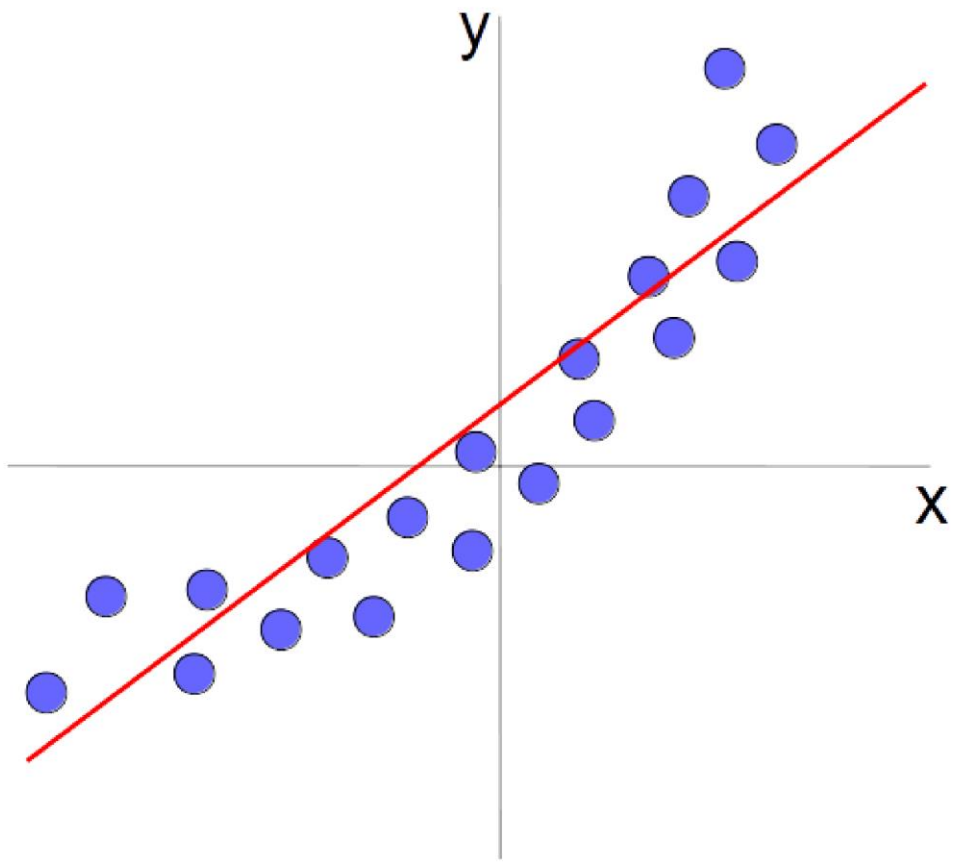
¿Cómo elegimos el mejor modelo para cada tarea?

- 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.

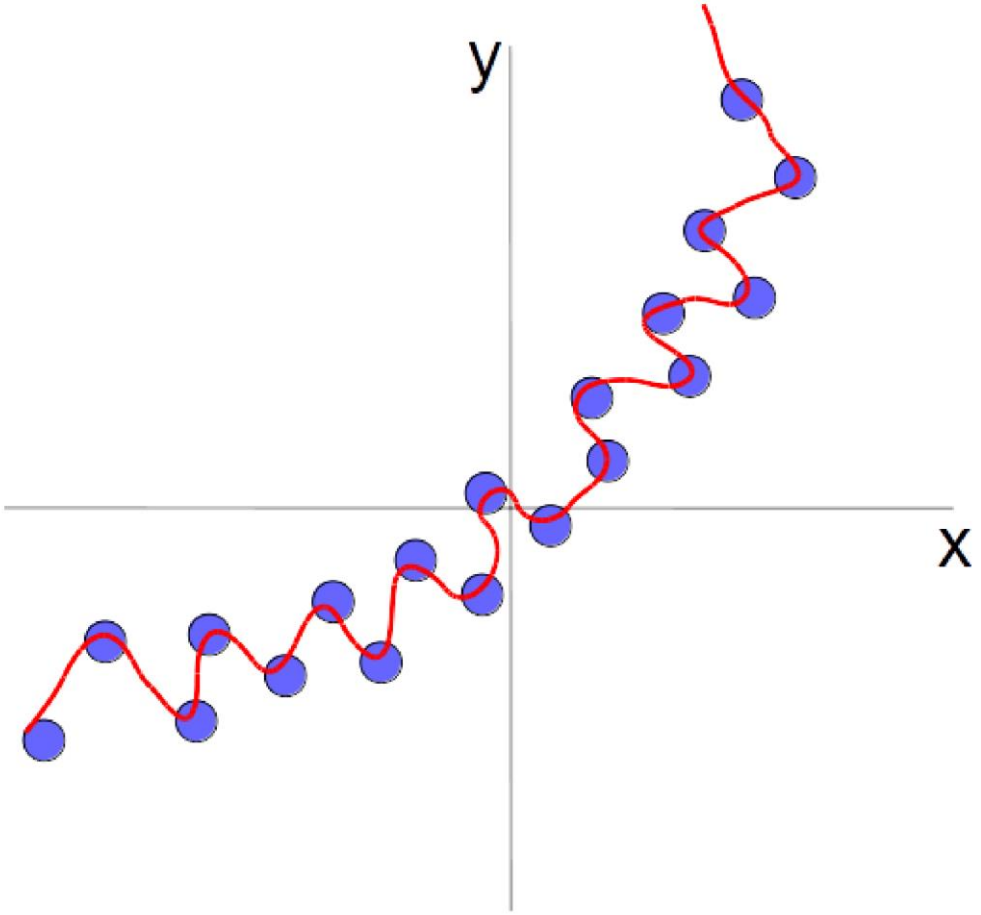
A pesar de ser clave, el set de entrenamiento no lo es todo

- 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.
- Poder de representación del algoritmo de aprendizaje pasa a ser el tema central.
- A continuación revisaremos el sobreentrenamiento, uno de los enemigos más terroríficos de ML.

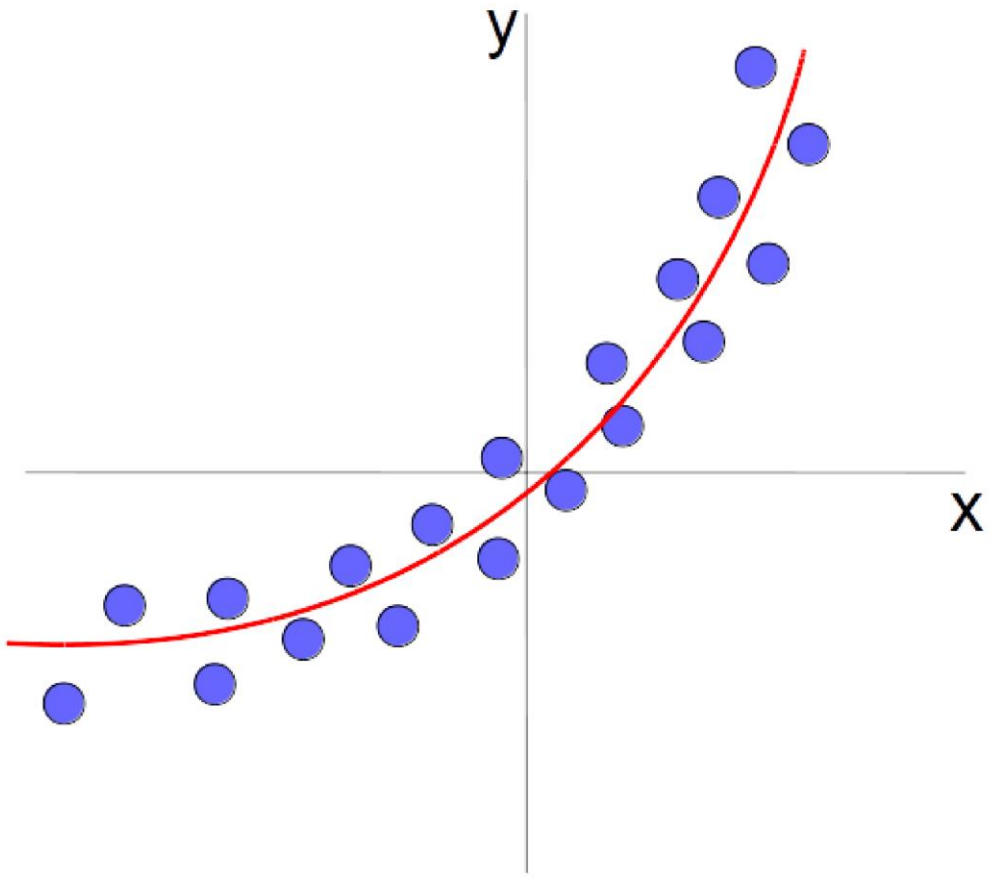
Subentrenamiento (o subajuste, o *underfitting*)



Sobreentrenamiento (o sobrebajuste, u *overfitting*)



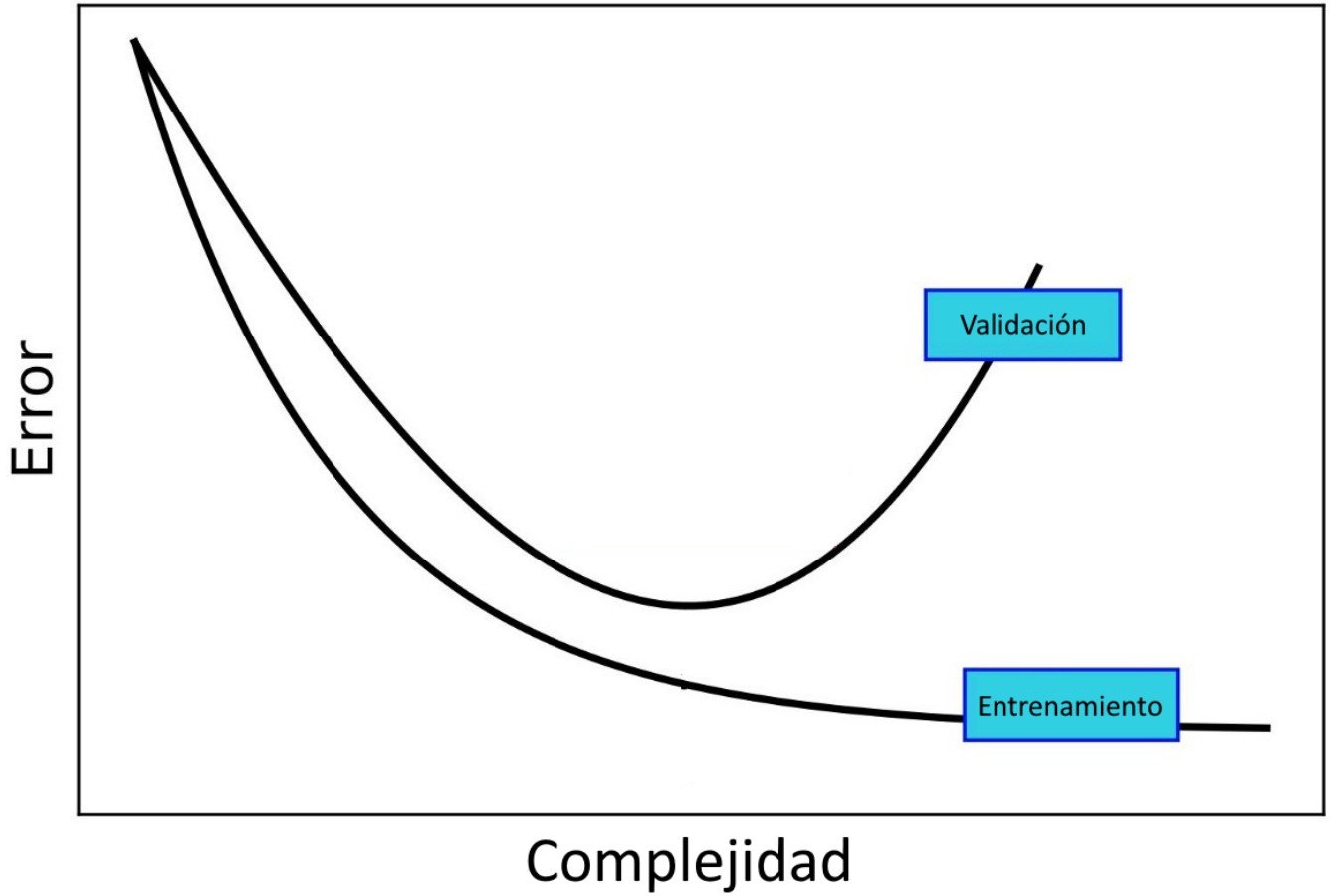
Complejidad correcta del modelo



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.
- El set con mejor rendimiento en validación es el elegido, y se usa para entrenar el modelo con todos los datos (entrenamiento + validación).

En general, error en validación baja y luego sube



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

IIC2115 – Programación como herramienta para la Ingeniería

Capítulo 4 – Análisis Exploratorio de Datos

Profesores: Hans Löbel
Francisco Garrido