

Series Temporales y Minería de Flujos de Datos

*Departamento de Ciencias de la Computación e
Inteligencia Artificial*

Universidad de Granada



**UNIVERSIDAD
DE GRANADA**

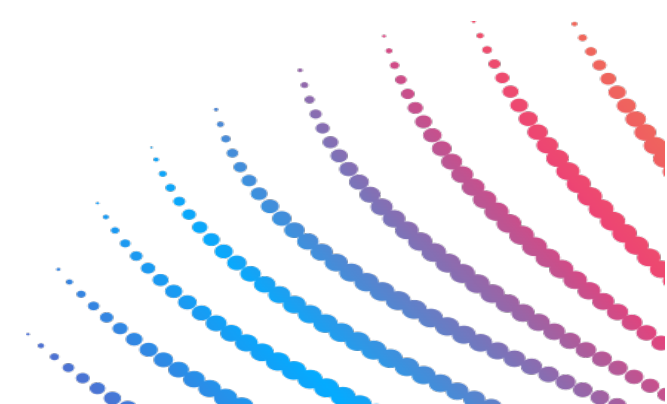
Prácticas de Series Temporales y Minería de Flujos de Datos:

Introducción

Prof. Manuel Pegalajar Cuéllar

¿ Quién soy ?

- **Manuel Pegalajar Cuéllar (*manupc@ugr.es*)**
 - Doctor en Ingeniería Informática
 - Profesor Titular de Universidad @ Universidad de Granada
 - Departamento de Ciencias de la Computación e I.A.
- **Intereses (docencia + investigación) :**
 - ✓ Diseño de algoritmos
 - ✓ Modelado y predicción de series temporales
 - ✓ Fusión de datos de sensores
 - ✓ Inteligencia Ambiental
 - ✓ Aprendizaje por Refuerzo
 - ✓ Redes Neuronales Artificiales
 - ✓ Computación Cuántica



Programa de prácticas

■ Sesión 1 (2.5h) : Introducción

- **Series Temporales** : Más allá de la predicción (forecasting)
- **Minería de Flujos de datos** : Presentación del problema y bibliotecas adicionales.

■ Sesión 2 (2.5h) : Series Temporales

- Metodología de análisis y predicción de series temporales.
- Bibliotecas para predicción con ARIMA.
- Ejemplos.

■ Sesión 3 (2.5h) : Minería de Flujos de Datos

- Minería de Flujos de Datos en Python : Biblioteca River.
- Ejemplos.





01 ●

Series Temporales : Más allá de la predicción

02 ●

Práctica de Minería de Flujo de Datos

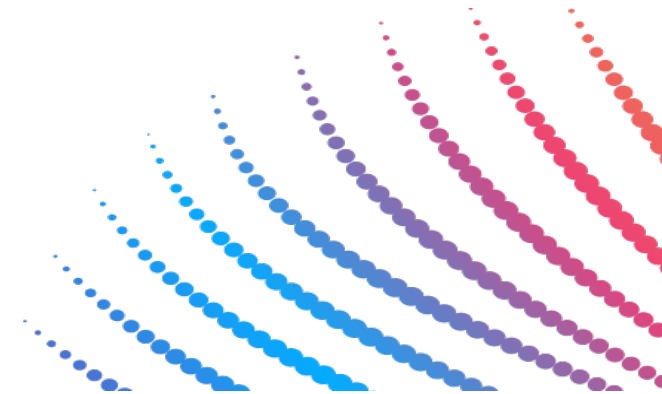


Bibliotecas requeridas

- **Statsmodels** : Biblioteca Python orientada a cálculos de estadística. Proporciona un amplio surtido de herramientas para realizar análisis de series temporales. **Será la usada para la práctica obligatoria.**



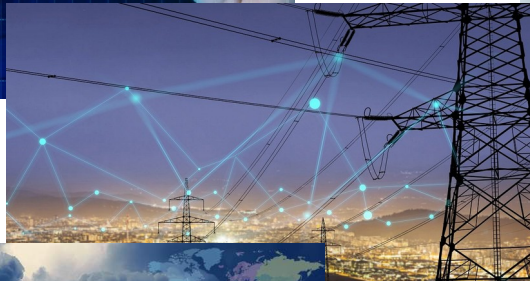
- **TsLearn** : Otra biblioteca de procesamiento de series temporales. Más centrada en resolución de problemas de clasificación, detección de anomalías, clustering...). Construida sobre SKLearn. **Será la utilizada en la primera sesión para ilustrar ejemplos de agrupamiento y clasificación.**
- **Otras bibliotecas requeridas** : pandas, matplotlib, numpy



Introducción

Definición

Una serie temporal $X(t) = \{x(1), x(2), x(3), \dots, x(t)\}$ es una secuencia de observaciones de un fenómeno dado, muestreadas periódicamente y indexadas en el tiempo.



Ejemplos :

Precios del mercado de valores
Uso de electricidad
Históricos meteorológicos
Monitorización de actividades
Frecuencia cardíaca
Etc.



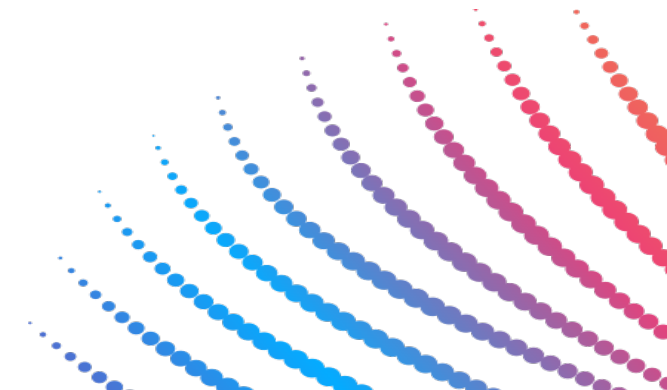
Introducción

■ Aplicación de series temporales : Predicción (futura)



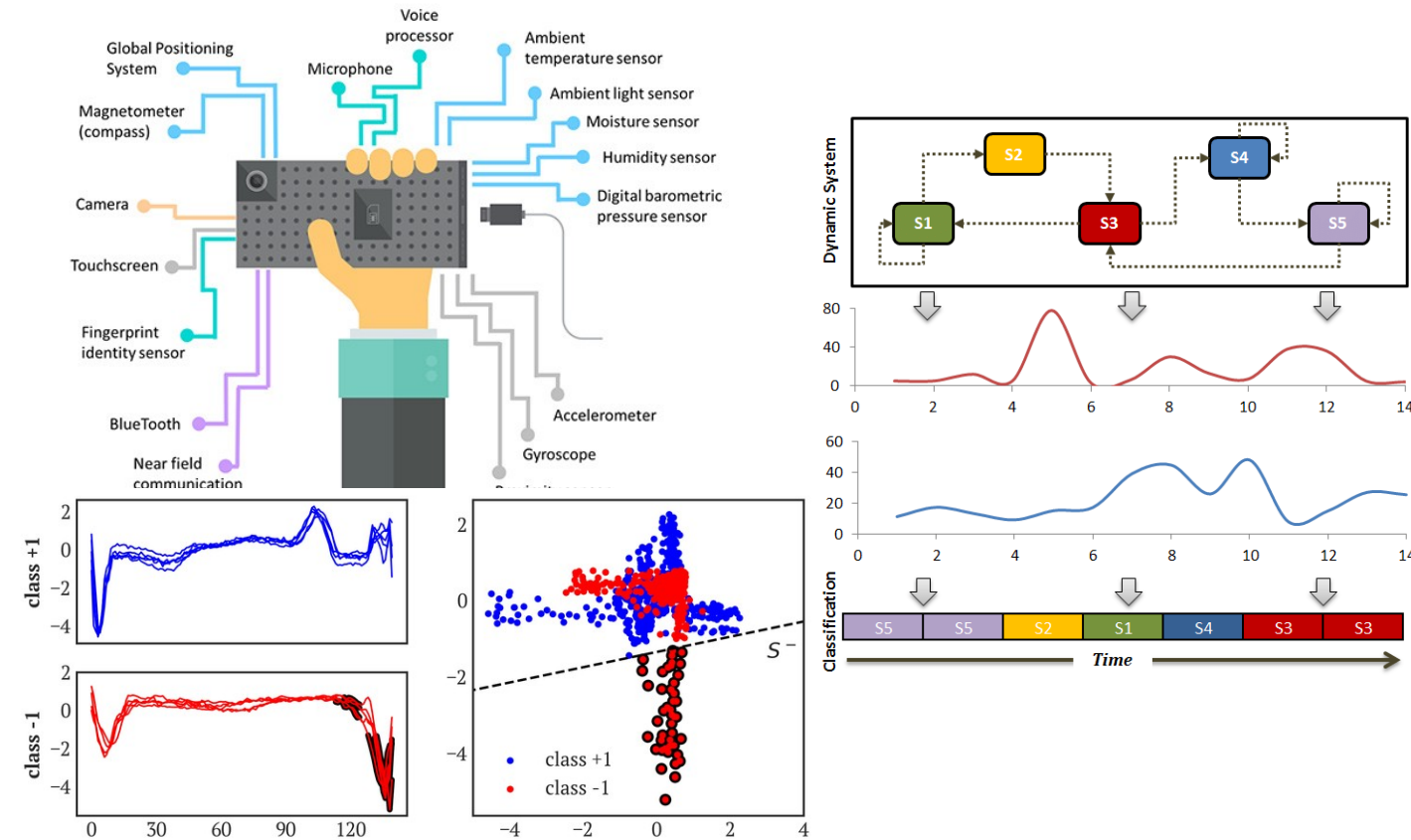
Ejemplos:

- Mercados de valores
- Ventas
- Meteorología
- ...



Introducción

■ Aplicación de series temporales : Clasificación



Ejemplos:

- Reconocimiento de actividades
- Minería de perfiles de usuario
- Reconocimiento de secuencias de ADN

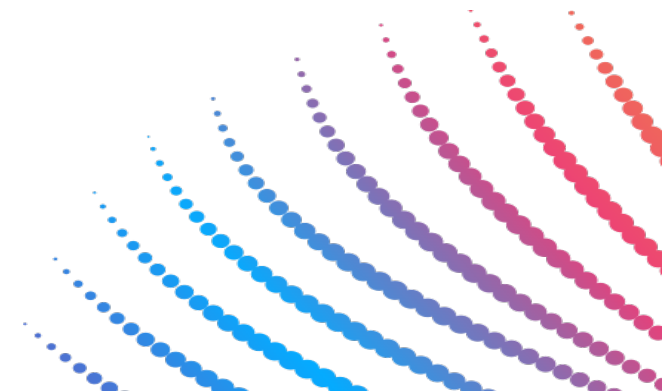
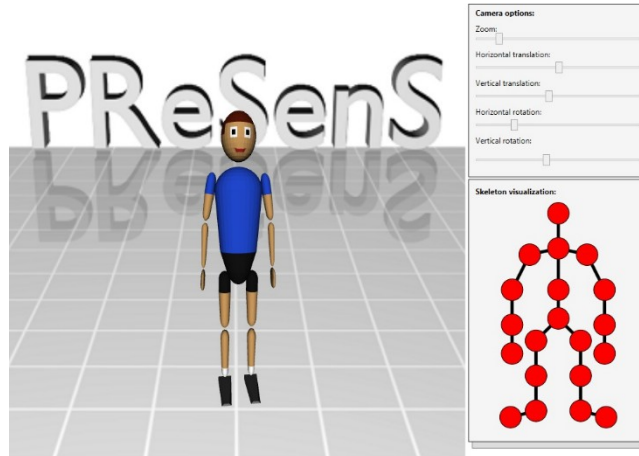
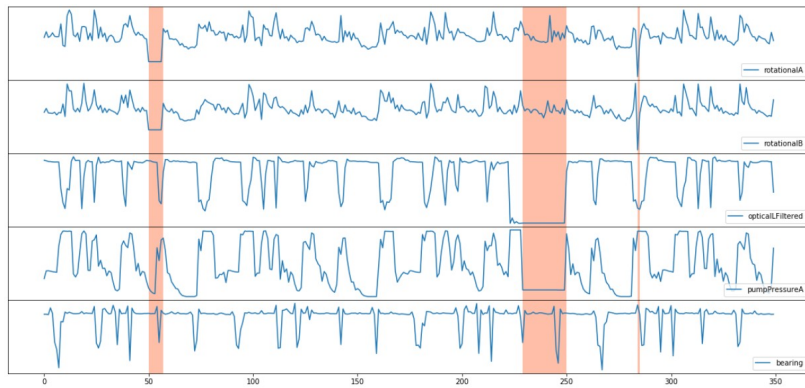
Introducción

■ Aplicación de series temporales : Detección de anomalías



Ejemplos:

- Detección de fraude en redes eléctricas
- Detección de fallos en sensores
- Asistencia guiada en ejercicio físico



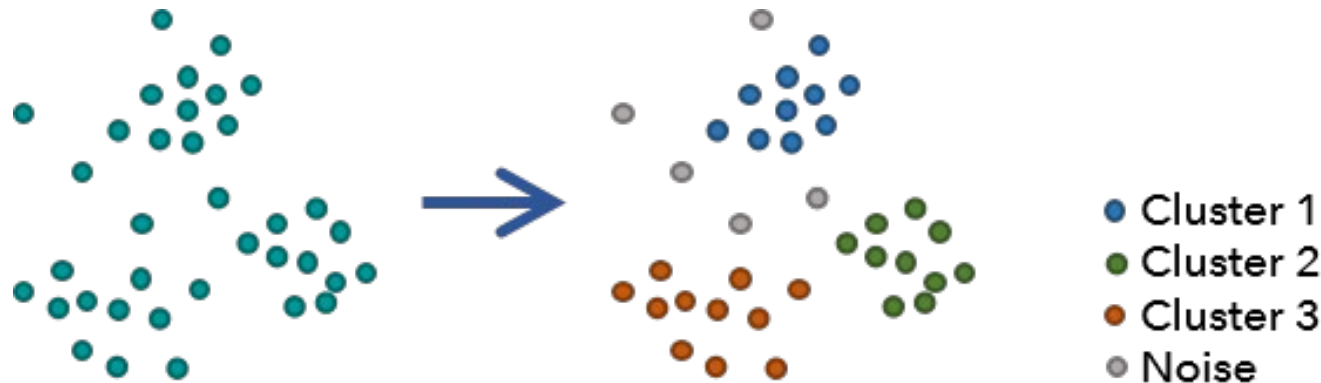
Agrupamiento

■ ¿Qué es el problema del agrupamiento (clustering) ?

El **Clustering** es un problema estudiado desde el aprendizaje no supervisado.

Se usa para encontrar patrones sobre cómo los datos se organizan en **categorías**, denominadas **clusters**.

Los elementos del mismo cluster se entiende que están relacionados « *de alguna manera* », según una **medida de similitud**.

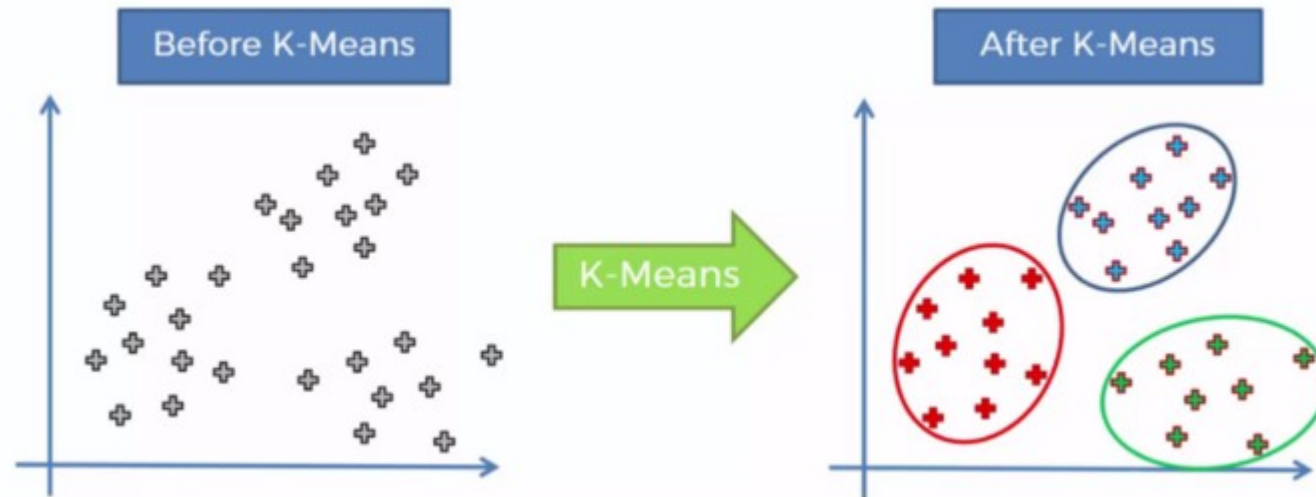


Agrupamiento

■ Ejemplo de algoritmo de clustering : K-means

K-Means es uno de los algoritmos más conocidos para agregar datos en un número fijado de **K** clusters. Es un algoritmo iterativo sobre elementos representativos de cada cluster (**centroides**), atendiendo a una medida de distancia entre elementos.

La **distancia euclídea** suele ser una medida muy usada para comparar distancias entre elementos.



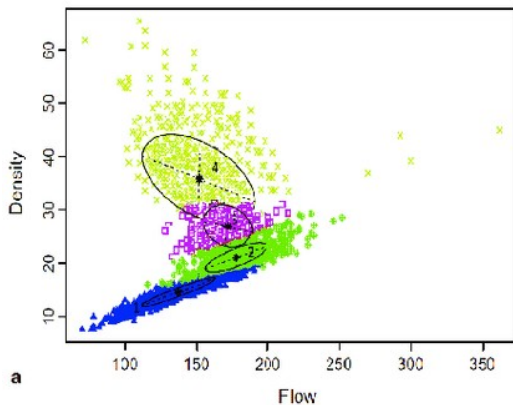
Agrupamiento

■ ¿Porqué hacer clustering de series temporales ?

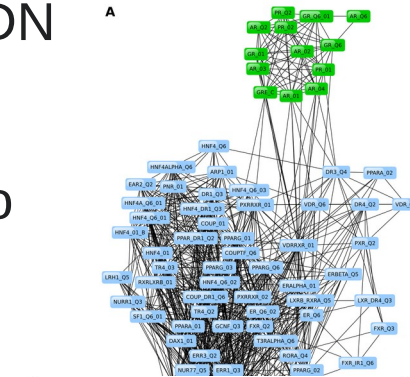
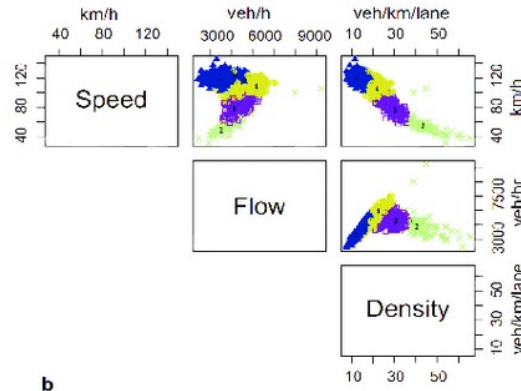
El **Clustering** de series temporales tiene muchas aplicaciones :

- Minería de perfiles de usuario (consumo energético, por ejemplo)
- Encontrar grupos de proteínas en ADN
- Análisis de EEC y ECG
- Minería de actividades y movimiento
- Análisis del comportamiento humano
- Análisis del tráfico
- Etc.

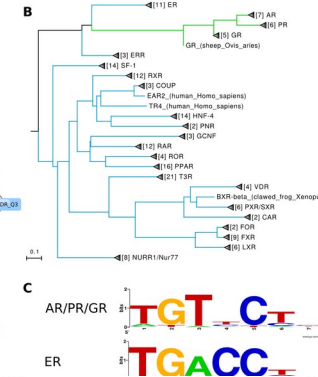
Classification



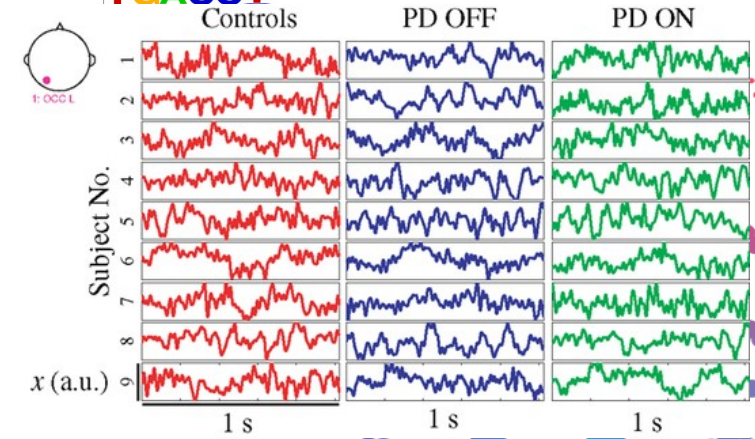
New Haven, CT, USA - 4 clusters



A



B



Building Load Profile (RT) and Electrical Power Consumption (kW)

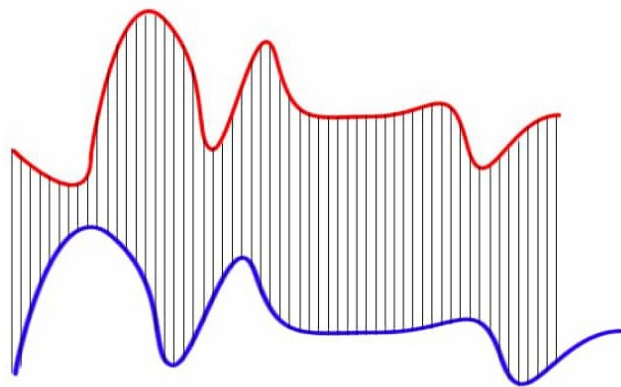
11-Apr (RT)
23-May (RT)
11-Apr (kW)
23-May (kW)

Agrupamiento

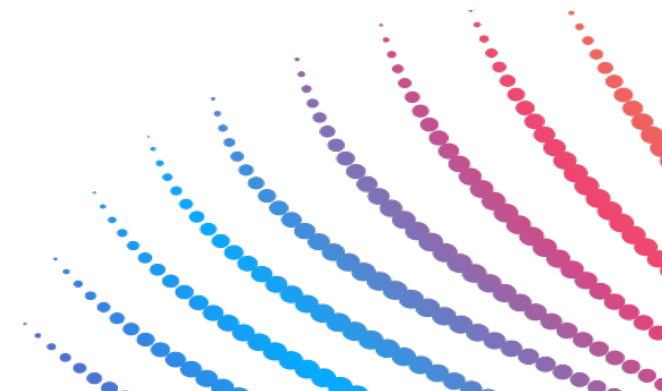
■ ¿Cómo hacer Clustering en series temporales ?

Lo primero : Necesitamos una ***medida de similitud*** que nos diga cómo de parecidas son dos series temporales ***x*** e ***y***. Ejemplo :

- **Suma de la distancia euclídea entre cada punto de las dos series. Problemas :**
 - Ambas series ***x*** e ***y*** deben tener la misma longitud.
 - Altamente sensible a traslaciones en el tiempo.

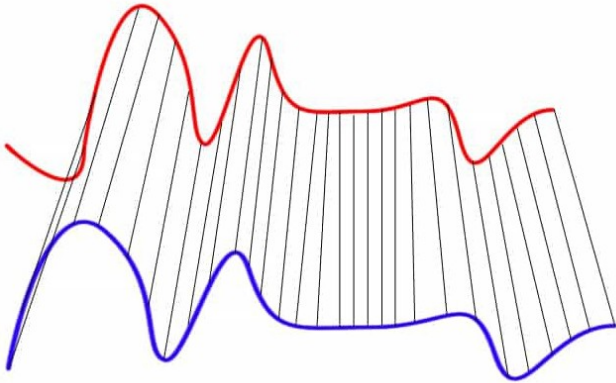


Euclidean Matching

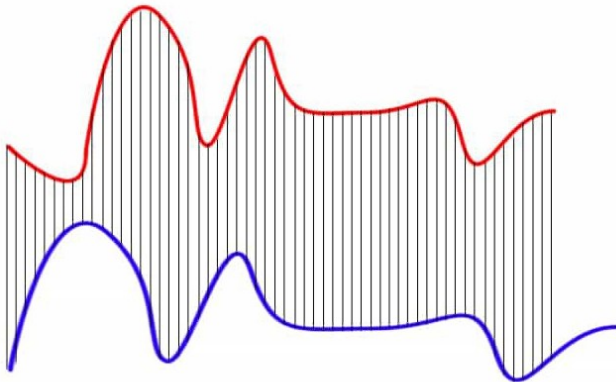


Agrupamiento

■ ¿Cómo hacer Clustering en series temporales ?



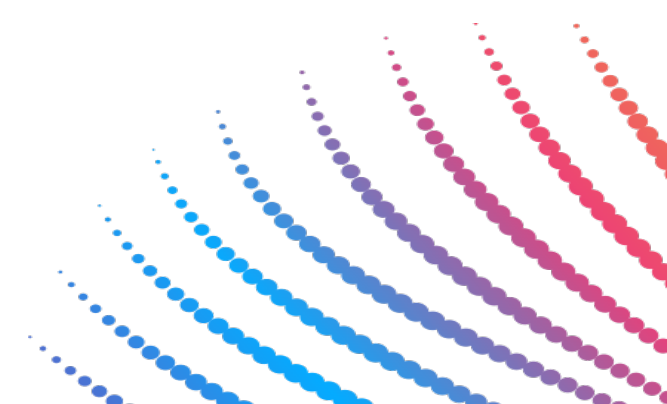
Dynamic Time Warping Matching



Euclidean Matching

Otro ejemplo de medida de similitud : **Dynamic Time Warping** (emparejamiento más idóneo entre dos series temporales)

- Ambas series x e y **no** deben tener la misma longitud.
- Invariante a traslaciones en el tiempo
- **Problema** : No es una métrica (falla la desigualdad triangular)



Agrupamiento

Dynamic Time Warping (DTW)

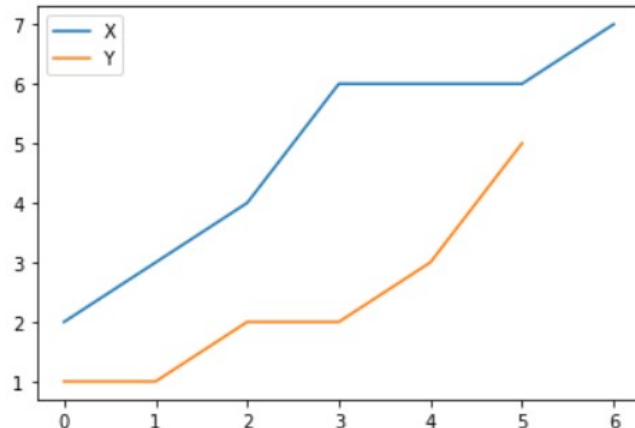
$DTW(n,m)$ mide el coste del mejor emparejamiento de una serie de datos $x(1..n)$ con otra serie de datos $y(1..m)$. Se calcula como:

$$DTW(i,j) = c(i,j) + \min\{DTW(i-1,j), DTW(i,j-1), DTW(i-1,j-1)\}$$

Donde $c(i,j)$ es el coste de emparejar $x(i)$ con $y(j)$

Time series examples

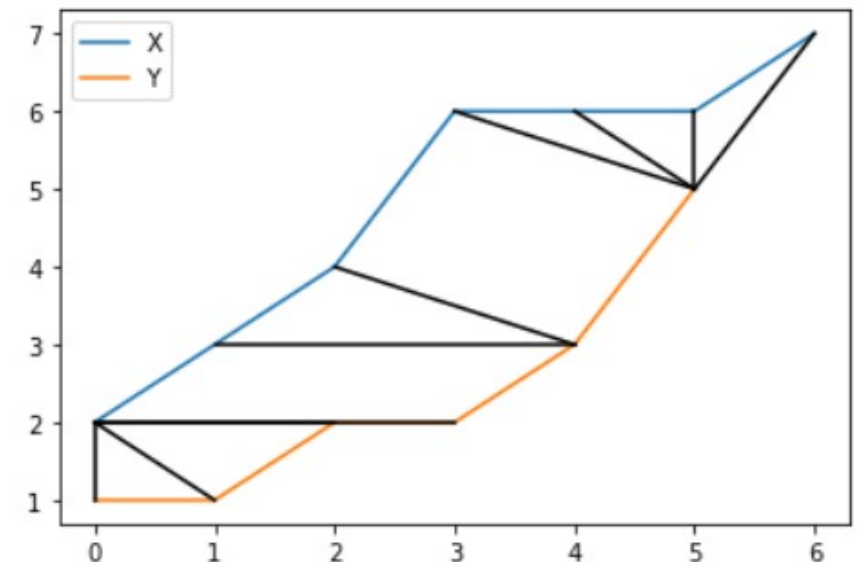
X = [2, 3, 4, 6, 6, 6, 7]
Y = [1, 1, 2, 2, 3, 5]



Matching matrix:

```
[[ 0. inf inf inf inf inf inf]
 [inf 1.  2.  2.  2.  3.  6.]
 [inf 3.  3.  3.  3.  2.  4.]
 [inf 6.  6.  5.  5.  3.  3.]
 [inf 11. 11.  9.  9.  6.  4.]
 [inf 16. 16. 13. 13.  9.  5.]
 [inf 21. 21. 17. 17. 12.  6.]
 [inf 27. 27. 22. 22. 16.  8.]]
```

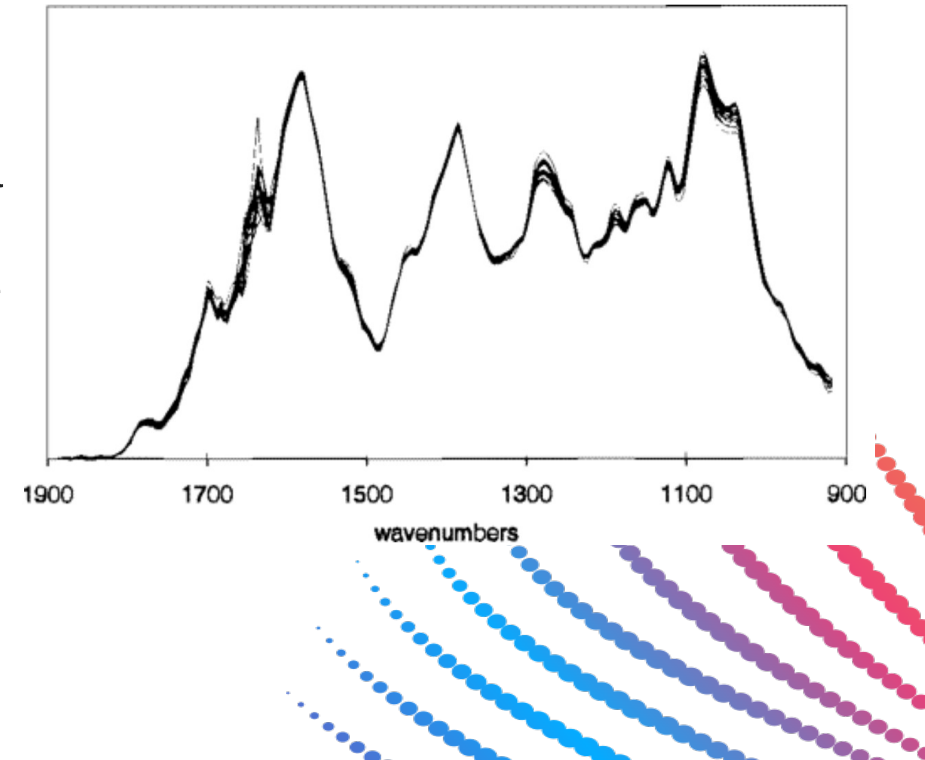
Retrieve best matching



Agrupamiento

■ Ejemplo de agrupamiento de series temporales (Problema Coffee)

- Se nos proporciona espectrogramas de dos variantes de café : Robusta y Arábica.
- **Objetivo** : ¿ Se puede conseguir un agrupamiento de los espectrogramas de ambas clases ?
- Se proporcionan dos ficheros :
 - **Coffee_TRAIN.csv** : Datos de entrenamiento. Cada fila es un espectrograma (serie temporal) de un grano de café. La última columna contiene la clase a la que pertenece (0=Robusta / 1=Arábica)
 - **Coffee_TEST.csv** : Datos para test.
- Dataset : 28 series temporales en entrenamiento y test. Balanceado. Cada serie temporal tiene la misma longitud : 286 valores para todas las frecuencias del espectrograma.

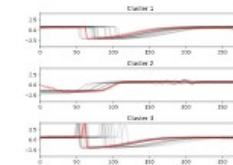


Agrupamiento : TSLearn

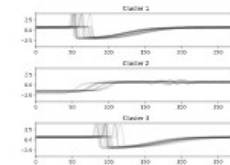
■ La biblioteca TSLearn

- La biblioteca TSLearn (Time Series Learn) proporciona varias herramientas para realizar agrupamiento, clasificación y regresión con series temporales.
- Está basada en *Scikit-learn* (*sklearn*)

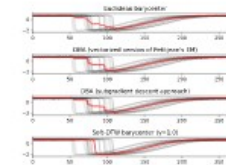
Clustering and Barycenters



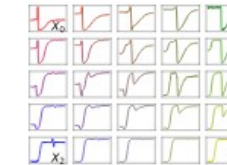
KShape



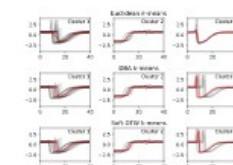
Kernel k-means



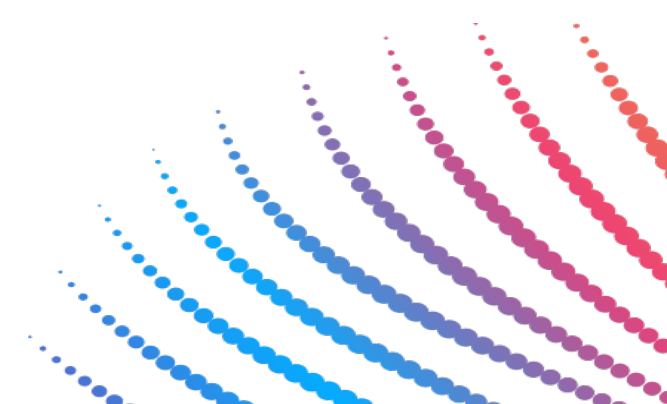
Barycenters



Soft-DTW weighted
barycenters



k-means



Agrupamiento : TSLearn

■ La biblioteca TSLearn : Ejemplo de agrupamiento con K-Means

- Uno de los métodos que contiene para realizar agrupamientos es el clásico *K-Means*, adaptado para series temporales.

`tslearn.clustering.TimeSeriesKMeans`

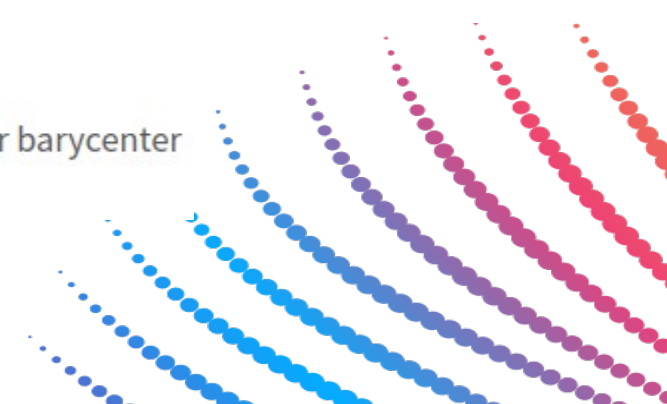
```
class tslearn.clustering. TimeSeriesKMeans (n_clusters=3, max_iter=50, tol=1e-06, n_init=1, metric='euclidean',  
max_iter_barycenter=100, metric_params=None, n_jobs=None, dtw_inertia=False, verbose=0, random_state=None, init='k-  
means++')
```

[\[source\]](#)

K-means clustering for time-series data.

metric : {"euclidean", "dtw", "softdtw"} (default: "euclidean")

Metric to be used for both cluster assignment and barycenter computation. If "dtw", DBA is used for barycenter computation.



Agrupamiento : TSlearn

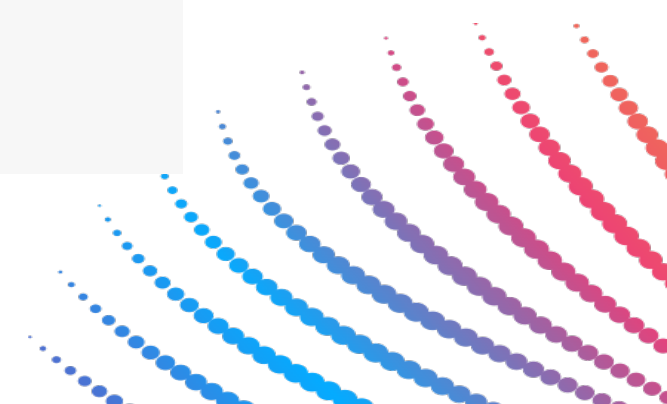
■ Time Series K-Means para el problema Coffee (*Coffee.py*)

Coffee Clustering

```
In [3]: import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from tslearn.clustering import TimeSeriesKMeans
from tslearn.datasets import CachedDatasets
from tslearn.preprocessing import TimeSeriesScalerMeanVariance, TimeSeriesResampler
```

Load data

```
In [4]: # Get data from file
train= pd.read_csv('./Coffee_TRAIN.csv', header=None).to_numpy()
x_train= train[:, :-1]
y_train= train[:, -1]
labels= np.unique(y_train)
```



Agrupamiento : TSLearn

■ Time Series K-Means (*Coffee.py*)

- Entendemos que puede haber 2 clusters : Uno para cada tipo de grano de café.
- Configuramos el agrupador y lo entrenamos con los datos de entrada (***x_train***)

Clustering

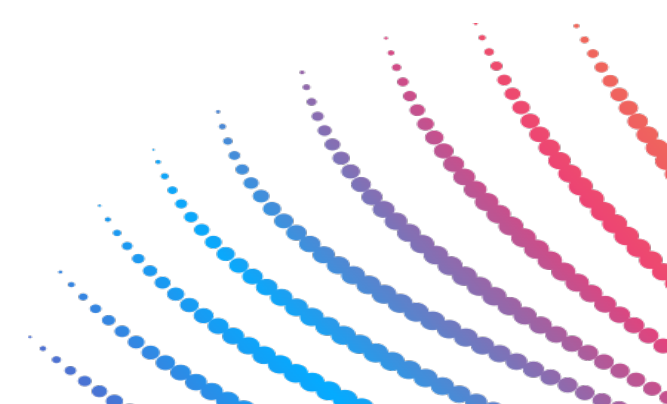
```
In [3]: # Suposición inicial del número de clusters
        Num_Clusters= 2

        # Creamos algoritmo de K-Means con DTW como medida de distancia entre series de datos
        DTWkm = TimeSeriesKMeans(n_clusters=Num_Clusters, metric="dtw", max_iter= 20)
        DTWkm.fit(x_train)
```

```
Out[3]: 

TimeSeriesKMeans


        TimeSeriesKMeans(max_iter=20, metric='dtw', n_clusters=2)
```



Clasificación con el agrupamiento

■ Time Series K-Means (Coffee.py)

- Mostramos los prototipos obtenidos como centroides e información adicional.

Results

In [4]:

```
plt.figure()

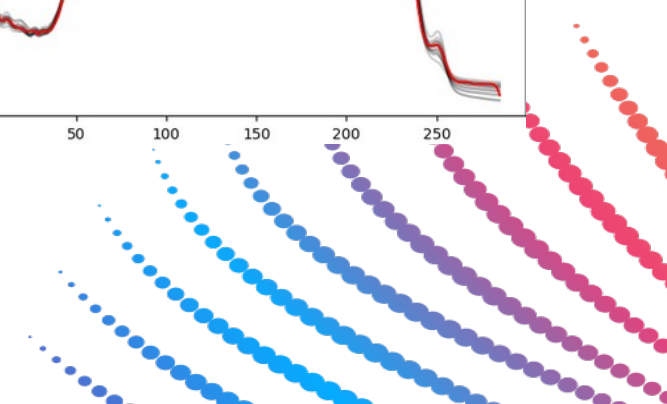
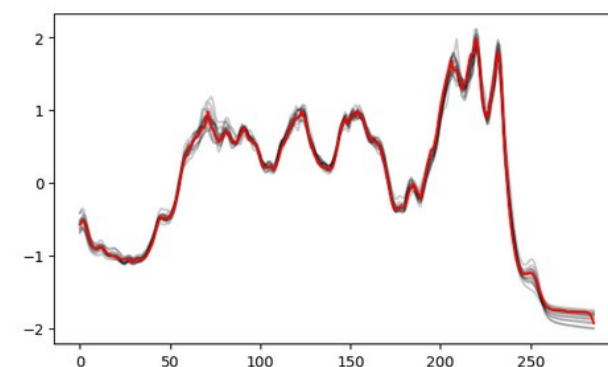
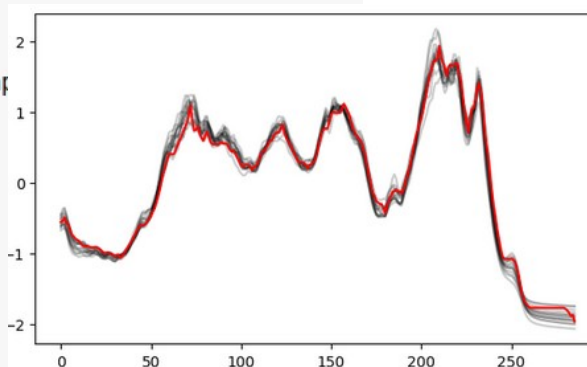
# Mostramos resultados reales
y_pred = DTWkm.predict(x_train)

# Mostramos tamaño de cada cluster
for i in range(Num_Clusters):
    print('Cluster {} contains {} elements.'.format(i+1, len(np

# Mostramos gráficos de los clusters y el centroide
plt.figure(figsize=(15, 4))
for yi in range(Num_Clusters):

    # Serie temporal del cluster actual
    plt.subplot(1, Num_Clusters, yi + 1)
    for xx in x_train[y_pred == yi, :]:
        plt.plot(xx.squeeze(), "k-", alpha=.2)
    # Centroide
    plt.plot(DTWkm.cluster_centers_[yi].ravel(), "r-")
plt.show()
```

```
Cluster 1 contains 14 elements.
Cluster 2 contains 14 elements.
```



Clasificación con el agrupamiento

■ Time Series K-Means (Coffee.py)

- Comprobamos si el clustering es correcto comprobando la clasificación de cada serie.
- En este caso : Cluster 0 a clase 0 ; Cluster 1 a clase 1

Check true class and assigned cluster

In [5]:

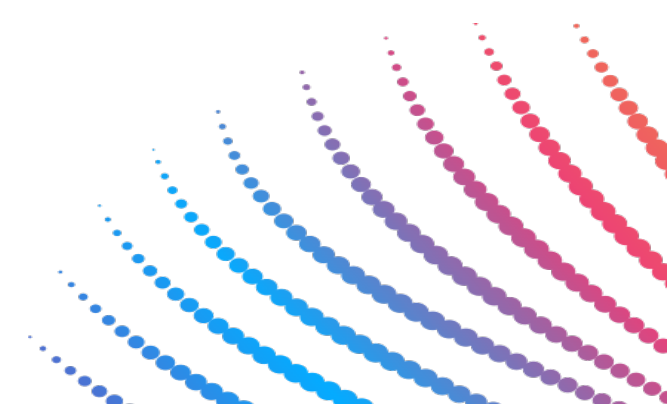
```
# No sabemos qué cluster es para cada clase. Usaremos la clase mayoritaria
# Contamos cuántos elementos de cada clase han ido a cada cluster
conf_mat= np.zeros((2, 2))
for y, yp in zip(y_train, y_pred):
    conf_mat[y, yp] += 1

cluster_de_clase= np.argmax(conf_mat, axis=1)
print('Matriz de confusión:\n', conf_mat)
print('Asignación de clase a cluster: ', cluster_de_clase)
```

Matriz de confusión:

```
[[14.  0.]
 [ 0. 14.]
```

Asignación de clase a cluster: [0 1]



Clasificación con el agrupamiento

■ Time Series K-Means (Coffee.py)

Check classification accuracy

```
In [6]: accuracy= 0
for y, yp in zip(y_train, y_pred):
    cluster_y= cluster_de_clase[y]
    if cluster_y == yp:
        accuracy+= 1
accuracy= accuracy*100/len(y_train)

print('Hay una precisión en entrenamiento del {}'.format(accuracy))
```

Hay una precisión en entrenamiento del 100.0%

Check in test

```
In [7]: # Cargamos conjunto de test
test= pd.read_csv('./Coffee_TEST.csv', header=None).to_numpy()
x_test= test[:, :-1]
y_test= test[:, -1].astype(int)

# Mostramos resultados reales: Predecimos cluster para todas las series en x_test
y_pred= DTWkm.predict(x_test).astype(int)
accuracy= 0
for y, yp in zip(y_train, y_pred):
    cluster_y= cluster_de_clase[y]
    if cluster_y == yp:
        accuracy+= 1
accuracy= accuracy*100/len(y_train)

print('Hay una precisión en test del {}'.format(accuracy))
```

Hay una precisión en test del 92.85714285714286%

Agrupamiento : Ejercicio

■ Ejercicio (Resuelto en *Trace.py*)

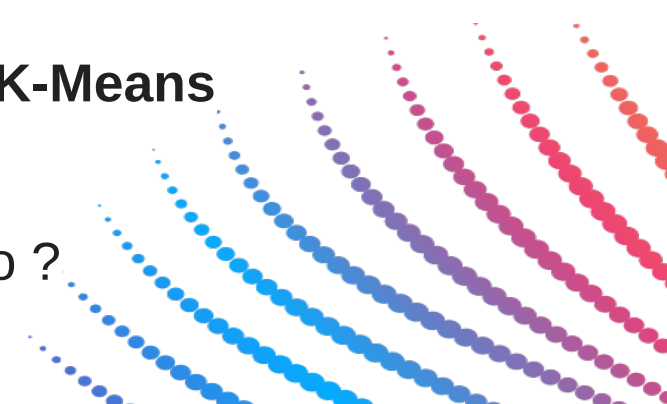
Transient Classification Benchmark (trace project) : El fichero **Trace.csv** contiene datos de simulaciones de fallos de instrumentación en una planta nuclear (créditos a Davide Roverso).

Este dataset contiene 3 clases. Hay 23 series temporales de cada clase, y cada una tiene una longitud de 275. Las series de datos han sido ya preprocesadas y normalizadas.

Las series temporales son multi-dimensionales : En cada instante de tiempo, el valor de la serie está formado por 4 atributos.

Se pide :

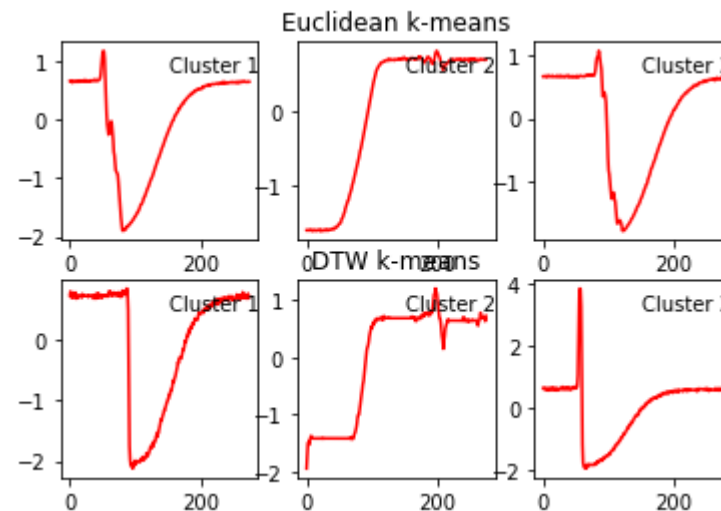
- **Agrupar las series temporales en clusters usando Time Series K-Means**
- Use tanto la medida de similitud ***Euclidean*** como ***DTW***
- Muestre los centroides con cada medida
- ¿Puede ver alguna diferencia entre los centroides de cada algoritmo ?



Agrupamiento : Ejercicio

■ Ejercicio (Resuelto en *Trace.py*)

Solución :



Respuesta : Los centroides 1 y 3 obtenidos con la distancia euclídea son muy similares entre sí. A su vez, existe una notable diferencia entre los mismos para DTW. Esto sugiere que, aunque las series tienen el mismo tamaño, podrían existir traslaciones en los datos que la distancia euclídea no puede modelar. Se ve claramente en el patrón del cluster 2, donde se acentúan los picos centrales existentes en el centroide para DTW.



01 ●

Series Temporales : Más allá de la predicción

02 ●

Práctica de Minería de Flujo de Datos



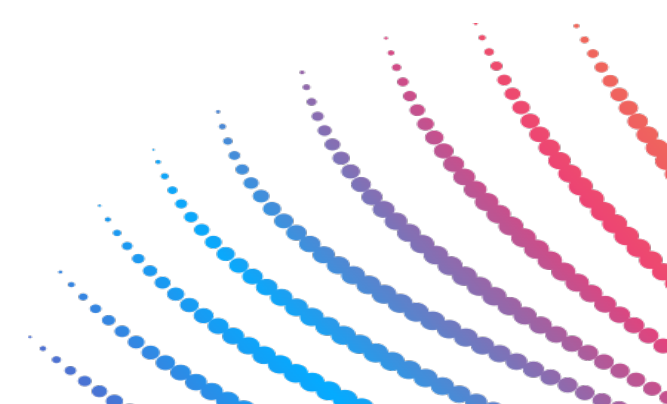
El entorno CartPole-v1

■ Materiales : Instalación de software

- **River** : Biblioteca para Minería de Flujos de Datos en Python.



```
conda create --name MFD python=3.9
conda activate MFD
pip install river
```



El entorno CartPole-v1

■ Materiales : Instalación de software

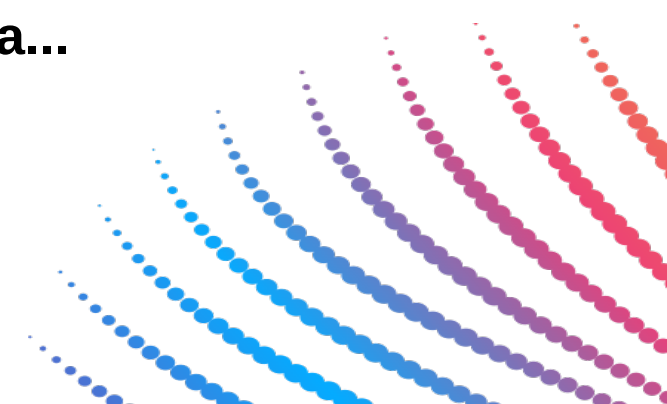
- **Gymnasium**: Fork de la biblioteca OpenAI Gym para **Reinforcement Learning**.



```
pip install gymnasium  
pip install pygame  
pip install gymnasium[classic-control]
```

¿Problemas de visualización en Ubuntu ? Intenta...

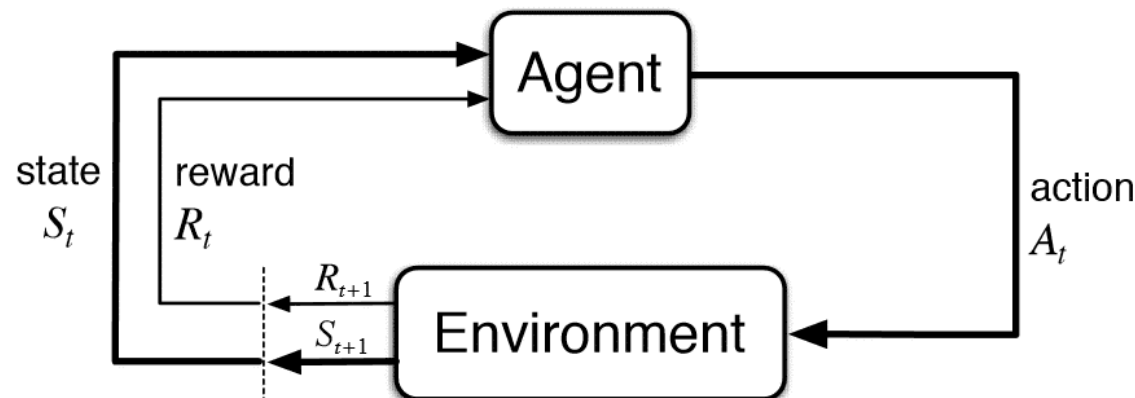
```
conda install -c conda-forge pyglet  
conda install -c conda-forge libstdcxx-ng
```



El entorno CartPole-v1

■ Gymnasium : Biblioteca para Aprendizaje por Refuerzo

- El sistema de interacción del modelo con el entorno es el siguiente :
 - **Agent** : El modelo entrenado
 - **Environment** : El entorno.
- En cada instante de tiempo t , el modelo puede **percibir** el estado del entorno $S(t)$ (posición, velocidad del carro y del poste).
- En base a estos valores, selecciona una **acción** $A(t)$ y la **ejecuta**.
- Se recibe una **recompensa** $R(t)$ por lo bien o mal que haya actuado.

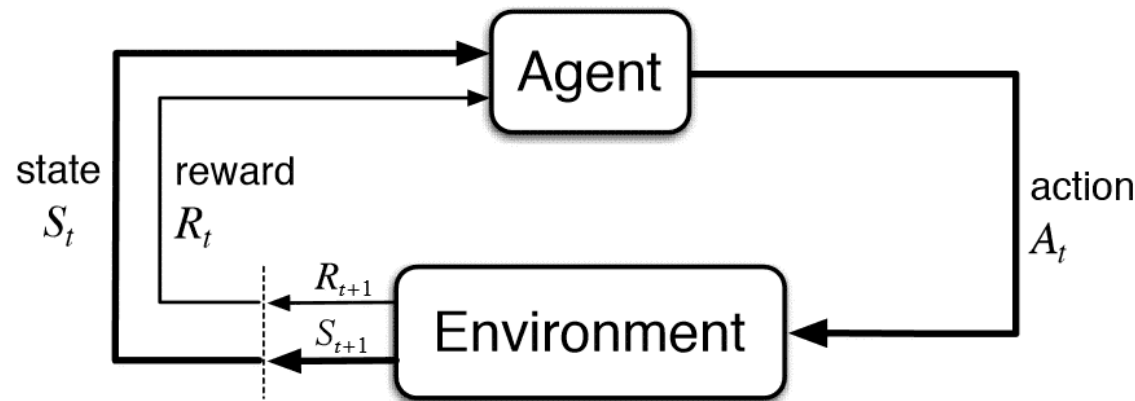


El entorno CartPole-v1

■ Gymnasium : Biblioteca para Aprendizaje por Refuerzo

- En definitiva, la interacción entre el agente y el entorno produce un **flujo de datos** del que se puede aprender : **Episodios / trayectorias**.
- Cada **episodio o trayectoria** está formado/a por una secuencia del tipo :

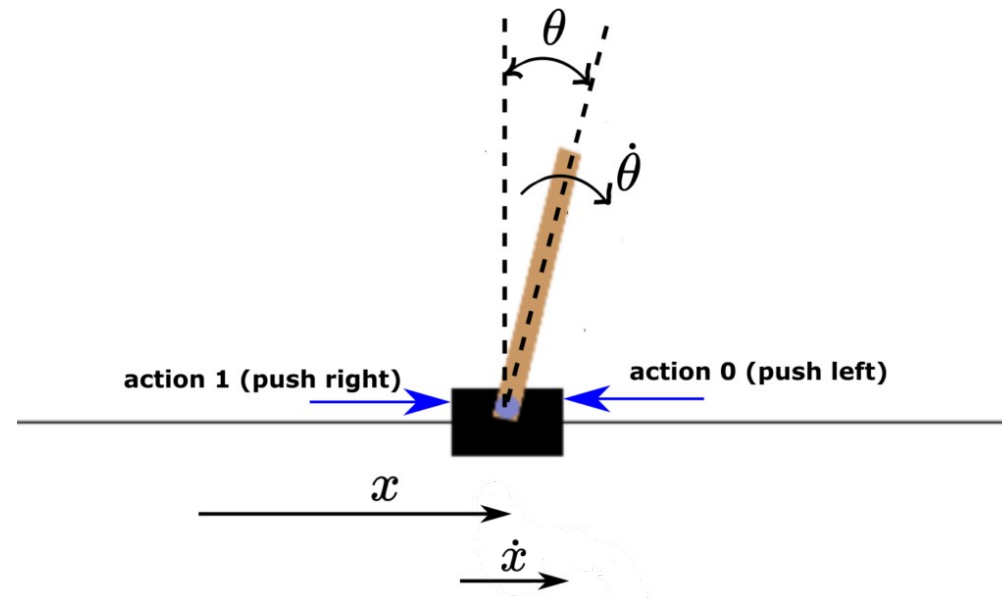
$(S(0), A(0), R(0), S(1), A(1), R(1), \dots, S(t), A(t), R(t), S(t+1), A(t+1), R(t+1), \dots)$



El entorno CartPole-v1

■ Descripción del problema

- Existe un carro (**cart**) que se puede desplazar lateralmente a izquierda y a derecha.
- El carro va equipado con un poste (**pole**), *que se encuentra en posición vertical*.
- **El problema consiste en evitar que el poste caiga** (en su defecto, que el ángulo con respecto a la vertical no sea superior a un valor dado).



El entorno CartPole-v1

■ Descripción : Datos

- Se nos proporciona un **flujo de datos** de un agente experto ejecutando una buena (pseudo-óptima) forma de resolver el problema (**fichero `CartPoleInstances.csv`**). **En total : 100.000 instancias de prueba.**
- **Cada instancia contiene los siguientes atributos:**
 - **Cart Position** : Numérico. Contiene la posición (**x**) del carro dentro de la vista del simulador. Valores >0 hacia la derecha, valores < 0 hacia la izquierda, valor=0 en el centro.
 - **Cart Velocity** : Numérico. Contiene la velocidad del carro (<0 hacia la izq ; >0 hacia la derecha ; 0= sin movimiento).
 - **Pole Angle** : Numérico. Contiene el ángulo del poste formado con respecto a la vertical (<0 inclinado a la izq ; >0 inclinado a la derecha).
 - **Pole Angular Velocity** : Numérico. Contiene la velocidad angular a la que se está moviendo el poste (<0 hacia la izq ; >0 hacia la dcha.).
 - **Action** : Acción ejecutada por el experto (0= mover a la izq ; 1= mover a la dcha.).

El entorno CartPole-v1

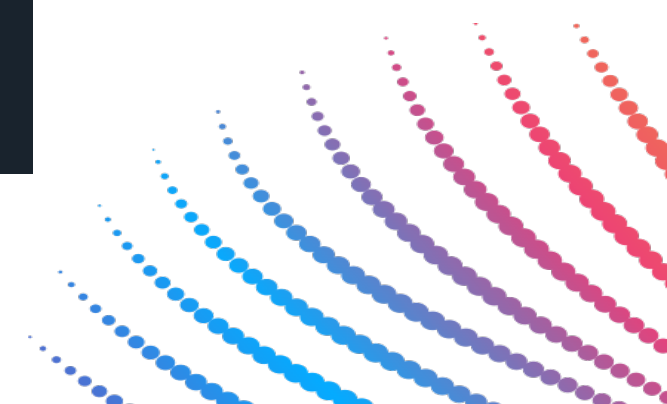
■ Descripción : Datos (ShowCartPoleFlow.py)

```
9  from river import stream
10
11
12  # Nombre del fichero de datos
13  datafile= 'CartPoleInstances.csv'
14
15
16  # Función que devuelve una lista con los nombres de los atributos de entrada
17  def AttributeNames():
18      return ['Cart Position', 'Cart Velocity', 'Pole Angle', 'Pole Angular Velocity']
19
20  # Función que devuelve el nombre del atributo objetivo (clase)
21  def ClassName():
22      return 'action'
23
24
25  # Conversores de datos para el flujo con iter_csv
26  converters= {}
27  for i in AttributeNames():
28      converters[i]= float
29  converters[ClassName()]= lambda x:int(float(x))
30
31
32
33  flujo= stream.iter_csv(datafile, converters= converters, target=ClassName())
34  for x,y in flujo:
35      print('-----')
36      print('Entradas:\n', x)
37      print('Salidas: ', y)
```


El entorno CartPole-v1

■ Descripción : Datos (ShowCartPoleFlow.py)

```
Console 1/A X
Entradas:
{'Cart Position': 0.0370326042175293, 'Cart Velocity': 0.18218320608139038,
'Pole Angle': 0.0018295111367478967, 'Pole Angular Velocity':
-0.3008362948894501}
Salidas: 0
-----
Entradas:
{'Cart Position': 0.040676265954971313, 'Cart Velocity':
-0.012964780442416668, 'Pole Angle': -0.004187214653939009, 'Pole Angular
Velocity': -0.007576943375170231}
Salidas: 0
-----
Entradas:
{'Cart Position': 0.04041697084903717, 'Cart Velocity':
-0.20802642405033112, 'Pole Angle': -0.004338753875344992, 'Pole Angular
Velocity': 0.2837819457054138}
Salidas: 1
```



El entorno CartPole-v1

■ Descripción : Materiales

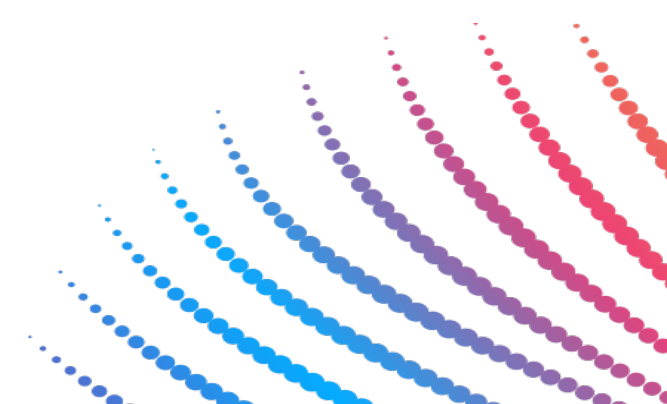
- Este problema, aunque es un ejemplo clásico del área del aprendizaje por refuerzo, puede resolverse con técnicas de Stream Mining basadas en ejemplos.
- **El objetivo es aprender un modelo automático en flujos de datos, basándonos en la experiencia de un agente experto (« copiar » lo que hace el agente).**
- Para poder validar el modelo obtenido, será necesario tener instalada la biblioteca de aprendizaje por refuerzo **gymnasium**.
- Dos formas de evaluación del modelo (se deben usar ambas):
 - ¿Cómo de similar es el comportamiento del modelo entrenado frente al experto ? → **Porcentaje de clasificación de acciones (Métrica Accuracy de la biblioteca river de Stream Mining).**
 - ¿Cómo de bueno es el modelo resolviendo el problema ? → **Métrica propia del problema CartPole-v1.**



El entorno CartPole-v1

■ Materiales : Métrica propia del problema CartPole-v1

- En nuestro caso, la recompensa será **+1 por cada instante de tiempo que el poste esté en alto.**
- **Objetivo :** Maximizar la **recompensa total obtenida** ; es decir, la suma de todas las recompensas en un episodio (**partida jugada**).
- **Máximo número de interacciones en test : 500** (por tanto, la máxima recompensa que podremos obtener es 500, sumando todas las recompensas obtenidas a lo largo de la interacción).



El entorno CartPole-v1

■ Ejemplo de interacción con el entorno: PlayCartPole.py

```
9  import gymnasium as gym
10 import numpy as np
11
12 # Nombre del entorno
13 envName= 'CartPole-v1'
14
15
16 # Función de selección de una acción a partir de la observación obs de entrada
17 def ActionSelection(obs):
18     action= np.random.randint(low=0, high=2) # Selección de acción 0/1 aleatoria
19     return action
20
21
22 # Creación del entorno
23 env= gym.make(envName, render_mode='human')
24
25
26 # Jugar un episodio
27 obs, _= env.reset() # Inicialización del entorno
28 env.render() # Renderizar entorno para visualización
29 R= 0 # Recompensa total obtenida
30 truncated, done= False, False
31 while not (truncated or done):
32
33     # Selección de acción
34     action= ActionSelection(obs)
35
36     # Ejecución de la acción y obtención de recompensa y siguiente estado
37     obs, r, done, truncated, _= env.step(action)
38     env.render() # Renderizar entorno para visualización
39     R+= r # Actualización de recompensa total (performance)
40
41 print('El episodio (partida) ha terminado con recompensa (performance)= {}'.format(R))
```

El entorno CartPole-v1

■ Ejemplo de interacción con el entorno: PlayCartPole.py

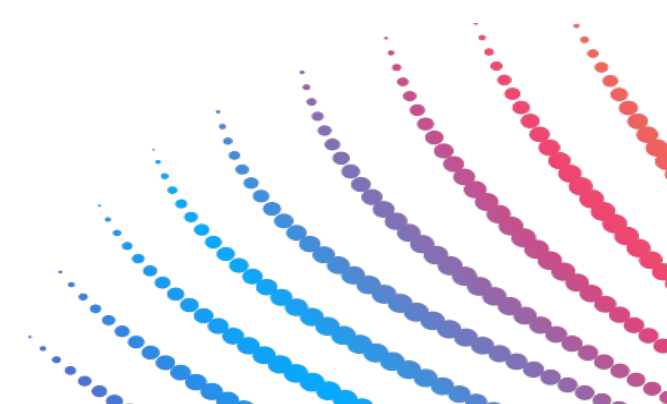
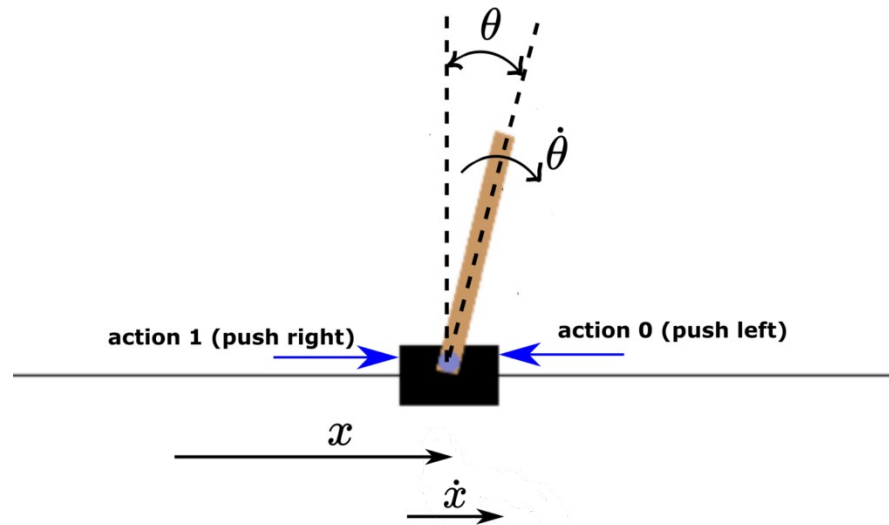
```
9 import gymnasium as gym
10 import numpy as np
11
12 # Nombre del entorno
13 envName= 'CartPole-v1'
14
15
16 # Función de selección de una acción a partir de la observación obs de entrada
17 def ActionSelection(obs):
18     action= np.random.randint(low=0, high=2) # Selección de acción 0/1 aleatoria
19     return action
20
21
22 # Creación del entorno
23 env= gym.make(envName, render_mode='human')
24
25
26 # Jugar un episodio
27 obs, _= env.reset() # Inicialización del entorno
28 env.render() # Renderizar entorno para visualización
29 R= 0 # Recompensa total obtenida
30 truncated, done= False, False
31 while not (truncated or done):
32
33     # Selección de acción
34     action= ActionSelection(obs)
35
36     # Ejecución de la acción y obtención de recompensa y siguiente estado
37     obs, r, done, truncated, _= env.step(action)
38     env.render() # Renderizar entorno para visualización
39     R+= r # Actualización de recompensa total (performance)
40
41 print('El episodio (partida) ha terminado con recompensa (performance)= {}'.format(R))
```

En la práctica, en lugar de esto se usarán modelos de Stream Mining para clasificación

El entorno CartPole-v1


■ Descripción : Tarea a realizar

- Problema :
- Diseñar modelos de stream mining para clasificación capaz de aprender lo mejor posible el comportamiento del experto, de modo que el modelo sea capaz de resolver el problema de la mejor forma posible.
- Puede hacer uso de todas (cualquiera de) las herramientas estudiadas en la asignatura (parte de MFD) para optimizar el comportamiento del clasificador.



El entorno CartPole-v1

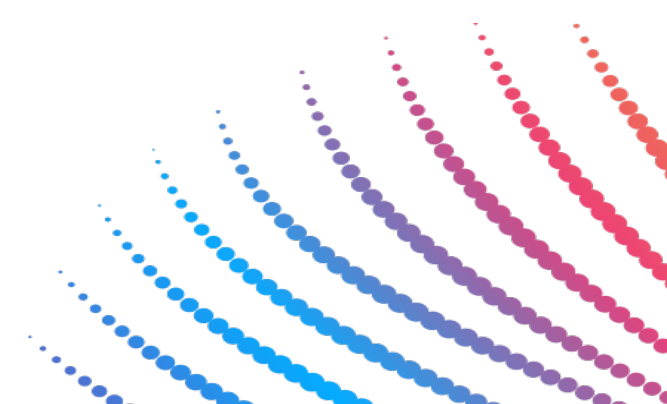
■ Descripción : Tarea a realizar

- Requisitos:
 - **Uno de los modelos** (se supone que el más básico) **deberá ser un Naive-Bayes Classifier Gaussiano**, donde cada clase tiene asociada una distribución gaussiana.
 - Se deberán implementar **al menos dos modelos más** entre los estudiados en teoría.
 - Cuando los modelos estén entrenados... :
 - Se deberá calcular la tasa de acierto (Accuracy) en train para cada uno, como mínimo 30 veces. Se deberá comprobar si hay diferencias significativas entre la tasa de clasificación de los modelos y, de ser el caso, hacer un ranking de mejor a peor de los mismos. **Sólo si es posible comparar.**
 - Se deberá calcular la recompensa total en 30 partidas (episodios) jugados por cada modelo en el entorno. Si hay diferencias significativas entre ellos, realizar un ranking de mejor a peor de los mismos.
- 

El entorno CartPole-v1

■ Descripción : Tarea a realizar

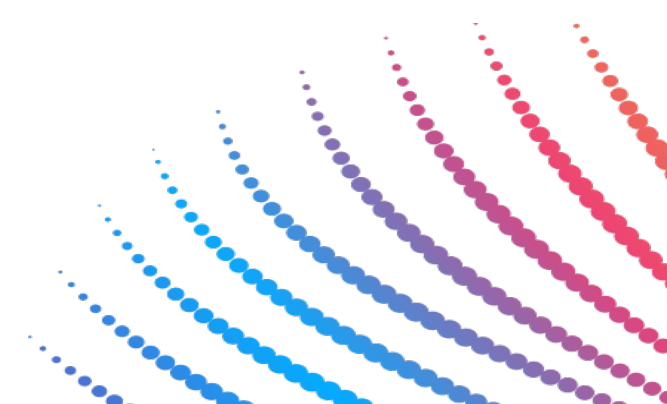
- Adicionalmente, se deberá explicar el estudio realizado, justificando cada tipo de análisis efectuado.
- Algunos análisis (ideas) pueden ir para responder preguntas tales como :
 - ¿Existe Concept Drift en los datos ?
 - ¿Qué tipo de preprocesamiento es necesario para el flujo dado ?
 - ¿Son mejores los modelos adaptativos o los no adaptativos para resolver este problema ?
 - ¿Proporciona algún tipo de ventajas el uso de técnicas como ADWIN o similares ?



El entorno CartPole-v1

■ Descripción : Tarea a realizar

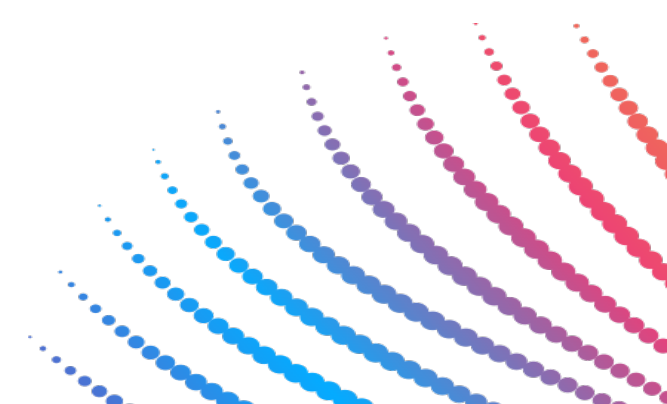
- **Resumiendo :**
 - Se debe tratar de resolver el problema de la mejor forma posible, tratando de aproximar el accuracy al 100 %, así como la recompensa total obtenida promedio al valor 500.
 - Se debe justificar los modelos escogidos con rigor, en base a análisis experimentales de los datos y/o del comportamiento de los modelos en test.
 - Se debe comparar diferentes modelos propuestos con rigor (tests estadísticos).
 - Se debe justificar la solución final planteada.



El entorno CartPole-v1

■ ¿Por dónde empezar ?

- Para la sesión de hoy, se podría...:
 - Familiarizar con la biblioteca **Gymnasium** y la interacción con el entorno.
 - Diseñar un script python que permita evaluar un modelo en **performance** (recompensa total en el entorno), y devuelva los resultados de un número determinado de experimentos.
 - Se puede basar en algún script proporcionado por el profesor.

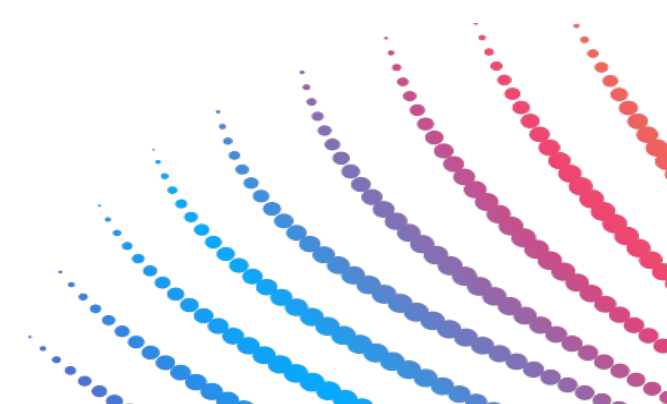



El entorno CartPole-v1

■ Reto: Implementación de la selección de acciones (hard-coded)

- ¿Serías capaz (sin mirar Internet) de modificar la función **ActionSelection** para que, en lugar de dar una acción aleatoria, proporcione una acción basada en la observación ?
- Y... (sin mirar Internet) ¿Podrías proponer una función de selección de acción que tenga un buen comportamiento (Recompensa Total > 400) ?

```
16 # Función de selección de una acción a partir de la observación obs de entrada
17 def ActionSelection(obs):
18     action= # TO-DO
19     return action
```





Series Temporales y Minería de Flujos de Datos

*Departamento de Ciencias de la Computación e
Inteligencia Artificial*

Universidad de Granada



**UNIVERSIDAD
DE GRANADA**

Prácticas de Series Temporales y Minería de Flujos de Datos:

Introducción

Prof. Manuel Pegalajar Cuéllar