

## 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:

*Stream Mining*

Prof. Manuel Pegalajar Cuéllar



- 01 ● River y lectura de datos
  - 02 ● Preprocesamiento
  - 03 ● Modelos y su evaluación
  - 04 ● Concept drift
  - 05 ● Ejemplo : Regresión en Stream Mining

# River y lectura de datos

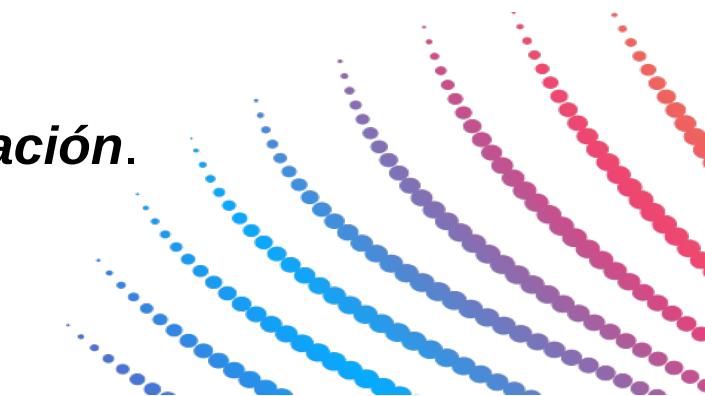


# River

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



- Se utiliza para creación de modelos de aprendizaje automático online, que operan sobre flujos de datos.
- Como **flujo de datos** entendemos una secuencia de elementos individuales, compuestos por varios (usualmente múltiples) atributos.
- A cada elemento individual se le denomina **muestra** u **observación**.



# River

- **Procesamiento online** : Se procesa una muestra (o conjunto de muestras) a la vez. Diferencia con Aprendizaje Automático tradicional : No se procesa por **batches**.
- **Evaluación de los modelos** :
  - En Aprendizaje Automático tradicional, se suele usar algún tipo de validación cruzada (entrenamiento + test).
  - En la evaluación de modelos online : entrenamiento y evaluación en línea.
- **Prequential** : Cada muestra se usa para evaluar el sistema. Acto seguido, se entrena con ella.
- **Heldout** : Se entrena con un número de muestras. Se reservan algunas para evaluar (lo más similar existente a evaluación train/test tradicional).

# River

- Tipos de flujos de datos :
  - **Reactivos** : Cada muestra del conjunto de datos llega al modelo sin control sobre ella (por ejemplo, clics en páginas web).
  - **Proactivos** : Existe un cierto control sobre el flujo de datos y cuándo se obtienen muestras (por ejemplo, lectura desde un fichero).
- Conjuntos de datos en ***river*** : Basados en diccionarios (pares clave-valor). Cada muestra se representa como un diccionario **{atributo : valor}** .

```
{  
    'empty_server_form_handler': 0.0,  
    'popup_window': 0.0,  
    'https': 0.0,  
    'request_from_other_domain': 0.0,  
    'anchor_from_other_domain': 0.0,  
    'is_popular': 0.5,  
    'long_url': 1.0,  
    'age_of_domain': 1,  
    'ip_in_url': 1  
}
```



# Lectura de datos

- El tipo de representación de datos en river es **basado en diccionarios**.
- Ventajas del tipo *dict* :
  - Reduce el tiempo de acceso a valores de atributos (por ejemplo, acceder a una celda de un Dataframe, un elemento de una lista, etc.).
  - Habilita la posibilidad de dar nombres a los atributos (no es posible con listas o tuplas).
  - Soportado de forma nativa en Python.

```
{  
    'empty_server_form_handler': 0.0,  
    'popup_window': 0.0,  
    'https': 0.0,  
    'request_from_other_domain': 0.0,  
    'anchor_from_other_domain': 0.0,  
    'is_popular': 0.5,  
    'long_url': 1.0,  
    'age_of_domain': 1,  
    'ip_in_url': 1  
}
```

# Lectura de datos

- River está preparado para afrontar diferentes tareas en stream mining, y proporciona algunos conjuntos de datos para realización de pruebas. **Ejemplos :**
  - **Clasificación binaria :**
    - **Dataset Bananas** : Dataset artificial donde las instancias corresponden a varios clusters con forma de banana (5300 muestras, 2 atributos).
    - **Dataset Phising** : Dataset que contiene características de páginas web clasificadas como phising (verdadero o falso). 1250 muestras, 9 atributos.
  - **Clasificación con múltiples clases :**
    - **Dataset Insects** : Varias variantes con distinta complejidad. Normalmente 52.848 muestras con 33 atributos, 6 clases).
    - **Dataset Keystroke** : Contiene datos de usuarios tecleando una clave, y el objetivo es determinar qué usuario está escribiendo. 20.400 muestras con 31 atributos.
  - **Regresión :**
    - **Dataset TrumpApproval** : Análisis de tasas de aprobación de leyes de Donald Trump. Contiene 1001 muestras con 5 atributos + la tasa de aprobación dada por **FiveThirtyEight**.

# Lectura de datos

## Notebook: *LecturaDeDatos.ipynb*

### Imports para cargar datasets existentes en *River*

```
In [1]: from river import datasets
```

### Lectura de un dataset

**Dataset *Bikes*:** Contiene datos sobre el número de bicicletas en 5 puntos de alquiler de bicicletas de la ciudad de Toulouse. El objetivo es predecir cuántas bicicletas habrá por estación disponibles. Hay 182.470 muestras.

```
In [2]:
```

```
data= datasets.Bikes() # Carga del dataset  
  
# Mostramos información del dataset  
print(data)
```

Bike sharing station information from the city of Toulouse.

The goal is to predict the number of bikes in 5 different bike stations from the city of Toulouse.

Name	Bikes
Task	Regression
Samples	182,470
Features	8
Sparse	False
Path	/home/manupc/river_data/Bikes/toulouse_bikes.csv
URL	<a href="https://maxhalford.github.io/files/datasets/toulouse_bikes.zip">https://maxhalford.github.io/files/datasets/toulouse_bikes.zip</a>
Size	12.52 MB
Downloaded	True



# Lectura de datos

## Notebook (continuación): *LecturaDeDatos.ipynb*

### Cada muestra es un diccionario

En aprendizaje supervisado, se tiene que cada muestra está compuesta por un par  $(x,y)$ , donde  $x$  es la entrada e  $y$  la salida esperada.

Podemos iterar sobre el dataset viéndolo como un iterador

```
In [3]:  
data_iter= iter(data) # El dataset como elemento iterable  
  
x, y= next(data_iter) # Siguiente elemento del conjunto  
print('Entrada: \n{}\n\nSalida: {}'.format(x, y))  
print('El valor del atributo clouds es: {}'.format(x['clouds']))  
  
x, y= next(data_iter) # Siguiente elemento del conjunto  
print('\nEntrada: \n{}\n\nSalida: \n{}'.format(x, y))  
  
Entrada:  
{'moment': datetime.datetime(2016, 4, 1, 0, 0, 7), 'station': 'metro-canal-du-midi', 'clouds': 75, 'description': 'light rain', 'humidity': 81, 'pressure': 1017.0, 'temperature': 6.54, 'wind': 9.3}  
  
Salida: 1  
El valor del atributo clouds es: 75  
  
Entrada:  
{'moment': datetime.datetime(2016, 4, 1, 0, 0, 16), 'station': 'place-des-carmes', 'clouds': 75, 'description': 'light rain', 'humidity': 81, 'pressure': 1017.0, 'temperature': 6.54, 'wind': 9.3}  
  
Salida:  
3
```

### Iteración sobre todo el dataset

```
In [4]:  
num_muestras= 0  
  
for x,y in data:  
    num_muestras+= 1  
  
print('El dataset contiene {} muestras'.format(num_muestras))
```

El dataset contiene 182470 muestras



# Lectura de datos

## ■ Conversión de datasets clásicos a streams: *LecturaDeDatos.ipynb*

### Lectura de datos externos y conversión a *stream*

Un conjunto de datos ya existente lo podemos transformar a un flujo usando **streams** de River. Por ejemplo, el dataset de SKLearn sobre cáncer de mama.

```
In [5]: from sklearn import datasets as sk_datasets # Datasets de SKLearn

# Carga del conjunto de datos
sk_data= sk_datasets.load_breast_cancer()
print(sk_data['DESCR']) # Imprimimos descripción del dataset

# Obtenemos entradas/salidas
X, Y= sk_data.data, sk_data.target
print('El conjunto de datos contiene {} patrones E/S'.format(len(X)))

... _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
-----
**Data Set Characteristics:**

:Number of Instances: 569

:Number of Attributes: 30 numeric, predictive attributes and the class

:Attribute Information:
- radius (mean of distances from center to points on the perimeter)
- texture (standard deviation of gray-scale values)
- perimeter
- area
- smoothness (local variation in radius lengths)
- compactness (perimeter2 / area - 1.0)
- concavity (severity of concave portions of the contour)
```



# Lectura de datos

## Conversion de datasets clásicos a streams: LecturaDeDatos.ipynb

In [6]:

```
from river import stream # Clase para modelar streams

rv_data= stream.iter_sklearn_dataset(sk_data) # Conversión del dataset a stream

for river_sample, sklearn_sample in zip(rv_data, zip(X, Y)): # Recorremos uno por uno los patrones en ambos datasets

    # Vemos que se obtienen las mismas entradas y salidas en ambos sistemas.
    # No obstante, en River, los datos se organizan como diccionarios
    print('Dato River entrada: {}'.format(river_sample[0]))
    print('Dato SKLearn entrada: {}'.format(sklearn_sample[0]))
    print('Dato River salida: {}'.format(river_sample[1]))
    print('Dato SKLearn salida: {}'.format(sklearn_sample[1]))
```

```
Dato River entrada: {'mean radius': 17.99, 'mean texture': 10.38, 'mean perimeter': 122.8, 'mean area': 1001.0, 'mean smoothness': 0.1184, 'mean compactness': 0.2776, 'mean concavity': 0.3001, 'mean concave points': 0.1471, 'mean symmetry': 0.2419, 'mean fractal dimension': 0.07871, 'radius error': 1.095, 'texture error': 0.9053, 'perimeter error': 8.589, 'area error': 153.4, 'smoothness error': 0.006399, 'compactness error': 0.04904, 'concavity error': 0.05373, 'concave points error': 0.01587, 'symmetry error': 0.03003, 'fractal dimension error': 0.006193, 'worst radius': 25.38, 'worst texture': 17.33, 'worst perimeter': 184.6, 'worst area': 2019.0, 'worst smoothness': 0.1622, 'worst compactness': 0.6656, 'worst concavity': 0.7119, 'worst concave points': 0.2654, 'worst symmetry': 0.4601, 'worst fractal dimension': 0.1189}
Dato SKLearn entrada: [1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01
1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02
6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01
1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01
4.601e-01 1.189e-01]
Dato River salida: 0
Dato SKLearn salida: 0
Dato River entrada: {'mean radius': 20.57, 'mean texture': 17.77, 'mean perimeter': 132.9, 'mean area': 1326.0, 'mean smoothness': 0.09474, 'mean compactness': 0.07964, 'mean concavity': 0.0869, 'mean concave points': 0.07017, 'mean symmetry': 0.2059, 'mean fractal dimension': 0.07871, 'radius error': 1.12, 'texture error': 0.9053, 'perimeter error': 8.589, 'area error': 153.4, 'smoothness error': 0.006399, 'compactness error': 0.04904, 'concavity error': 0.05373, 'concave points error': 0.01587, 'symmetry error': 0.03003, 'fractal dimension error': 0.006193, 'worst radius': 25.38, 'worst texture': 17.33, 'worst perimeter': 184.6, 'worst area': 2019.0, 'worst smoothness': 0.1622, 'worst compactness': 0.6656, 'worst concavity': 0.7119, 'worst concave points': 0.2654, 'worst symmetry': 0.4601, 'worst fractal dimension': 0.1189}
```

# Lectura de datos

## Lectura de datos desde fichero: *LecturaDeDatos.ipynb*

### iter\_csv

Iterates over rows from a CSV file.

Reading CSV files can be quite slow. If, for whatever reason, you're going to loop through the same file multiple times, then we recommend that you use the `stream.Cache` utility.

#### Parameters

---

- **filepath\_or\_buffer**

Either a string indicating the location of a file, or a buffer object that has a `read` method.

- **target** (`Union[str, List[str]]`) – defaults to `None`

A single target column is assumed if a string is passed. A multiple output scenario is assumed if a list of strings is passed. A `None` value will be assigned to each `y` if this parameter is omitted.

- **converters** (`dict`) – defaults to `None`

All values in the CSV are interpreted as strings by default. You can use this parameter to cast values to the desired type. This should be a `dict` mapping feature names to callables used to parse their associated values. Note that a callable may be a type, such as `float` and `int`.

- **parse\_dates** (`dict`) – defaults to `None`

A `dict` mapping feature names to a format passed to the `datetime.datetime.strptime` method.

- **drop** (`List[str]`) – defaults to `None`

Fields to ignore.

- **drop\_nones** – defaults to `False`

Whether or not to drop fields where the value is a `None`.

- **fraction** – defaults to `1.0`

Sampling fraction.



# Lectura de datos

## Lectura de datos desde fichero: *LecturaDeDatos.ipynb*

### Ejemplo de carga de un flujo desde fichero .csv

In [9]:

```
# Leeremos el dataset de iris, que se encuentra en el fichero iris.csv
from river import stream
flujo= stream.iter_csv('iris.csv', target='label') # Cargamos fichero identificando columna target
for x,y in flujo:
    print('Entrada: ')
    print(x)
    print('Salida: ')
    print(y)
    print('---')
```

Entrada:  
{'Sepal-length': '5.1', 'Sepal-width': '3.5', 'Petal-length': '1.4', 'Petal-width': '0.2'}  
Salida:  
Iris-setosa  
---  
Entrada:  
{'Sepal-length': '4.9', 'Sepal-width': '3.0', 'Petal-length': '1.4', 'Petal-width': '0.2'}  
Salida:  
Iris-setosa  
---  
Entrada:  
{'Sepal-length': '4.7', 'Sepal-width': '3.2', 'Petal-length': '1.3', 'Petal-width': '0.2'}  
Salida:  
Iris-setosa  
---  
Entrada:  
{'Sepal-length': '4.6', 'Sepal-width': '3.1', 'Petal-length': '1.5', 'Petal-width': '0.2'}  
Salida:  
Iris-setosa



# Lectura de datos

## ■ Problemas de los datasets de prueba

- Son conjuntos de datos que, por lo general tienen pocas muestras.
- Por lo general, obtenidos desde datasets clásicos para Aprendizaje Automático tradicional.
- Se pierde la naturaleza de flujo de datos continuo (y posiblemente infinito).
- **Solución :** Crear conjuntos de datos sintéticos.



# Lectura de datos

## Conjuntos de datos sintéticos

- Son conjuntos de datos que se usan ampliamente en experimentación de algoritmos para stream mining, debido a que se pueden generar tantas muestras como se desee.
- La API de River dispone de los más comunes :

synth	
Agrawal	Mixed
AnomalySine	Mv
ConceptDriftStream	Planes2D
Friedman	RandomRBF
FriedmanDrift	RandomRBFDrift
Hyperplane	RandomTree
LED	SEA
LEDDrift	STAGGER
Logical	Sine



# Lectura de datos

## Conjuntos de datos sintéticos : Agrawal

### Agrawal

Agrawal stream generator.

The generator was introduced by Agrawal et al.<sup>1</sup>, and was a common source of data for early work on scaling up decision tree learners. The generator produces a stream containing nine features, six numeric and three categorical. There are 10 functions defined for generating binary class labels from the features. Presumably these determine whether the loan should be approved. Classification functions are listed in the original paper<sup>1</sup>.

#### Feature | Description | Values

- `salary` | salary | uniformly distributed from 20k to 150k
- `commission` | commission | 0 if `salary` < 75k else uniformly distributed from 10k to 75k
- `age` | age | uniformly distributed from 20 to 80
- `elevel` | education level | uniformly chosen from 0 to 4
- `car` | car maker | uniformly chosen from 1 to 20
- `zipcode` | zip code of the town | uniformly chosen from 0 to 8
- `hvalue` | house value | uniformly distributed from 50k x zipcode to 100k x zipcode
- `hyears` | years house owned | uniformly distributed from 1 to 30
- `loan` | total loan amount | uniformly distributed from 0 to 500k



# Lectura de datos

## ■ Conjuntos de datos sintéticos : RandomRBF

### RandomRBF

Random Radial Basis Function generator.

Produces a radial basis function stream. A number of centroids, having a random central position, a standard deviation, a class label and weight are generated. A new sample is created by choosing one of the centroids at random, taking into account their weights, and offsetting the attributes in a random direction from the centroid's center. The offset length is drawn from a Gaussian distribution.

This process will create a normally distributed hypersphere of samples on the surrounds of each centroid.



# Lectura de datos

## ■ Conjuntos de datos sintéticos : RandomRBFDrift

### RandomRBFDrift

Random Radial Basis Function generator with concept drift.

This class is an extension from the `RandomRBF` generator. Concept drift can be introduced in instances of this class.

The drift is created by adding a "speed" to certain centroids. As the samples are generated each of the moving centroids' centers is changed by an amount determined by its speed.



# Lectura de datos

## Conjuntos de datos sintéticos : Ejemplo con Agrawal en *LecturaDeDatos.ipynb*

### Parameters

---

- **classification\_function** ('int') – defaults to `0`

The classification function to use for the generation. Valid values are from 0 to 9.

- **seed** ('Optional[int / np.random.RandomState]') – defaults to `None`

If int, `seed` is used to seed the random number generator; If RandomState instance, `seed` is the random number generator; If None, the random number generator is the RandomState instance used by `np.random`.

- **balance\_classes** ('bool') – defaults to `False`

If True, the class distribution will converge to a uniform distribution.

- **perturbation** ('float') – defaults to `0.0`

The probability that noise will happen in the generation. Each new sample will be perturbed by the magnitude of `perturbation`. Valid values are in the range [0.0 to 1.0].



# Lectura de datos

## Conjuntos de datos sintéticos : Ejemplo en *LecturaDeDatos.ipynb*

### Ejemplo de carga de conjunto de datos de Agrawal

In [9]:

```
agrwal_dataset= datasets.synth.Agrawal(classification_function= 0, # Carga del dataset de Agrawal  
                                         seed= 1) # Es importante establecer semilla aleatoria por reproducibilidad
```

```
print(agrwal_dataset)
```

```
x, y= next(iter(agrwal_dataset))
```

```
print('Entrada: ')
```

```
print(x)
```

```
print('Salida: ')
```

```
print(y)
```

```
Synthetic data generator
```

```
Name Agrawal
```

```
Task Binary classification
```

```
Samples ∞
```

```
Features 9
```

```
Outputs 1
```

```
Classes 2
```

```
Sparse False
```

```
Configuration
```

```
-----
```

```
classification_function 0
```

```
          seed 1
```

```
balance_classes False
```

```
perturbation 0.0
```

```
Entrada:
```

```
{'salary': 74212.86061133462, 'commission': 64024.337008161856, 'age': 29, 'elevel': 3, 'car': 5, 'zipcode': 0, 'hva  
lue': 533104.735291918, 'hyears': 13, 'loan': 193955.3705810037}
```

```
Salida:
```

```
1
```

# Preprocesamiento



# Preprocesamiento

## ■ Preprocesamiento en Stream Mining

- El preprocesamiento de los datos puede diferir sustancialmente en los casos de *stream mining* con respecto al caso clásico.
- Ejemplo (**Preprocesamiento.ipynb**) : Una simple estandarización requiere conocer el dataset completo para calcular media y desviación típica.

### Estandarización en el caso clásico

```
In [1]: from sklearn.preprocessing import StandardScaler # Normalizador de datos
         from sklearn import datasets as sk_datasets # Datasets de SKLearn

         # Carga del conjunto de datos
         sk_data= sk_datasets.load_breast_cancer()
         X, Y= sk_data.data, sk_data.target

         InputScaler= StandardScaler() # Objeto para realizar la normalización
         InputScaler.fit(X) # Calculamos media y desviación típica para las entradas del dataset
         X_prime= InputScaler.transform(X)

         for i in range(3):
             print('Entrada original: {}'.format(X[i]))
             print('Entrada normalizada: {}'.format(X_prime[i]))
```

Entrada original: [1.799e+01 1.038e+01 1.228e+02 1.001e+03 1.184e-01 2.776e-01 3.001e-01  
1.471e-01 2.419e-01 7.871e-02 1.095e+00 9.053e-01 8.589e+00 1.534e+02  
6.399e-03 4.904e-02 5.373e-02 1.587e-02 3.003e-02 6.193e-03 2.538e+01  
1.733e+01 1.846e+02 2.019e+03 1.622e-01 6.656e-01 7.119e-01 2.654e-01  
4.601e-01 1.189e-01]  
Entrada normalizada: [ 1.09706398 -2.07333501 1.26993369 0.9843749 1.56846633 3.28351467  
2.65287398 2.53247522 2.21751501 2.25574689 2.48973393 -0.56526506  
2.83303087 2.48757756 -0.21400165 1.31686157 0.72402616 0.66081994  
1.14875667 0.90708308 1.88668963 -1.35929347 2.30360062 2.00123749  
1.30768627 2.61666502 2.10952635 2.29607613 2.75062224 1.93701461]



# Preprocesamiento

## ■ Preprocesamiento en Stream Mining

- En el caso online, no disponemos de todo el dataset.
- Se debe ir actualizando los estadísticos de interés de forma también online :
  - **La media :**

$$\begin{cases} n_{t+1} = n_t + 1 \\ \mu_{t+1} = \mu_t + \frac{x - \mu_t}{n_{t+1}} \end{cases}$$

- **La desviación típica :**

$$\begin{cases} n_{t+1} = n_t + 1 \\ \mu_{t+1} = \mu_t + \frac{x - \mu_t}{n_{t+1}} \\ s_{t+1} = s_t + (x - \mu_t) \times (x - \mu_{t+1}) \\ \sigma_{t+1} = \frac{s_{t+1}}{n_{t+1}} \end{cases}$$



# Preprocesamiento

## ■ Preprocesamiento en Stream Mining : CUIDADO con el aprendizaje preprocesado desde el principio

Se debería dejar aprender el Scaler antes de ejecutar modelos

### Estandarización en el caso online

```
In [3]: from river.preprocessing import StandardScaler # Normalizador de datos ONLINE
from river import stream

rv_scaler= StandardScaler()
rv_data= stream.iter_sklearn_dataset(sk_data)
for i in range(3):
    sample_x, sample_y= next(rv_data)
    rv_scaler= rv_scaler.learn_one(sample_x) # Actualizamos normalizador online
    sample_x_prime= rv_scaler.transform_one(sample_x) # Normalizamos
    print('Entrada original: {}'.format(sample_x))
    print('Entrada normalizada: {}'.format(sample_x_prime))
```

```
Entrada original: {'mean radius': 17.99, 'mean texture': 10.38, 'mean perimeter': 122.8, 'mean area': 1001.0, 'mean smoothness': 0.1184, 'mean compactness': 0.2776, 'mean concavity': 0.3001, 'mean concave points': 0.1471, 'mean symmetry': 0.2419, 'mean fractal dimension': 0.07871, 'radius error': 1.095, 'texture error': 0.9053, 'perimeter error': 8.589, 'area error': 153.4, 'smoothness error': 0.006399, 'compactness error': 0.04904, 'concavity error': 0.05373, 'concave points error': 0.01587, 'symmetry error': 0.03003, 'fractal dimension error': 0.006193, 'worst radius': 25.38, 'worst texture': 17.33, 'worst perimeter': 184.6, 'worst area': 2019.0, 'worst smoothness': 0.1622, 'worst compactness': 0.6656, 'worst concavity': 0.7119, 'worst concave points': 0.2654, 'worst symmetry': 0.4601, 'worst fractal dimension': 0.1189}
```

```
Entrada normalizada: {'mean radius': 0.0, 'mean texture': 0.0, 'mean perimeter': 0.0, 'mean area': 0.0, 'mean smoothness': 0.0, 'mean compactness': 0.0, 'mean concavity': 0.0, 'mean concave points': 0.0, 'mean symmetry': 0.0, 'mean fractal dimension': 0.0, 'radius error': 0.0, 'texture error': 0.0, 'perimeter error': 0.0, 'area error': 0.0, 'smoothness error': 0.0, 'compactness error': 0.0, 'concavity error': 0.0, 'concave points error': 0.0, 'symmetry error': 0.0, 'fractal dimension error': 0.0, 'worst radius': 0.0, 'worst texture': 0.0, 'worst perimeter': 0.0, 'worst area': 0.0, 'worst smoothness': 0.0, 'worst compactness': 0.0, 'worst concavity': 0.0, 'worst concave points': 0.0, 'worst symmetry': 0.0, 'worst fractal dimension': 0.0}
```

# Preprocesamiento

## ■ Tipos de preprocesamiento incluidos en River

- Los más habituales : Cambios de escala, codificación OneHot, imputación de valores perdidos. Todos basados en SKLearn.

AdaptiveStandardScaler

Binarizer

FeatureHasher

LDA

MaxAbsScaler

MinMaxScaler

Normalizer

OneHotEncoder

PredClipper

PreviousImputer

RobustScaler

StandardScaler

StatImputer

TargetStandardScaler



# Modelos y su evaluación



# Modelos

## ■ Modelos existentes

- River incluye algunos de los más usados para Stream Mining :
  - Naïve-Bayes
  - Clasificadores K-NN
  - Perceptrones Multi-capas para regresión
  - Adaptive Model Rules para regresión
  - SNARIMAX (Seasonal, Non-linear, ARIMA, Exogenous variables)
  - Hoeffding (Adaptive) trees
  - Stochastic Gradient trees
  - Regresión lineal
  - ...
  - Etc.



# Modelos

## ■ Modelos existentes : Naïve-Bayes classifier

### GaussianNB

Gaussian Naive Bayes.

A Gaussian distribution  $G_{cf}$  is maintained for each class  $c$  and each feature  $f$ . Each Gaussian is updated using the amount associated with each feature; the details can be found in `proba.Gaussian`. The joint log-likelihood is then obtained by summing the log probabilities of each feature associated with each class.

### MultinomialNB

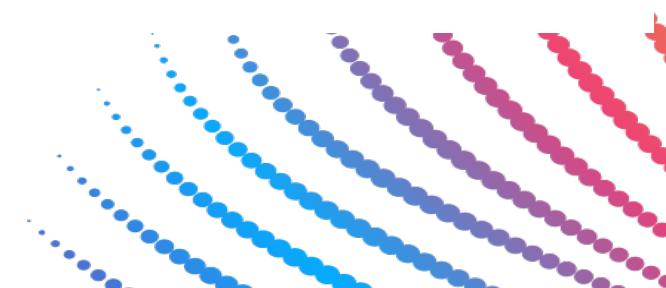
### BernoulliNB

Bernoulli Naive Bayes.

Bernoulli Naive Bayes model learns from occurrences between features such as word counts and discrete classes. The input vector must contain positive values, such as counts or TF-IDF values.

Naive Bayes classifier for multinomial models.

Multinomial Naive Bayes model learns from occurrences between features such as word counts and discrete classes. The input vector must contain positive values, such as counts or TF-IDF values.



# Modelos

## ■ Modelos existentes : Hoeffding tree classifier y regressor

### **HoeffdingTreeClassifier**

Hoeffding Tree or Very Fast Decision Tree classifier.

### **HoeffdingAdaptiveTreeClassifier**

Hoeffding Adaptive Tree classifier.

### **HoeffdingAdaptiveTreeRegressor**

Hoeffding Adaptive Tree regressor (HATR).

This class implements a regression version of the Hoeffding Adaptive Tree Classifier. Hence, it also uses an ADWIN concept-drift detector instance at each decision node to monitor possible changes in the data distribution. If a drift is detected in a node, an alternate tree begins to be induced in the background. When enough information is gathered, HATR swaps the node where the change was detected by its alternate tree.

### **HoeffdingTreeRegressor**

Hoeffding Tree regressor.



# Modelos

## ■ Modelos existentes : Regresión lineal y redes neuronales MLP

### LinearRegression

Linear regression.

This estimator supports learning with mini-batches. On top of the single instance methods, it provides the following methods: `learn_many`, `predict_many`, `predict_proba_many`. Each method takes as input a `pandas.DataFrame` where each column represents a feature.

It is generally a good idea to scale the data beforehand in order for the optimizer to converge. You can do this online with a `preprocessing.StandardScaler`.

### BayesianLinearRegression

Bayesian linear regression.

An advantage of Bayesian linear regression over standard linear regression is that features do not have to scaled beforehand. Another attractive property is that this flavor of linear regression is somewhat insensitive to its hyperparameters. Finally, this model can output instead a predictive distribution rather than just a point estimate.

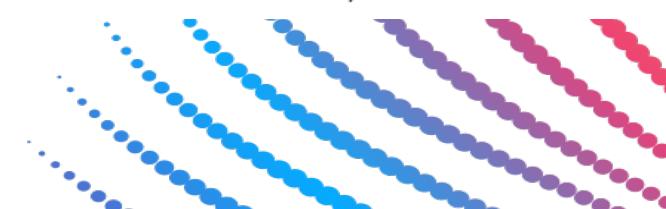
The downside is that the learning step runs in  $O(n^2)$  time, whereas the learning step of standard linear regression takes  $O(n)$  time.

### MLPRegressor

Multi-layer Perceptron for regression.

This model is still work in progress. Here are some features that still need implementing:

- `learn_one` and `predict_one` just cast the input `dict` to a single row dataframe and then call `learn_many` and `predict_many` respectively. This is very inefficient. - Not all of the optimizers in the `optim` module can be used as they are not all vectorised. - Emerging and disappearing features are not supported. Each instance/batch has to have the same features. - The gradient haven't been numerically checked.



# Modelos

## ■ Modelos existentes : Ejemplo de clasificación logística en *Modelos.ipynb*

### Lectura del dataset Phishing

Detección de si una web es confiable o no a través de 9 atributos

```
In [1]: from river import datasets

dataset= datasets.Phishing()
print(dataset)

x, y= next(iter(dataset))
print('\nMuestra (entrada):\n', x, '\nSalida:\n', y)
```

Phishing websites.

This dataset contains features from web pages that are classified as phishing or not.

```
Name  Phishing
Task  Binary classification
Samples 1,250
Features 9
Sparse  False
Path  /home/manupc/anaconda3/envs/STMFD/lib/python3.9/site-packages/river/datasets/phishing.csv.gz
```

```
Muestra (entrada):
{'empty_server_form_handler': 0.0, 'popup_window': 0.0, 'https': 0.0, 'request_from_other_domain': 0.0, 'anchor_fro
m_other_domain': 0.0, 'is_popular': 0.5, 'long_url': 1.0, 'age_of_domain': 1, 'ip_in_url': 1}
Salida:
True
```

# Modelos

## ■ Modelos existentes : Ejemplo de clasificación logística en *Modelos.ipynb*

### Creación del modelo

```
In [2]: from river.linear_model import LogisticRegression # Importamos la regresión logística  
model = LogisticRegression() # Creación del modelo  
  
out= model.predict_proba_one(x) # Predecimos etiqueta y probabilidad (resultado: diccionario)  
print('Predicción: ', out)
```

Predicción: {False: 0.5, True: 0.5}

### Aprendizaje de una muestra

```
In [3]: model = model.learn_one(x, y) # Aprendemos de la muestra y mostramos de nuevo resultados  
  
out= model.predict_proba_one(x) # Predecimos etiqueta y probabilidad (resultado: diccionario)  
print('Predicción: ', out)  
  
out= model.predict_one(x) # Si sólo queremos la clase más probable  
print('Predicción: ', out)
```

Predicción: {False: 0.494687699901455, True: 0.505312300098545}  
Predicción: True



# Modelos

## ■ Modelos existentes : Ejemplo de clasificación logística en *Modelos.ipynb*

### Parameters

- **optimizer** (*optim.base.Optimizer*) – defaults to `None`

The sequential optimizer used for updating the weights. Note that the intercept is handled separately.

- **loss** (*optim.losses.BinaryLoss*) – defaults to `None`

The loss function to optimize for. Defaults to `optim.losses.Log`.

- **l2** – defaults to `0.0`

Amount of L2 regularization used to push weights towards 0. For now, only one type of penalty can be used. The joint use of L1 and L2 is not explicitly supported.

- **l1** – defaults to `0.0`

Amount of L1 regularization used to push weights towards 0. For now, only one type of penalty can be used. The joint use of L1 and L2 is not explicitly supported.

- **intercept\_init** – defaults to `0.0`

Initial intercept value.

- **intercept\_lr** (*Union[float, optim.base.Scheduler]*) – defaults to `0.01`

Learning rate scheduler used for updating the intercept. A `optim.schedulers.Constant` is used if a `float` is provided. The intercept is not updated when this is set to 0.

- **clip\_gradient** – defaults to `1000000000000.0`

Clips the absolute value of each gradient value.

- **initializer** (*optim.base.Initializer*) – defaults to `None`

Weights initialization scheme.

- Siempre conviene mirar la API para conocer qué parámetros necesita cada modelo y cuáles son sus valores por defecto.



# Evaluación

## Evaluación de modelos : métricas

- El aprendizaje y el desempeño de los modelos debe ser continuamente evaluado.
- En cada caso, se debe escoger la métrica (o conjunto de métricas) que mejor se ajuste al problema y los objetivos de la experimentación.
- River contiene múltiples métricas a disposición del programador.

metrics	FBeta	MacroJaccard	Precision	WeightedFBeta
Accuracy	FowlkesMallows	MacroPrecision	R2	WeightedJaccard
AdjustedMutualInfo	GeometricMean	MacroRecall	RMSE	WeightedPrecision
AdjustedRand	Homogeneity	MicroF1	RMSLE	WeightedRecall
BalancedAccuracy	Jaccard	MicroFBeta	ROCAUC	base
ClassificationReport	LogLoss	MicroJaccard	Rand	BinaryMetric
CohenKappa	MAE	MicroPrecision	Recall	ClassificationMetric
Completeness	MCC	MicroRecall	SMAPE	Metric
ConfusionMatrix	MSE	MultiFBeta	Silhouette	Metrics
CrossEntropy	MacroF1	MutualInfo	VBeta	MultiClassMetric
F1	MacroFBeta	NormalizedMutualInfo	WeightedF1	RegressionMetric
				WrapperMetric
				multioutput
				ExactMatch
				MacroAverage
				MicroAverage
				MultiLabelConfusionMatrix
				PerOutput

# Evaluación

## ■ Modelos existentes : Ejemplo de clasificación logística en *Modelos.ipynb*

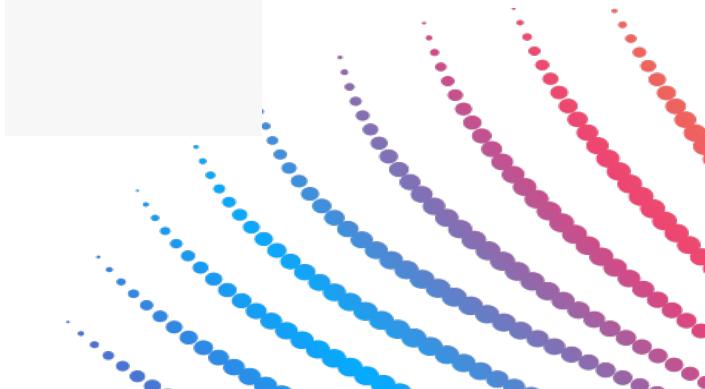
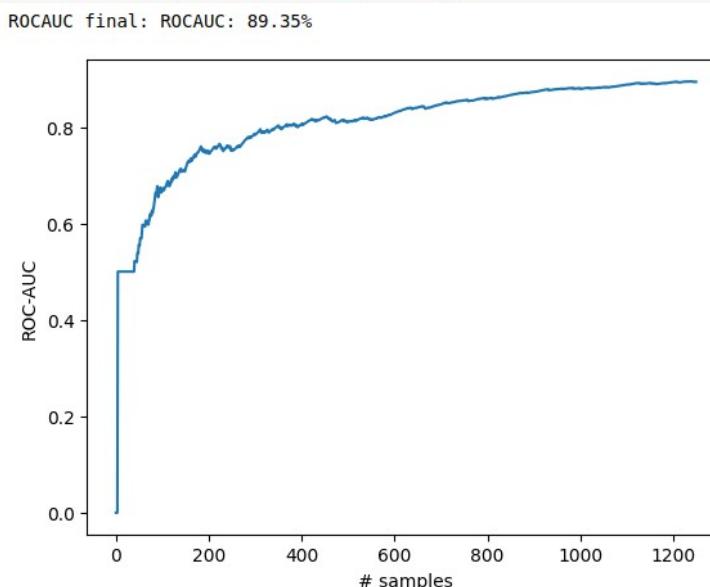
### Ejemplo de métrica: ROC-AUC (Receiver Operating Characteristic/Area Under Curve)

ROC se usa para calcular la calidad predictiva entre la sensibilidad del modelo y la especificidad. En una gráfica, en el eje Y muestra la tasa de verdaderos positivos (bien clasificados) y en el eje X la tasa de falsos positivos (negativos mal clasificados como positivos).

```
In [4]: from river import metrics # Para cargar métricas
import matplotlib.pyplot as plt # para mostrar el history

model= LogisticRegression()
metric = metrics.ROCAUC() # Creación de la métrica
history= [] # Lista para guardar la historia de evolución de la métrica
for x,y in dataset: # Hacemos una pasada sobre todo el dataset, aprendiendo y luego actualizando la métrica
    out= model.predict_proba_one(x)
    model.learn_one(x,y)
    metric.update(y, out) # Actualizamos la métrica
    history.append(metric.get()) # Obtenemos el valor de la métrica en cada iteración

print('ROCAUC final: {}'.format(metric))
plt.plot(history)
plt.ylabel('ROC-AUC')
plt.xlabel('# samples')
display(plt.show())
```



# Evaluación

## ■ Modelos existentes : Ejemplo de clasificación logística en *Modelos.ipynb*

### Evaluación progresiva

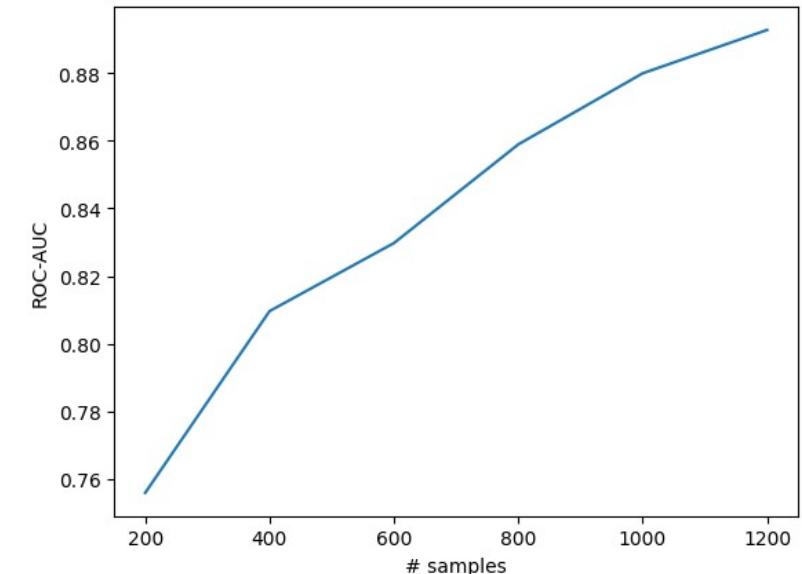
Se puede realizar una evaluación progresiva parando la ejecución cada número de muestras indicado

```
In [6]: steps = evaluate.iter_progressive_val_score(model=LogisticRegression(),
                                                 dataset=dataset,
                                                 metric=metrics.ROCAUC(),
                                                 step=200)

history= []
for step in steps:
    print('Tras {} muestras el ROCAUC vale {}'.format(step['Step'], step['ROCAUC']))
    history.append([step['Step'], step['ROCAUC'].get()])

import numpy as np
history= np.array(history)
plt.plot(history[:, 0], history[:, 1])
plt.ylabel('ROC-AUC')
plt.xlabel('# samples')
display(plt.show())
```

```
Tras 200 muestras el ROCAUC vale ROCAUC: 75.59%
Tras 400 muestras el ROCAUC vale ROCAUC: 80.96%
Tras 600 muestras el ROCAUC vale ROCAUC: 82.98%
Tras 800 muestras el ROCAUC vale ROCAUC: 85.90%
Tras 1000 muestras el ROCAUC vale ROCAUC: 88.00%
Tras 1200 muestras el ROCAUC vale ROCAUC: 89.28%
```



# Composición de modelos

■ Se puede también combinar varios modelos en pipeline de ejecución

## Creación de pipelines

Ejemplo: Creación de pipeline que agrupa un StandardScaler seguido de un modelo de regresión logística

```
In [7]: from river import compose # Para componer modelos
from river import preprocessing

model = compose.Pipeline(
    preprocessing.StandardScaler(),
    LogisticRegression()
)

model
```

Out[7]:



```
In [8]: evaluate.progressive_val_score(dataset, model, metric, print_every=200)
```

```
[200] ROCAUC: 88.74%
[400] ROCAUC: 89.10%
[600] ROCAUC: 89.75%
[800] ROCAUC: 90.51%
[1,000] ROCAUC: 91.26%
[1,200] ROCAUC: 91.81%
```

Out[8]: ROCAUC: 91.87%



# Ejemplo : Clasificación

## Comparación de modelos para clasificación (*Ejemplo1.ipynb*)

### Ejemplo 1

Clasificación binaria. Se dispone de un stream de 100.000 de muestras, que se obtiene desde un Random Tree generator con los siguientes parámetros:

- Número de clases: 2
- Número de atributos numéricos: 5
- Número de atributos categóricos: 5
- Número de categorías por atributo (categórico): 5
- Máxima profundidad del árbol: 5
- Primer nivel del árbol donde puede haber hojas: 3
- Fracción de hojas por nivel: 0.15 (fracción de hojas/nivel desde primer nivel del árbol con hojas en adelante)

Se desea comparar dos clasificadores:

- Naive-Bayes
- Hoeffding Tree (parámetros por defecto)

La métrica de comparación será la tasa de aciertos (accuracy).

Se deberá realizar 30 ejecuciones con semilla aleatoria distinta en todo lo que sea susceptible de tenerla, y comparar los resultados obtenidos.



# Ejemplo : Clasificación

## Comparación de modelos para clasificación (Ejemplo1.ipynb)

### RandomTree

Random Tree generator.

This generator is based on [1](#). The generator creates a random tree by splitting features at random and setting labels at its leaves.

The tree structure is composed of node objects, which can be either inner nodes or leaf nodes. The choice comes as a function of the parameters passed to its initializer.

Since the concepts are generated and classified according to a tree structure, in theory, it should favor decision tree learners.

- **seed\_tree** ('Optional[int / np.random.RandomState]') – defaults to `None`  
Seed for random generation of tree.
- **seed\_sample** ('Optional[int / np.random.RandomState]') – defaults to `None`  
Seed for random generation of instances.
- **n\_classes** ('int') – defaults to `2`  
The number of classes to generate.
- **n\_num\_features** ('int') – defaults to `5`  
The number of numerical features to generate.
- **n\_cat\_features** ('int') – defaults to `5`  
The number of categorical features to generate.
- **n\_categories\_per\_feature** ('int') – defaults to `5`  
The number of values to generate per categorical feature.
- **max\_tree\_depth** ('int') – defaults to `5`  
The maximum depth of the tree concept.
- **first\_leaf\_level** ('int') – defaults to `3`  
The first level of the tree above `max_tree_depth` that can have leaves.
- **fraction\_leaves\_per\_level** ('float') – defaults to `0.15`  
The fraction of leaves per level from `first_leaf_level` onwards.

# Ejemplo : Clasificación

## Comparación de modelos para clasificación (*Ejemplo1.ipynb*)

### Lectura del dataset

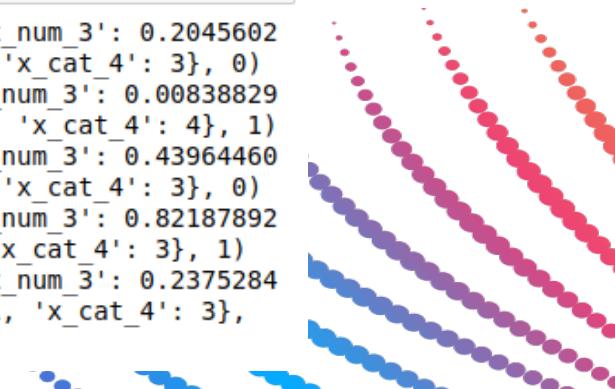
```
: from river import datasets

initial_seed= 12345

dataset= datasets.synth.RandomTree(seed_tree= initial_seed,
                                    seed_sample= initial_seed,
                                    n_classes= 2,
                                    n_num_features= 5,
                                    n_cat_features= 5,
                                    n_categories_per_feature= 5,
                                    max_tree_depth= 5,
                                    first_leaf_level= 3,
                                    fraction_leaves_per_level= 0.15)

first_data= dataset.take(5)
for sample in first_data:
    print(sample)
```

```
({'x_num_0': 0.9296160928171479, 'x_num_1': 0.3163755545817859, 'x_num_2': 0.18391881167709445, 'x_num_3': 0.2045602785530397, 'x_num_4': 0.5677250290816866, 'x_cat_0': 1, 'x_cat_1': 1, 'x_cat_2': 3, 'x_cat_3': 1, 'x_cat_4': 3}, 0)
({'x_num_0': 0.6535698708517353, 'x_num_1': 0.7477148092712739, 'x_num_2': 0.9613067360728214, 'x_num_3': 0.00838829794155349, 'x_num_4': 0.10644437669771933, 'x_cat_0': 3, 'x_cat_1': 0, 'x_cat_2': 1, 'x_cat_3': 3, 'x_cat_4': 4}, 1)
({'x_num_0': 0.7174536208124137, 'x_num_1': 0.4675990072192965, 'x_num_2': 0.3255846775393778, 'x_num_3': 0.43964460588480947, 'x_num_4': 0.7296890827468514, 'x_cat_0': 3, 'x_cat_1': 1, 'x_cat_2': 3, 'x_cat_3': 2, 'x_cat_4': 3}, 0)
({'x_num_0': 0.36370684226490013, 'x_num_1': 0.7949705705074238, 'x_num_2': 0.698481149254162, 'x_num_3': 0.8218789251497105, 'x_num_4': 0.8994658924024351, 'x_cat_0': 0, 'x_cat_1': 3, 'x_cat_2': 0, 'x_cat_3': 0, 'x_cat_4': 3}, 1)
({'x_num_0': 0.6561195917061681, 'x_num_1': 0.8119895661493106, 'x_num_2': 0.10274398936592044, 'x_num_3': 0.23752845451401727, 'x_num_4': 0.37934521444282976, 'x_cat_0': 2, 'x_cat_1': 1, 'x_cat_2': 2, 'x_cat_3': 1, 'x_cat_4': 3}, 1)
```



# Ejemplo : Clasificación

## Comparación de modelos para clasificación (Ejemplo1.ipynb)

### GaussianNB

Gaussian Naive Bayes.

A Gaussian distribution  $G_{cf}$  is maintained for each class  $c$  and each feature  $f$ . Each Gaussian is updated using the amount associated with each feature; the details can be found in `proba.Gaussian`. The joint log-likelihood is then obtained by summing the log probabilities of each feature associated with each class.

### Creación del modelo de Naive-Bayes

```
In [2]: from river import naive_bayes as nb # Import del modelo Naive-Bayes  
NBClassifier= nb.GaussianNB() # Creación del modelo  
  
first_data= dataset.take(5) # Ejemplo: Cogemos sólo 5 valores  
for x,y in first_data: # Ejemplo para lectura de datos y actualización del modelo  
    NBClassifier= NBClassifier.learn_one(x,y)
```

# HoeffdingTreeClassifier

Hoeffding Tree or Very Fast Decision Tree classifier.

## Comparación de modelos para clasificación (Ejemplo 1.ipynb)

- **grace\_period** (*int*) – defaults to `200`

Number of instances a leaf should observe between split attempts.

- **max\_depth** (*int*) – defaults to `None`

The maximum depth a tree can reach. If `None`, the tree will grow indefinitely.

- **split\_criterion** (*str*) – defaults to `info_gain`

Split criterion to use.

- `'gini'` - Gini
- `'info_gain'` - Information Gain
- `'hellinger'` - Helinger Distance

- **delta** (*float*) – defaults to `1e-07`

Significance level to calculate the Hoeffding bound. The significance level is given by `1 - delta`.

Values closer to zero imply longer split decision delays.

- **tau** (*float*) – defaults to `0.05`

Threshold below which a split will be forced to break ties.

- **leaf\_prediction** (*str*) – defaults to `nba`

Prediction mechanism used at leafs.

- `'mc'` - Majority Class
- `'nb'` - Naive Bayes
- `'nba'` - Naive Bayes Adaptive

- **nb\_threshold** (*int*) – defaults to `0`

Number of instances a leaf should observe before allowing Naive Bayes.

- **nominal\_attributes** (*list*) – defaults to `None`

List of Nominal attributes identifiers. If empty, then assume that all numeric attributes should be treated as continuous.

- **splitter** (*river.tree.splitter.base.Splitter*) – defaults to `None`

The Splitter or Attribute Observer (AO) used to monitor the class statistics of numeric features and perform splits. Splitters are available in the `tree.splitter` module. Different splitters are available for classification and regression tasks. Classification and regression splitters can be distinguished by their property `is_target_class`. This is an advanced option. Special care must be taken when choosing different splitters. By default, `tree.splitter.GaussianSplitter` is used if `splitter` is `None`.

- **binary\_split** (*bool*) – defaults to `False`

If True, only allow binary splits.

- **max\_size** (*float*) – defaults to `100.0`

The max size of the tree, in Megabytes (MB).

- **memory\_estimate\_period** (*int*) – defaults to `1000000`

Interval (number of processed instances) between memory consumption checks.

- **stop\_mem\_management** (*bool*) – defaults to `False`

If True, stop growing as soon as memory limit is hit.

- **remove\_poor\_attrs** (*bool*) – defaults to `False`

If True, disable poor attributes to reduce memory usage.

- **merit\_preprune** (*bool*) – defaults to `True`

If True, enable merit-based tree pre-pruning.

# Ejemplo : Clasificación

## Comparación de modelos para clasificación (Ejemplo1.ipynb)

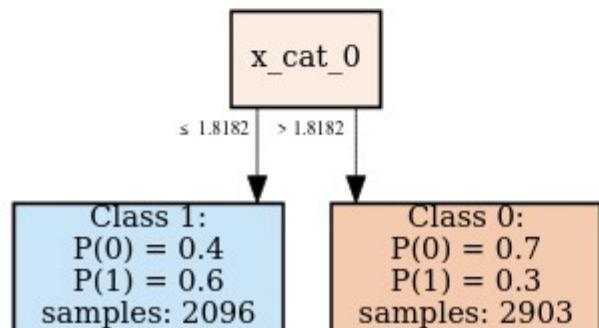
### Creación del modelo de Hoeffding Tree

```
In [3]: from river.tree import HoeffdingTreeClassifier

HTClassifier= HoeffdingTreeClassifier()
first_data= dataset.take(5000) # Ejemplo: Cogemos sólo 5 valores
for x,y in first_data: # Ejemplo para lectura de datos y actualización del modelo
    HTClassifier.learn_one(x,y)

# Visualización del árbol
# Descomentar la siguiente línea si no se tiene instalado el paquete graphviz
#!pip install graphviz
HTClassifier.draw()
```

Out[3]:



# Ejemplo : Clasificación

## Comparación de modelos para clasificación (*Ejemplo1.ipynb*)

### Métrica a usar

```
In [4]: from river.metrics import Accuracy

metric= Accuracy() # Creación de la métrica
first_data= dataset.take(5000) # Ejemplo: Cogemos sólo 5000 valores
for x,y in first_data: # Ejemplo para lectura de datos y actualización del modelo
    HTClassifier.learn_one(x,y)
    predicted= HTClassifier.predict_one(x)
    metric.update(y, predicted) # y= valor de la clase real; predicted= Valor de la clase predicho
print('El valor accuracy={} %'.format(metric.get()*100))
```

El valor accuracy=73.0 %

# Ejemplo : Clasificación

## Comparación de modelos para clasificación (Ejemplo1.ipynb)

- Ahora que ya lo tenemos todo, podemos crear una función que nos ejecute un experimento.

### Definición de función para ejecución de experimentos

```
import numpy as np
import time
import matplotlib.pyplot as plt

# Tiene como entrada un modelo a probar, un flujo de datos sobre el que ejecutar el aprendizaje y la métrica a medir
# Devuelve la historia de la métrica
def Experimento(modelo, flujo, steps, metrica, history_step= 100):
    history= []

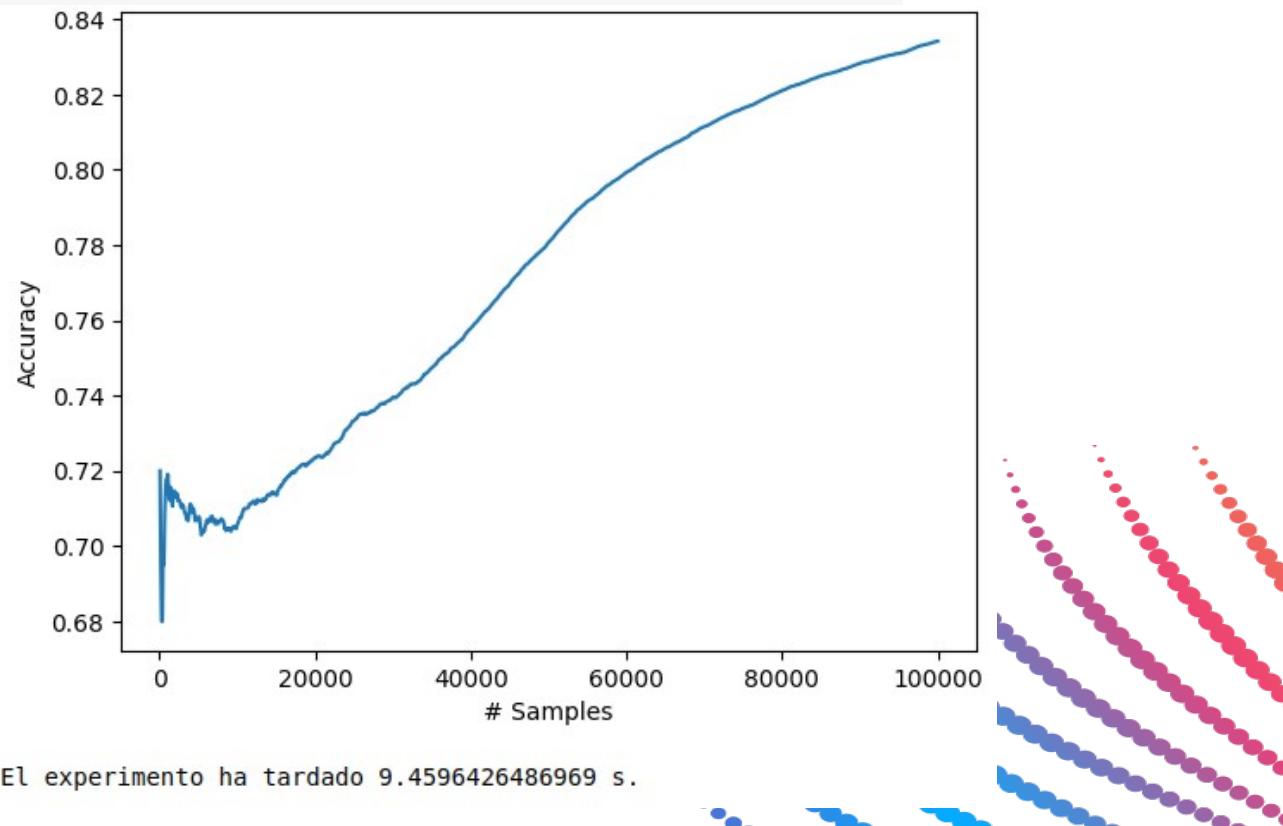
    t0= time.time()
    data= flujo.take(steps) # Obtenemos steps muestras
    for step, sample in enumerate(data):
        x,y= sample
        modelo.learn_one(x,y)
        pred= modelo.predict_one(x)
        metrica.update(y, pred)

        if (step+1)%history_step == 0 or step==steps-1:
            history.append([step, metrica.get()])
    tf= time.time()
    return history, tf-t0
```

# Ejemplo : Clasificación

## Comparación de modelos para clasificación (*Ejemplo1.ipynb*)

```
# Ejemplo de uso
history, tiempo= Experimento(HoeffdingTreeClassifier(), dataset, 100000, Accuracy())
history= np.array(history)
plt.plot(history[:, 0], history[:, 1])
plt.xlabel('# Samples')
plt.ylabel('Accuracy')
plt.show()
print('El experimento ha tardado {} s.'.format(tiempo))
```



# Ejemplo : Clasificación

## Comparación de modelos para clasificación (Ejemplo1.ipynb)

Ahora la experimentación completa con Hoeffding trees

```
In [7]: NumExper= 30
NumMuestras= 100000
semillaInicial= 12345

HTHistories= [] # Para guardar todas las historias del modelo
HTtiempos= [] # Para guardar tiempos de ejecución del modelo
for experimento in range(NumExper):

    # Crear flujo
    dataset= datasets.synth.RandomTree(seed_tree= initial_seed + experimento,
                                         seed_sample= initial_seed + experimento)
    # Crear modelo
    modelo= HoeffdingTreeClassifier()

    # Crear métrica
    metrica= Accuracy()

    # Experimento
    print('Ejecutando experimento {} de {}'.format(experimento+1, NumExper))
    history, tiempo= Experimento(modelo, dataset, NumMuestras, metrica)
    HTHistories.append( history )
    HTtiempos.append( tiempo )

    print('Fin del experimento')
print('Fin de la experimentación')
```



# Ejemplo : Clasificación

## Comparación de modelos para clasificación (Ejemplo1.ipynb)

### Ahora experimentación con Naive-Bayes

```
In [24]: NumExper= 30
NumMuestras= 100000
semillaInicial= 12345

NBHistories= [] # Para guardar todas las historias del modelo
NBtiempos= [] # Para guardar tiempos de ejecución del modelo
for experimento in range(NumExper):

    # Crear flujo
    dataset= datasets.synth.RandomTree(seed_tree= initial_seed + experimento,
                                         seed_sample= initial_seed + experimento)
    # Crear modelo
    modelo= nb.GaussianNB()

    # Crear métrica
    metrica= Accuracy()

    # Experimento
    print('Ejecutando experimento {} de {}'.format(experimento+1, NumExper))
    history, tiempo= Experimento(modelo, dataset, NumMuestras, metrica)
    NBHistories.append( history )
    NBtiempos.append( tiempo )

    print('Fin del experimento')
print('Fin de la experimentación')
```

# Ejemplo : Clasificación

## Comparación de modelos para clasificación (*Ejemplo1.ipynb*)

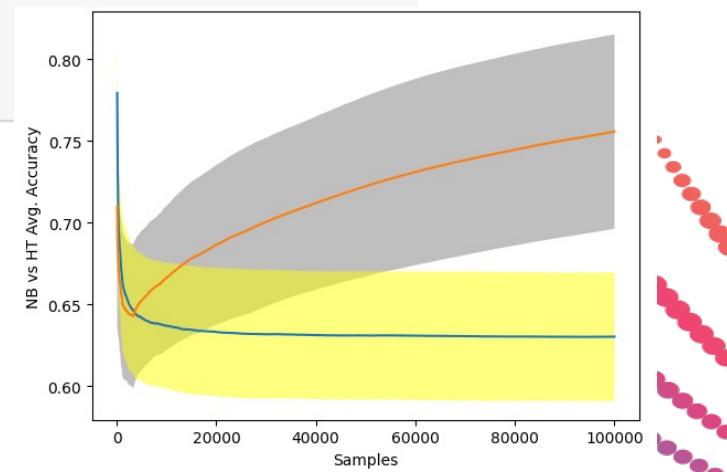
### Mostramos resultados de HT y NB

```
In [30]: print('Accuracy promedio final de HT: {} +- {}'.format(hist_mean[-1, 1], hist_std[-1, 1]))
print('Accuracy promedio final de NB: {} +- {}'.format(nb_hist_mean[-1, 1], nb_hist_std[-1, 1]))

print('Tiempo promedio de ejecución por experimento de HT: {} +- {} s.'.format(np.mean(HTtiempos), np.std(HTtiempos)))
print('Tiempo promedio de ejecución por experimento de NB: {} +- {} s.'.format(np.mean(NBtiempos), np.std(NBtiempos)))

fig, ax = plt.subplots(1)
ax.plot(nb_hist_mean[:, 0], nb_hist_mean[:, 1])
ax.plot(hist_mean[:, 0], hist_mean[:, 1])
ax.fill_between(hist_mean[:, 0], hist_mean[:, 1]-hist_std[:, 1], hist_mean[:, 1]+hist_std[:, 1],
                facecolor='grey', alpha=0.5)
ax.fill_between(nb_hist_mean[:, 0], nb_hist_mean[:, 1]-nb_hist_std[:, 1], nb_hist_mean[:, 1]+nb_hist_std[:, 1],
                facecolor='yellow', alpha=0.5)
plt.xlabel('Samples')
plt.ylabel('NB Avg. Accuracy')
plt.show()
```

Accuracy promedio final de HT: 0.7556836666666666 +- 0.05939099367655746  
Accuracy promedio final de NB: 0.6302506666666668 +- 0.03928115255720258  
Tiempo promedio de ejecución por experimento de HT: 10.156875109672546 +- 0.2723176904312247 s.  
Tiempo promedio de ejecución por experimento de NB: 9.111747288703919 +- 0.08381582061503456 s.



# Ejemplo : Clasificación

## Comparación de modelos para clasificación (*Ejemplo1.ipynb*)

### Tests de comparación de resultados finales

```
HTacc= ht_histories[:, -1, 1]
NBacc= nb_histories[:, -1, 1]

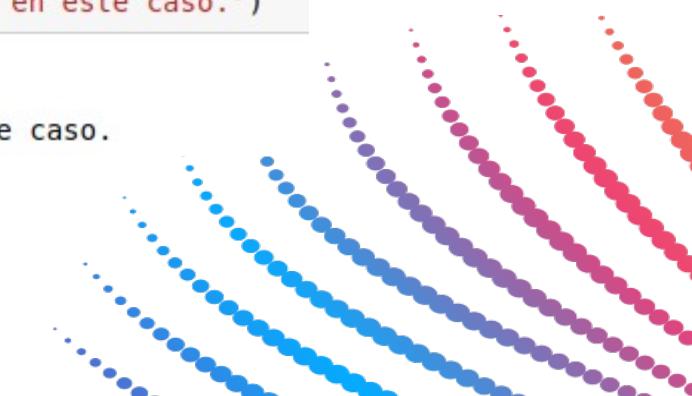
# Test de normalidad
from scipy.stats import normaltest
_, ht_pval= normaltest(HTacc)
_, nb_pval= normaltest(NBacc)
print('p-value de test de normalidad para HT: {} y NB: {}'.format(ht_pval, nb_pval))

# Asumimos que no hay normalidad en al menos una de las distribuciones
from scipy.stats import wilcoxon
_, comparison_pval= wilcoxon(HTacc, NBacc)
print('p-value de comparación de HT y NB: {}'.format(comparison_pval))
print('Hay diferencias significativas con confianza del 95% (pvalue<0.05). Es mejor HT en este caso.')
```

p-value de test de normalidad para HT: 0.867572214595823 y NB: 0.02537374756611938

p-value de comparación de HT y NB: 1.7343976283205784e-06

Hay diferencias significativas con confianza del 95% (pvalue<0.05). Es mejor HT en este caso.



# Evaluación por Heldout

## ■ Evaluación de modelos : Heldout

- La evaluación **Heldout** es la más similar que se puede encontrar en Stream Mining con respecto a la evaluación train/test tradicional.
- Consiste en reservar un conjunto de muestras (**train**) para entrenar los modelos, y un conjunto de muestras siguientes (**test**) para evaluar.
- Esta subdivisión se realiza iterativamente según llegan las muestras desde el flujo de datos.



# Evaluación por Heldout

## ■ Evaluación de modelos con Heldout : *Heldout.ipynb*

- Ejemplo : Clasificación de páginas web de *Phising*.

### Lectura del dataset de Phising

```
In [1]: from river import datasets  
  
flujo= datasets.Phishing() # Lectura del dataset  
  
tr_samples= 150 # Entrenamos con tr_samples muestras  
ts_samples= 100 # Validamos con ts_samples muestras  
  
print(flujo)
```

Phishing websites.

This dataset contains features from web pages that are classified as phishing or not.

Name	Phishing
Task	Binary classification
Samples	1,250
Features	9
Sparse	False
Path	/home/manupc/anaconda3/envs/STMFD/lib/python3.9/site-packages/river/datasets/phishing.csv.gz



# Evaluación por Heldout

## ■ Evaluación de modelos con Heldout : *Heldout.ipynb*

### Creación del modelo y de las métricas a usar para su evaluación

```
In [2]: from river.tree import HoeffdingTreeClassifier  
from river.metrics import Accuracy, Precision, Recall  
  
# Creación de un Hoeffding Tree para clasificación  
model= HoeffdingTreeClassifier()
```

# Evaluación por Heldout

## Evaluación de modelos con Heldout : *Heldout.ipynb*

### Clasificación con heldout

```
In [3]: history= []

action= 0 # valor 0 para train, valor 1 para test
processed_samples= 0 # Muestras procesadas en la acción actual
for t, (x,y) in enumerate(flujo):
    if action == 0: # Sólo training
        model.learn_one(x, y)

    else: # sólo evaluación
        yp= model.predict_one(x)
        accMetric.update(y, yp)
        precMetric.update(y, yp)
        recMetric.update(y, yp)

    # Comprobamos si hay que cambiar entre training y test
    processed_samples+= 1
    if action == 0 and (processed_samples%tr_samples==0):

        # Creación de las métricas a medir
        accMetric= Accuracy()
        precMetric= Precision()
        recMetric= Recall()

        # Cambiamos a acción de test
        action= 1
        processed_samples= 0

    elif action == 1 and (processed_samples%ts_samples==0):

        # Guardamos History
        history.append([t, accMetric.get()*100, precMetric.get()*100, recMetric.get()*100])

        # Cambiamos a acción de training
        action= 0
        processed_samples= 0
```



# Evaluación por Heldout

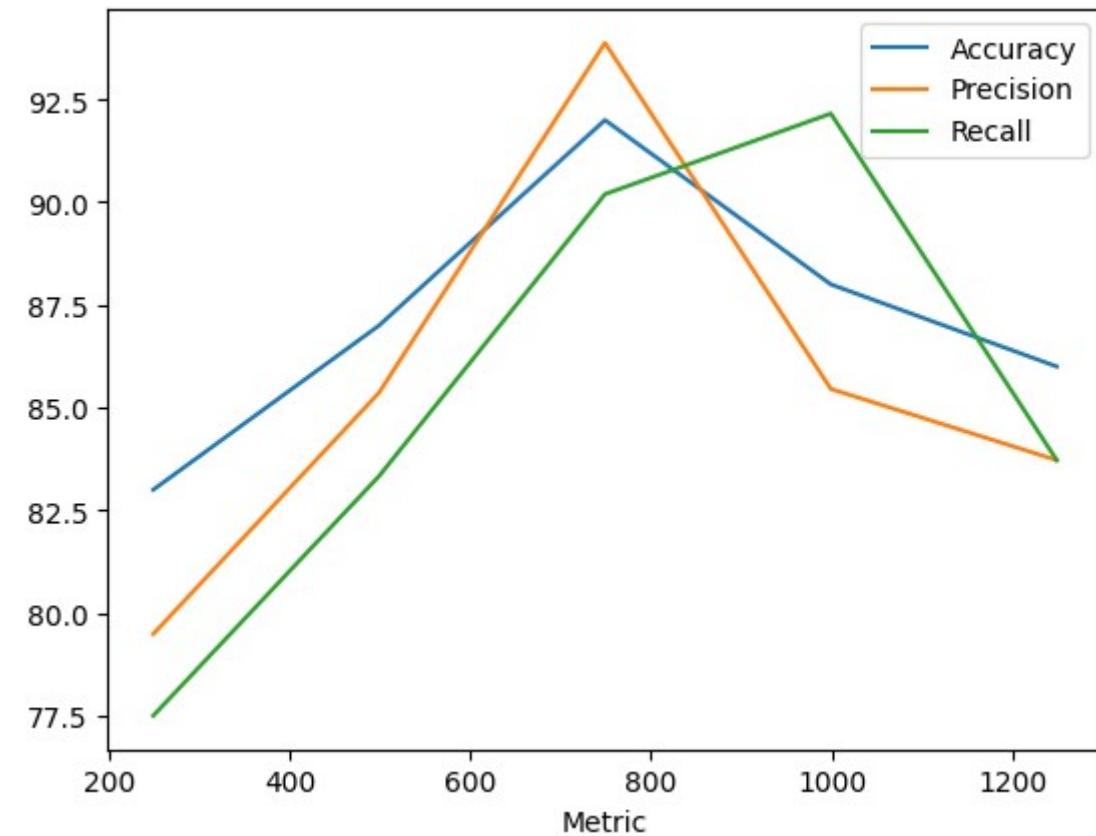
## Evaluación de modelos con Heldout : *Heldout.ipynb*

### Mostramos evolución de resultados

```
In [4]: import numpy as np
import matplotlib.pyplot as plt

history= np.array(history)

# Métrica de accuracy
plt.plot(history[:, 0], history[:, 1])
plt.plot(history[:, 0], history[:, 2])
plt.plot(history[:, 0], history[:, 3])
plt.xlabel('# samples')
plt.xlabel('Metric')
plt.legend(['Accuracy', 'Precision', 'Recall'])
plt.show()
```



# Evaluación por Heldout

## Evaluación de modelos con Heldout : *Heldout.ipynb*

### Comparación con Prequential

```
In [5]: # Modelo
model= HoeffdingTreeClassifier()

# Creación de las métricas a medir
accMetric= Accuracy()
precMetric= Precision()
recMetric= Recall()

# Evolución prequential
history= []
for t, (x,y) in enumerate(flujo):
    yp= model.predict_one(x)
    accMetric.update(y, yp)
    precMetric.update(y, yp)
    recMetric.update(y, yp)
    model.learn_one(x, y)
    if (t+1)%(tr_samples+ts_samples) == 0:
        history.append([t, accMetric.get()*100, precMetric.get()*100, recMetric.get()*100])
```

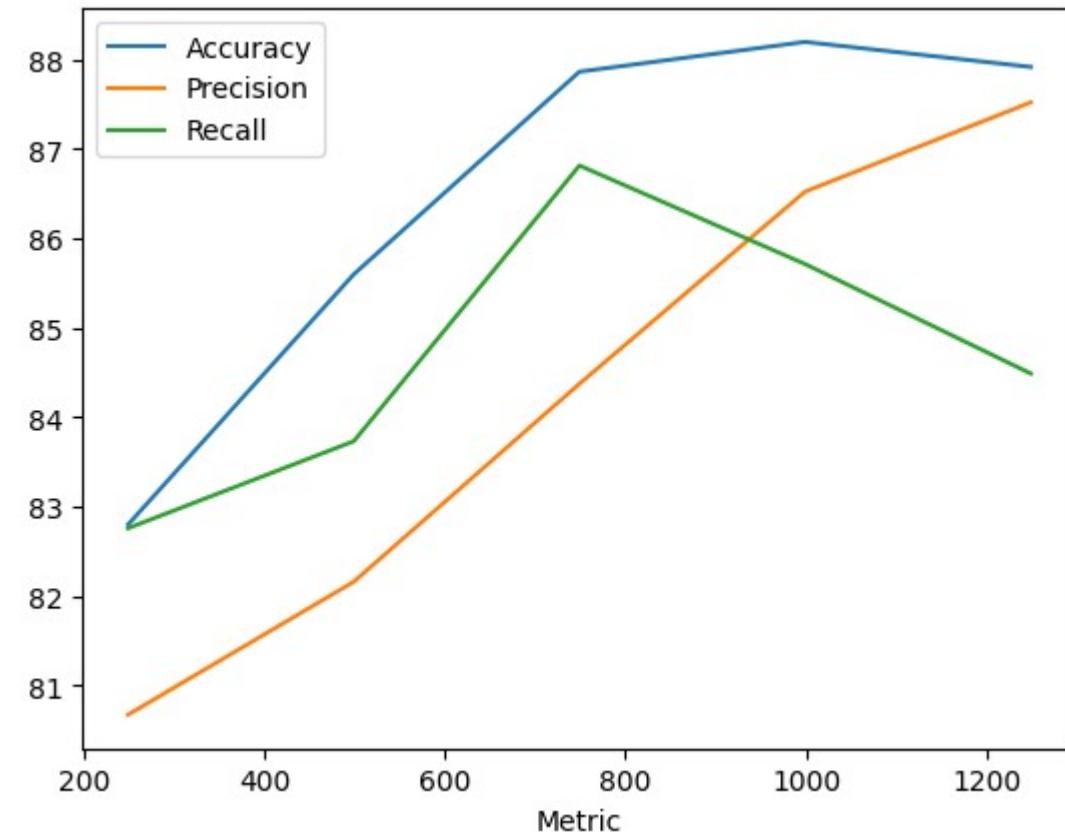
# Evaluación por Heldout

## Evaluación de modelos con Heldout : *Heldout.ipynb*

### Resultados

```
In [6]: history= np.array(history)

# Métrica de accuracy
plt.plot(history[:, 0], history[:, 1])
plt.plot(history[:, 0], history[:, 2])
plt.plot(history[:, 0], history[:, 3])
plt.xlabel('# samples')
plt.xlabel('Metric')
plt.legend(['Accuracy', 'Precision', 'Recall'])
plt.show()
```



# Evaluación por Heldout

## Evaluación de modelos con Heldout : *Heldout.ipynb*

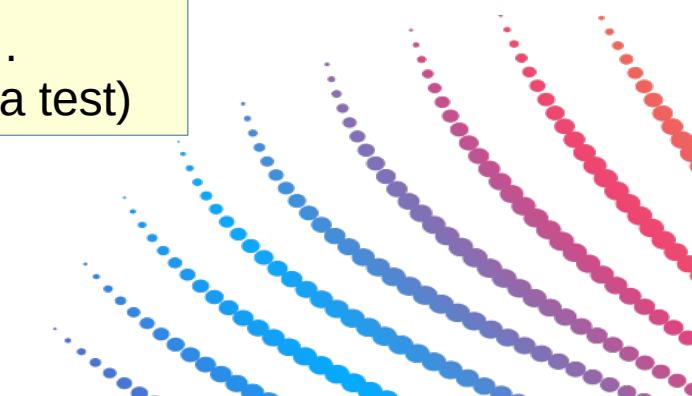
### Comparación al final del dataset

```
In [7]: print('Heldout (final):')
print(heldout_history[-1, :])

print('prequential (final):')
print(prequential_history[-1, :])
```

Heldout (final):  
[1249. 86. 83.72093023 83.72093023]  
prequential (final):  
[1249. 87.92 87.52362949 84.48905109]

Ningún método es mejor que otro en todos los casos.  
Depende del objetivo perseguido, recursos, naturaleza de los datos...  
En datasets con inicio y fin, prequential aprende de más datos (heldout usa test)



# Concept drift



# Concept drift

## ■ Desvío de concepto (Concept drift)

- El término **Concept Drift** se utiliza para denominar aquellas situaciones donde las propiedades estadísticas de un fenómeno en estudio cambian a lo largo del tiempo.
- River pone a disposición del programador diversas utilidades para detección y tratamiento del **concept drift**.

drift	synth
	Agrawal
	AnomalySine
	ConceptDriftStream
ADWIN	Friedman
DDM	FriedmanDrift
EDDM	Hyperplane
HDDM_A	LED
HDDM_W	LEDDrift
KSWIN	Logical
PageHinkley	Mixed
	Mv
	Planes2D
	RandomRBF
	RandomRBFDrift



# Concept drift

## ■ Desvío de concepto (*Concept drift*)

- La **detección** de *concept drift* es importante para conocer cuándo los datos que están siendo evaluados/aprendidos difieren sustancialmente de los previamente presentados.
- Tipos de generadores para cambios de concepto: Cambio abrupto, cambio paulatino, sin cambio.
- Algoritmos: DDM, ADWIN, CUSUM, Page-Hinkley, etc.
- La clave está en encontrar los parámetros óptimos del algoritmo de detección para evitar falsas alarmas de cambio de concepto.



# Concept drift

## Ejemplo de Concept Drift : *ConceptDrift.ipynb*

### Ejemplo de detección de cambio de concepto

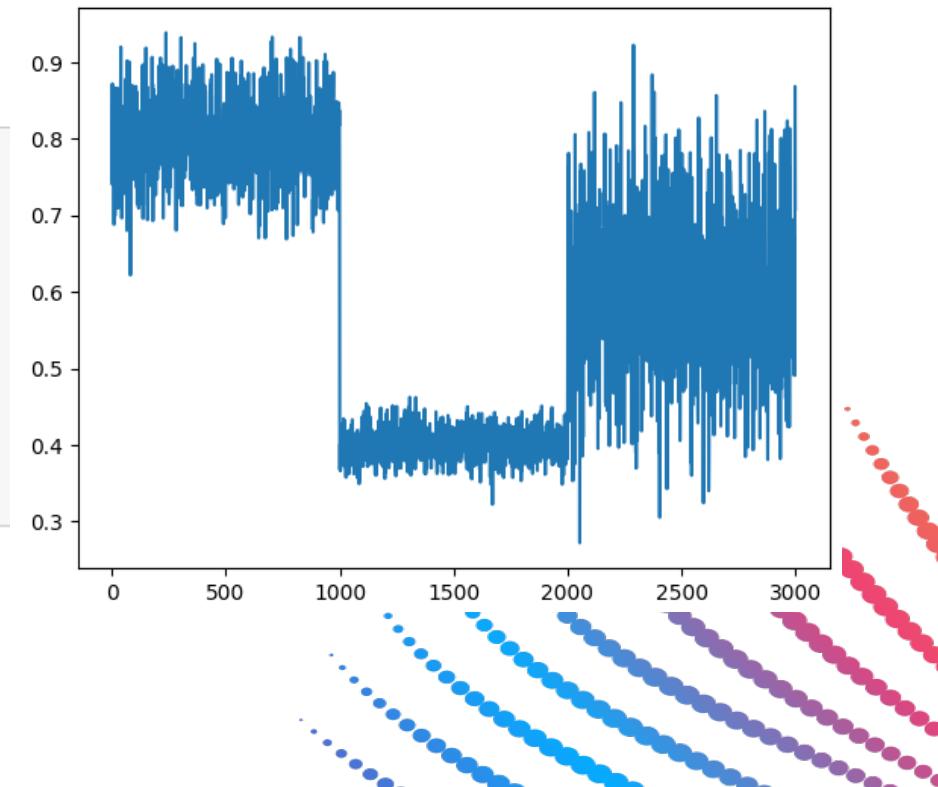
Vamos a ver un ejemplo simple donde hay un flujo de datos  $\{x_1, x_2, \dots, x_{t-1}, x_t, x_{t+1}, \dots\}$  donde medimos datos procedentes de una distribución normal  $\mathcal{N}(\mu, \sigma)$ . Simularemos dos cambios de concepto concatenando 1000 muestras procedentes de las siguientes distribuciones:

- A:  $\mathcal{N}(\mu = 0.8, \sigma = 0.05)$
- B:  $\mathcal{N}(\mu = 0.4, \sigma = 0.02)$
- C:  $\mathcal{N}(\mu = 0.6, \sigma = 0.1)$

```
In [2]: import numpy as np
import matplotlib.pyplot as plt

np.random.seed(1234) # Inicialización de semilla por reproducibilidad de experimentos

A= np.random.normal(0.8, 0.05, size= 1000)
B= np.random.normal(0.4, 0.02, size= 1000)
C= np.random.normal(0.6, 0.1, size= 1000)
stream= np.concatenate([A, B, C])
plt.plot(stream)
```



# Concept drift

## Ejemplo de Concept Drift : *ConceptDrift.ipynb*

### Ejemplo de uso del detector ADWIN sobre el dataset

In [4]:

```
from river import drift

detectorCD= drift.ADWIN() # Algoritmo ADWIN con parámetros por defecto
detected_drifts= [] # muestras donde se ha detectado cambio de concepto
true_drifts= [1000, 2000] # Muestras donde hay un cambio de concepto real (0...999, 1000...1999)

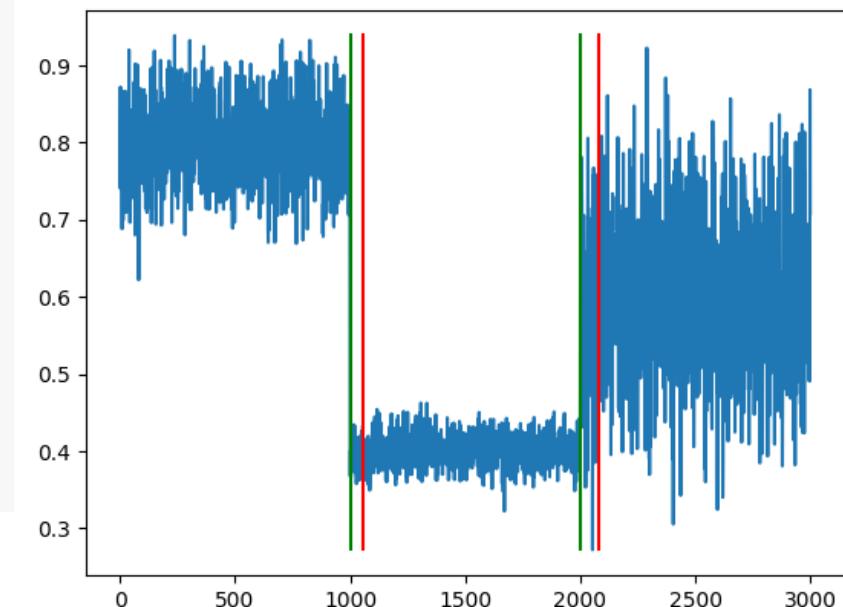
for sample_number, value in enumerate(stream):
    detectorCD.update(value) # Actualizamos el detector con el nuevo dato
    if detectorCD.drift_detected: # Si se ha detectado un cambio de concepto, guardamos
        detected_drifts.append(sample_number)
        print('Detectado drift en sample {}'.format(sample_number))

# Mostramos datos gráficamente
plt.plot(stream) # Stream
minimo, maximo= np.min(stream), np.max(stream)

# True drifts
for tdrift in true_drifts:
    plt.plot([tdrift, tdrift], [minimo, maximo], '--', color='green')

# Detected drifts
for ddrift in detected_drifts:
    plt.plot([ddrift, ddrift], [minimo, maximo], '--', color='red')
```

Detectado drift en sample 1055  
Detectado drift en sample 2079



# Concept drift y clasificación

## ■ Entradas a un detector de Concept drift : *ConceptDrift.ipynb*

- La entrada a un detector de cambio de concepto puede ser cualquier cosa (depende de para qué lo vayamos a usar).
- En combinación con un clasificador, las entradas a un detector de cambio de concepto suelen ser los aciertos (valor 0) o fallos (valor 1) cometidos por el clasificador.
- Cuando la tasa de aciertos (fallos) cambia su distribución de probabilidad, se detecta un cambio de concepto (o, al menos, debería...)
- Veremos un ejemplo con el ***Drift Detection Method*** (DDM)



# Concept drift y clasificación

## ■ Entradas a un detector de Concept drift : *ConceptDrift.ipynb*

### DDM

Drift Detection Method.

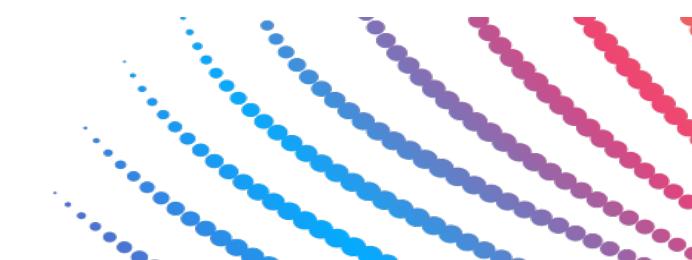
DDM (Drift Detection Method) is a concept change detection method based on the PAC learning model premise, that the learner's error rate will decrease as the number of analysed samples increase, as long as the data distribution is stationary.

If the algorithm detects an increase in the error rate, that surpasses a calculated threshold, either change is detected or the algorithm will warn the user that change may occur in the near future, which is called the warning zone.

### Parameters

---

- **warm\_start** (*int*) – defaults to `30`  
The minimum required number of analyzed samples so change can be detected. Warm start parameter for the drift detector.
- **warning\_threshold** (*float*) – defaults to `2.0`  
Threshold to decide if the detector is in a warning zone. The default value gives 95\% of confidence level to the warning assessment.
- **drift\_threshold** (*float*) – defaults to `3.0`  
Threshold to decide if a drift was detected. The default value gives a 99\% of confidence level to the drift assessment.



# Concept drift y clasificación

## ■ Entradas a un detector de Concept drift : *ConceptDrift.ipynb*

### Simulación de aciertos (0) y fallos (1) de un clasificador

```
In [3]: import numpy as np

num_samples= 1000 # Número de muestras a tomar antes y después del Concept Drift

# Suponemos que el clasificador tiene un 80% de aciertos (20% de fallos)
# Se asume valor 0 para fallo y valor 1 para acierto
clasificador= np.random.choice(range(2), p= [0.8, 0.2], size= num_samples)

# Performance del clasificador tras concept drift
# Suponemos que el clasificador tiene un 60% de aciertos (40% de fallos)
clasificador_cd= np.random.choice(range(2), p= [0.2, 0.8], size= num_samples)

# Creamos varios cambios de concepto seguidos
todas_salidas= np.concatenate((clasificador, clasificador_cd, clasificador, clasificador_cd))
```

# Concept drift y clasificación

## ■ Entradas a un detector de Concept drift : *ConceptDrift.ipynb*

### Creación del DDM

```
In [4]: from river.drift import DDM

detector= DDM(warm_start= 30) # Tras 20 muestras se lanza alarma

# Entrenamos al detector con los datos iniciales antes de que se produzca ningún CD
for i, sample in enumerate(clasificador):
    detector.update(sample)

real_cd= [num_samples, 2*num_samples, 3*num_samples]
detected_cd= []
for i, sample in enumerate(todas_salidas):
    detector.update(sample)
    if detector.drift_detected:
        #detector= DDM(warm_start= 30) # Tras 20 muestras se lanza alarma
        detected_cd.append(i)
        print('Detectado cambio de concepto en muestra {}'.format(i))
```

Detectado cambio de concepto en muestra 1054  
Detectado cambio de concepto en muestra 3142

OJO: DDM se basa en que el clasificador mejorará con el tiempo.  
Por ese motivo no detecta en 2000

# Concept drift y clasificación

## ■ Efecto de clasificadores en datos con desvío de concepto

- Si los datos tienen concept drift y despreciamos su existencia, esto puede tener un gran impacto en la precisión del clasificador.
- Ejemplo :
  - Aprendemos con modelo HoeffdingTree
  - Usaremos como flujo de datos el flujo sintético SEA. Simularemos un concept drift intercambiando variantes del flujo cada 10.000 instancias.
  - Evaluaremos el modelo con la métrica Accuracy. Usaremos un total de 100.000 instancias en total.

# Concept drift y clasificación

## Efecto de clasificadores en datos con desvío de concepto

### SEA

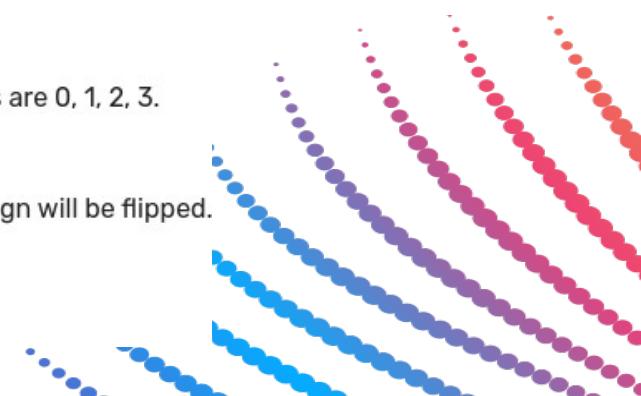
SEA synthetic dataset.

Implementation of the data stream with abrupt drift described in [1](#). Each observation is composed of 3 features. Only the first two features are relevant. The target is binary, and is positive if the sum of the features exceeds a certain threshold. There are 4 thresholds to choose from. Concept drift can be introduced by switching the threshold anytime during the stream.

#### Parameters

- **Variant 0:** True if  $att1 + att2 > 8$
- **Variant 1:** True if  $att1 + att2 > 9$
- **Variant 2:** True if  $att1 + att2 > 7$
- **Variant 3:** True if  $att1 + att2 > 9.5$

- **variant** – defaults to 0  
Determines the classification function to use. Possible choices are 0, 1, 2, 3.
- **noise** – defaults to 0.0  
Determines the amount of observations for which the target sign will be flipped.
- **seed (int)** – defaults to None  
Random seed number used for reproducibility.



# Concept drift y clasificación

## Efecto de clasificadores en datos con desvío de concepto (ClasificadorConceptDrift.ipynb)

### Creación de los flujos de datos

```
In [1]: from river import datasets

SemillaInicial= 12345 # Para reproducibilidad de resultados
SamplesConceptDrift= 5000 # Cada cuántas muestras hay un desvío de concepto (punto central del CD)

flujo= [] # Array de flujos
flujo0= datasets.synth.SEA(variant= 0, seed= SemillaInicial)
flujo1= datasets.synth.SEA(variant= 1, seed= SemillaInicial)

# Creamos un concept drift en datos
dataset= datasets.synth.ConceptDriftStream(stream= flujo0, drift_stream= flujo1, position=SamplesConceptDrift)

# Creamos dos más concatenando el dataset
dataset= datasets.synth.ConceptDriftStream(stream= dataset, drift_stream= dataset, position=2*SamplesConceptDrift)
```

# Concept drift y clasificación

## Efecto de clasificadores en datos con desvío de concepto (ClasificadorConceptDrift.ipynb)

### Creación del modelo

```
In [2]: from river.tree import HoeffdingTreeClassifier
from river.metrics import Accuracy

ht= HoeffdingTreeClassifier()
metric= Accuracy()
```

In [3]:

```
MaxSamples= 20000 # Máximo número de muestras a procesar

# Pre-entrenamos el modelo para aprender el flujo de variante 0
print('Preentrenando modelo...')
samples= flujo0.take(MaxSamples) # Obtenemos muestras
for x,y in samples:
    yp= ht.predict_one(x)
    metric.update(y, yp)
    ht.learn_one(x,y)
print('Tras preentrenar, el modelo tiene un accuracy: {}'.format(metric))

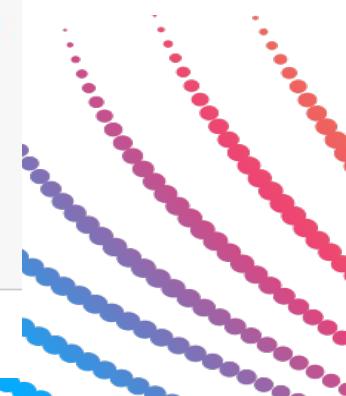
num_samples= 0 # Número de muestras procesadas
currentDataset= 0 # Dataset actual desde donde se cogen los datos
accHistory= [] # Histórico de tasa de aciertos

samples= dataset.take(MaxSamples) # Cogemos MaxSamples muestras del dataset

for x,y in samples:
    num_samples+= 1
    yp= ht.predict_one(x) # Evaluación Prequential (Test-Then-Train)
    metric.update(y, yp)
    accHistory.append(metric.get()*100) # *100 para que sea porcentaje
    ht.learn_one(x, y)
```

Preentrenando modelo...

Tras preentrenar, el modelo tiene un accuracy: Accuracy: 96.50%

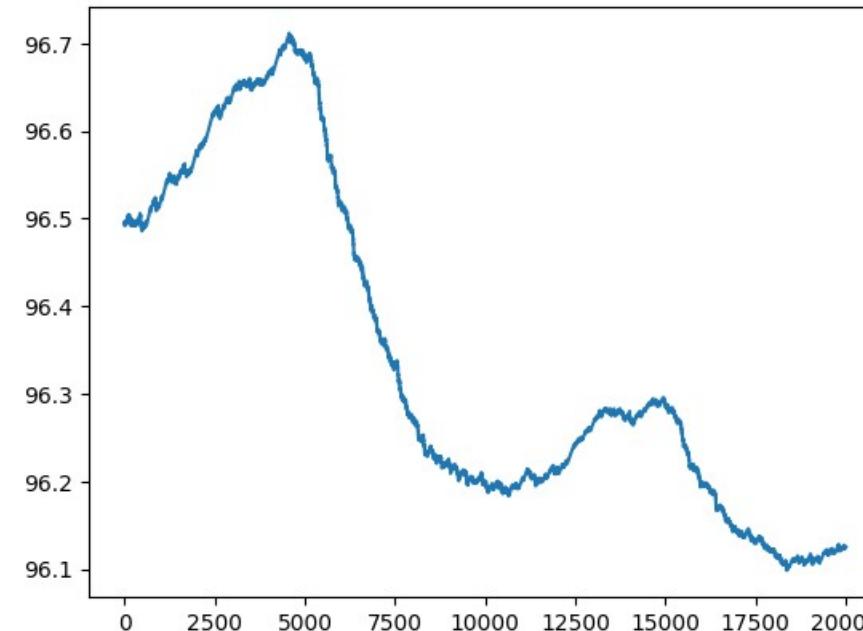


# Concept drift y clasificación

## Efecto de clasificadores en datos con desvío de concepto (ClasificadorConceptDrift.ipynb)

Mostramos resultado de aprendizaje gráficamente

```
In [4]: import matplotlib.pyplot as plt
plt.plot(accHistory)
plt.show()
```



# Concept drift y clasificación

## ■ Combinando detectores de Concept Drift con reinicialización *(ReiniciarConceptDrift.ipynb)*

- Idea general :
  - Si el clasificador ha dejado de funcionar, reiniciarlo desde cero.
  - Útil cuando el modelo no tiene capacidades para adaptarse a un cambio de concepto.
  - Usaremos el Drift Detection Method con ventana de 20 muestras.



# Concept drift y clasificación

## ■ Combinando detectores de Concept Drift con reinicialización (ReiniciarConceptDrift.ipynb)

- Igual que antes :

### Creación de los flujos de datos

```
In [1]: from river import datasets

SemillaInicial= 12345 # Para reproducibilidad de resultados
SamplesConceptDrift= 5000 # Cada cuántas muestras hay un desvío de concepto (punto central del CD)

flujo= [] # Array de flujos
flujo0= datasets.synth.SEA(variant= 0, seed= SemillaInicial)
flujo1= datasets.synth.SEA(variant= 1, seed= SemillaInicial)

# Creamos un concept drift en datos
dataset= datasets.synth.ConceptDriftStream(stream= flujo0, drift_stream= flujo1, position=SamplesConceptDrift)

# Creamos dos más concatenando el dataset
dataset= datasets.synth.ConceptDriftStream(stream= dataset, drift_stream= dataset, position=2*SamplesConceptDrift)
```

# Concept drift y clasificación

## ■ Combinando detectores de Concept Drift con reinicialización (ReiniciarConceptDrift.ipynb)

### Creación del modelo

```
In [2]: from river.tree import HoeffdingTreeClassifier
from river.metrics import Accuracy
from river.drift import DDM

ht= HoeffdingTreeClassifier()
metric= Accuracy()
detectorCD= DDM(warm_start= 20) # Ventana de 20 samples para lanzar alarma
```

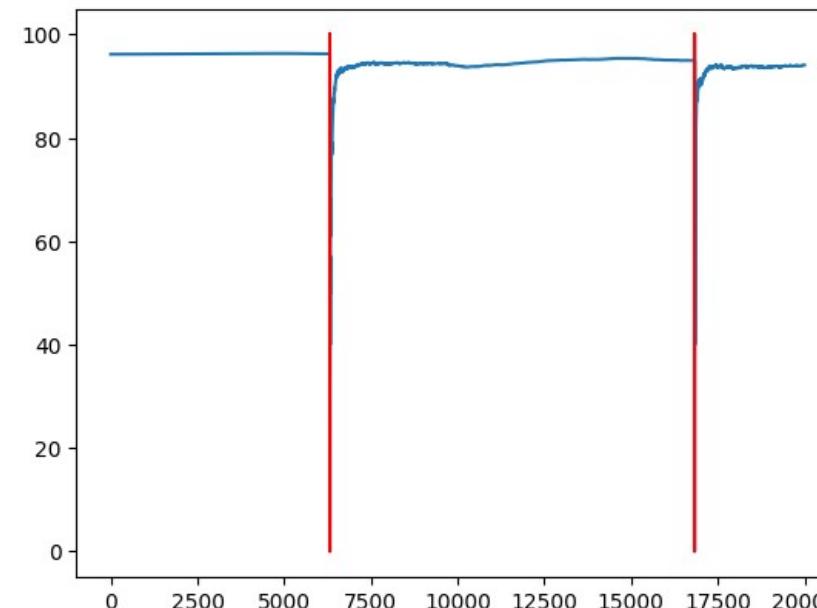
# Concept drift y clasificación

## ■ Combinando detectores de Concept Drift con reinicialización (ReiniciarConceptDrift.ipynb)

Mostramos resultado de aprendizaje gráficamente

```
In [6]: import matplotlib.pyplot as plt
import numpy as np

plt.plot(accHistory)
minimo, maximo= np.min(accHistory), np.max(accHistory)
for x in detected_cd:
    plt.plot([x, x], [minimo, maximo], '--', color='red')
plt.plot()
plt.show()
```



# Concept drift y clasificación

## ■ Combinando Concept Drift con modelos adaptativos ([AdaptiveConceptDrift.ipynb](#))

- Igual que antes :

### Creación de los flujos de datos

```
In [1]: from river import datasets

SemillaInicial= 12345 # Para reproducibilidad de resultados
SamplesConceptDrift= 5000 # Cada cuántas muestras hay un desvío de concepto (punto central del CD)

flujo= [] # Array de flujos
flujo0= datasets.synth.SEA(variant= 0, seed= SemillaInicial)
flujo1= datasets.synth.SEA(variant= 1, seed= SemillaInicial)

# Creamos un concept drift en datos
dataset= datasets.synth.ConceptDriftStream(stream= flujo0, drift_stream= flujo1, position=SamplesConceptDrift)

# Creamos dos más concatenando el dataset
dataset= datasets.synth.ConceptDriftStream(stream= dataset, drift_stream= dataset, position=2*SamplesConceptDrift)
```

# Concept drift y clasificación

## ■ Combinando Concept Drift con modelos adaptativos (*AdaptiveConceptDrift.ipynb*)

- Como ejemplo de modelo adaptativo : Adaptive Hoeffding Tree

### Creación del modelo

```
In [2]: from river.tree import HoeffdingAdaptiveTreeClassifier
from river.metrics import Accuracy

ht = HoeffdingAdaptiveTreeClassifier(grace_period= 100, drift_window_threshold= 100)
metric = Accuracy()
```

# Concept drift y clasificación

## ■ Combinando Concept Drift con modelos adaptativos (*AdaptiveConceptDrift.ipynb*)

### Simulación manual de evolución del aprendizaje

In [3]:

```
MaxSamples= 20000 # Máximo número de muestras a procesar

num_samples= 0 # Número de muestras procesadas
currentDataset= 0 # Dataset actual desde donde se cogen los datos
accHistory= [] # Histórico de tasa de aciertos

samples= dataset.take(MaxSamples) # Cogemos MaxSamples muestras del dataset
for i, (x,y) in enumerate(samples):
    num_samples+= 1
    yp= ht.predict_one(x) # Evaluación Prequential (Test-Then-Train)
    metric.update(y, yp)
    accHistory.append(metric.get()*100) # *100 para que sea porcentaje
    ht.learn_one(x, y)
```

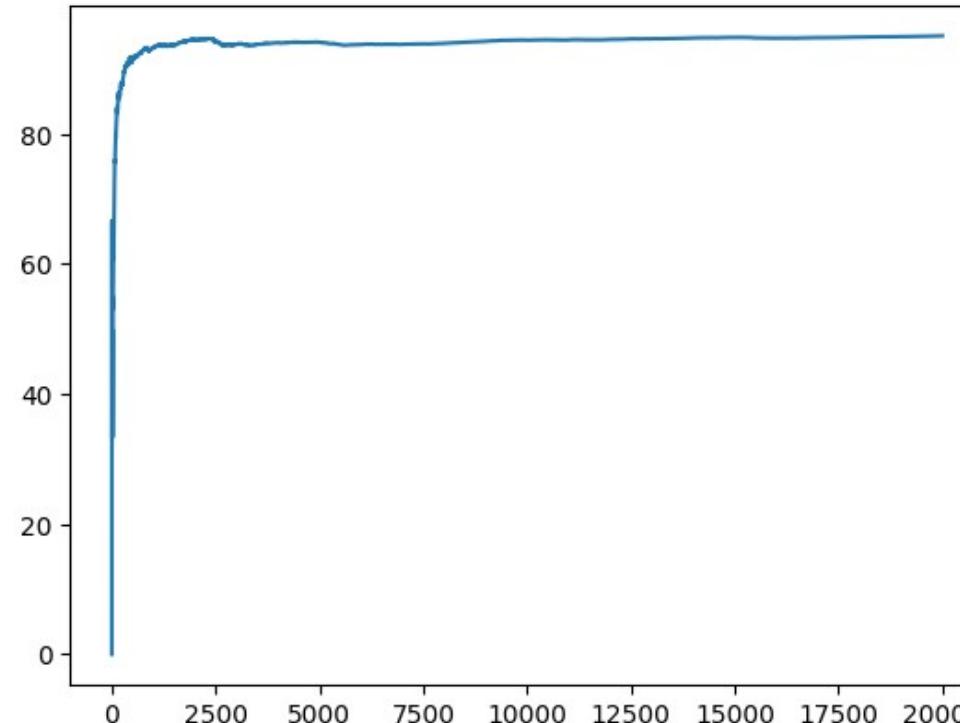


# Concept drift y clasificación

## ■ Combinando Concept Drift con modelos adaptativos (*AdaptiveConceptDrift.ipynb*)

**Mostramos resultado de aprendizaje gráficamente**

```
In [4]: import matplotlib.pyplot as plt  
import numpy as np  
  
plt.plot(accHistory)  
plt.show()
```



# Concept drift y clasificación

## ■ Ejercicio de Concept Drift

- Ejecute el algoritmo ADWIN con la distribución de aciertos (0) / fallos (1) del cuaderno ***ConceptDrift.ipynb***
- Compare los resultados de ADWIN con DDM.
- Indique qué diferencias puede detectar entre ambos detectores.
- Indique cuál es mejor en este conjunto de datos.
- Solución : ***SoluciónEjercicioConceptDrift.ipynb***



# Ejemplo : Regresión en Stream Mining



# Regresión en Stream Mining

## Modelos de Regresión

- River proporciona diversos modelos que pueden ser usados para regresión en Stream Mining :

linear_model	▼	neural_net	▼	tree	▼
ALMAClassifier		MLPRegressor		ExtremelyFastDecisionTreeCla	
BayesianLinearRegression		activations	➤	HoeffdingAdaptiveTreeClassifie	
LinearRegression				HoeffdingAdaptiveTreeRegress	
LogisticRegression				HoeffdingTreeClassifier	
PAClassifier				HoeffdingTreeRegressor	
PARegressor				LabelCombinationHoeffdingTre	
Perceptron				SGTClassifier	
SoftmaxRegression				SGTRegressor	
base	➤			iSOUPTreeRegressor	

# Regresión en Stream Mining

## ■ Modelos de Regresión

- River proporciona métricas adaptadas para el problema de regresión (MAE , MSE, ...)

MAE	R2
MCC	RMSE
MSE	RMSLE
	ROCAUC



# Regresión en Stream Mining

## ■ Modelos de Regresión

- Como ejemplo, usaremos un Hoeffding tree adaptativo para regresión para predecir la tasa de aprobaciones de Donald Trump (**Regresion.ipynb**).

### HoeffdingAdaptiveTreeRegressor

Hoeffding Adaptive Tree regressor (HATR).

This class implements a regression version of the Hoeffding Adaptive Tree Classifier. Hence, it also uses an ADWIN concept-drift detector instance at each decision node to monitor possible changes in the data distribution. If a drift is detected in a node, an alternate tree begins to be induced in the background. When enough information is gathered, HATR swaps the node where the change was detected by its alternate tree.



# Regresión en Stream Mining

## ■ Modelos de Regresión : *Regresion.ipynb*

### Lectura del dataset

```
In [1]: from river import datasets  
  
dataset = datasets.TrumpApproval()  
print(dataset)  
x,y= next(iter(dataset))  
print('Entrada:\n', x)
```

Donald Trump approval ratings.

This dataset was obtained by reshaping the data used by FiveThirtyEight for analyzing Donald Trump's approval ratings. It contains 5 features, which are approval ratings collected by 5 polling agencies. The target is the approval rating from FiveThirtyEight's model. The goal of this task is to see if we can reproduce FiveThirtyEight's model.

```
Name    TrumpApproval  
Task    Regression  
Samples 1,001  
Features 6  
Sparse   False  
Path    /home/manupc/anaconda3/envs/STMFD/lib/python3.9/site-packages/river/datasets/trump_approval.csv.gz  
Entrada:  
{'ordinal_date': 736389, 'gallup': 43.843213, 'ipsos': 46.19925042857143, 'morning_consult': 48.318749, 'rasmussen': 44.104692, 'you_gov': 43.636914000000004}
```

# Regresión en Stream Mining

## Modelos de Regresión : *Regresion.ipynb*

### Creación del modelo de regresión

```
In [2]: from river import compose
from river.tree import HoeffdingAdaptiveTreeRegressor # Red neuronal Perceptrón multi-capa
from river.neural_net.activations import ReLU, Identity # Funciones de activación
from river.optim import SGD # Algoritmo de entrenamiento: Stochastic Gradient Descent
from river.metrics import MSE # Función de pérdida
from river.evaluate import progressive_val_score
from river import compose # Para componer modelos
from river import preprocessing

model = HoeffdingAdaptiveTreeRegressor() # Creación del modelo

model = compose.Pipeline( # Empaquetamos en un pipeline con preprocessamiento
    preprocessing.StandardScaler(),
    model
)
metric= MSE() # Métrica a usar

true_y= [] # salidas esperadas
predicted_y= [] # Salidas predichas por el modelo

for x,y in dataset: # Iteramos sobre el flujo

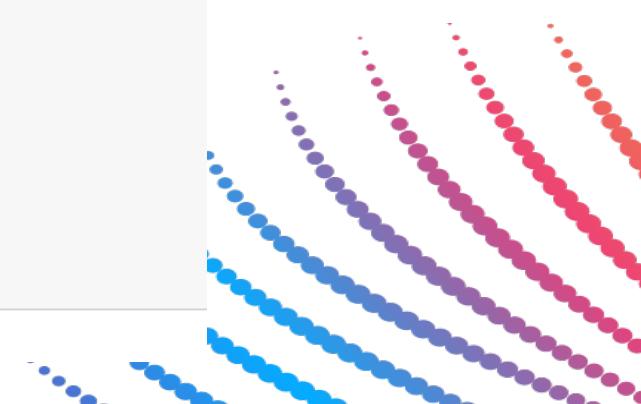
    true_y.append(y) # Añadimos valor real al history

    yp= model.predict_one(x) # Prequential
    metric.update(y, yp)
    model.learn_one(x, y)

    predicted_y.append(yp) # Añadimos valor predicho a History

print('El valor MSE al final es: ', metric)

El valor MSE al final es:  MSE: 4.957847
```

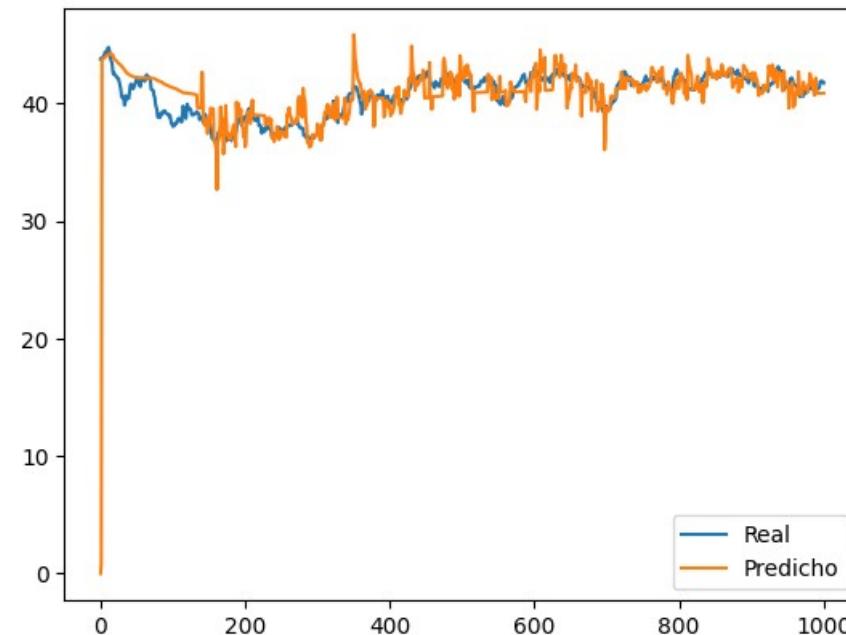


# Regresión en Stream Mining

## ■ Modelos de Regresión : *Regresion.ipynb*

### Mostrar resultados

```
In [3]: import matplotlib.pyplot as plt  
  
plt.plot(true_y)  
plt.plot(predicted_y)  
plt.legend(['Real', 'Predicho'])  
plt.show()
```



Al igual que en el problema de clasificación, se observa una baja tasa de acierto en las primeras muestras y un crecimiento al final.

