# Data stream analysis with MOA software

Iñaki Inza, Usue Mori

## Introducción

Como parte del 'boom' del `BigData`, el paradigma del `data stream mining` ha reverdecido viejos laureles. Trabajos pioneros [5] de hace ya más de una década definieron lo que entonces era un escenario novedoso y lleno de retos:

- las instancias llegan en un flujo continuado e infinito;
- las intancias no se pueden almacenar y deben procesarse bien individual o por grupos ('chunks') una única vez para actualizar el modelo;
- disponibilidad limitada de memoria para tratar un grupo de casos;
- el modelo responde de manera inmediata a la llegada de instancias (e.g. predicción, clustering...);
- entorno dinámico: necesidad de adaptarse automáticamente a cambios temporales en la naturaleza del flujo de datos (i.e. 'concept drift' [7]).

Tras unos años en los que no gozó de un excesivo protagonismo en la comunidad 'machine learning', los nuevos tipos de datos en forma de streaming (e.g. IoT) le han vuelto a dar un papel protagonista. Tal y como afirman recientes trabajos relevantes en el área [6], nos encontramos en el momento de mover este tipo de modelos desde los laboratorios de investigación a la industria, tal y como ha ocurrido en los últimos años con los modelos tradicionales de 'machine learning'.

La literatura relacionada es muy amplia, con numerosos de algoritmos de aprendizaje. La tecnología, también acompaña. El conocido software `MOA 'Massive Online Analysis'` [2] ha sido todo un referente desde su lanzamiento, y siempre en continua actualización y añadido de nuevos algoritmos. Su libro asociado [1], toda una 'enciclopedia' en el área. Aún habiendo varios proyectos en `R` y `python-scikit-learn`, éstos no recogen a día de hoy la amplitud de algoritmos de MOA y en notable parte de su software es un interfaz a las funciones de modelado originales de `MOA`. Para un primer contacto con este tipo de datos, el interfaz de `MOA` es de gran ayuda.

El siguiente tutorial es fruto de la lectura y síntesis de las referencias previas, así como de mi experiencia con el software MOA. Nos centraremos en tres escenarios de aprendizaje para nuestro flujo de datos en streaming: clasificación supervisada, clustering. Resumiendo un subconjunto de los principales algoritmos de cada escenario. Y ejemplos prácticos de su ejecución en el software.

> Antes de seguir: 'cambia el chip'. Los datos, bien individualmente o en 'chunks', pasan. No se almacenan. No se puede disponer de todos los datos para posteriormente evaluar el modelo. La evaluación también debe ser 'durante el stream'. Lo único que se almacena permanentemente es el 'modelo', que se debe ir actualizando según avance el stream de casos.

Año:
Profesor(a): Iñaki Inza, Usue Mori
Teléfono: 943 01 50 26
E-mail: inaki.inza,usue.mori@ehu.eus

GIPUZKOAKO CAMPUSA
CAMPUS DE GUIPUZCOA
Paseo Manuel de Lardizabal, 1
20018 Donostia (Gipuzkoa)

## Very-very shortly: concept drift

Mientras que en la referencia [7] tienes un amplio recorrido en el mundo del 'concept drift', me atrevo a 'super-resumir', los 3 tipos de 'concept drift'. La Figura 1, extraída del citado trabajo, siendo $X$ el vector de predictoras e $y$ la variable clase, nos muestra cómo:

- Escenario 1: existe un tiempo $t$ en el stream a partir del cuál $p_t(X) \neq p_{t+1}(X)$, mientras que $p_t(y|X) = p_{t+1}(y|X)$ continúa inalterada;
- Escenario 2: existe un tiempo $t$ en el stream a partir del cuál $p_t(y|X) \neq p_{t+1}(y|X)$, mientras que $p_t(X) = p_{t+1}(X)$ continúa inalterada;
- Escenario 3: en el que se da el 'concept drift' de ambos escenarios previos.

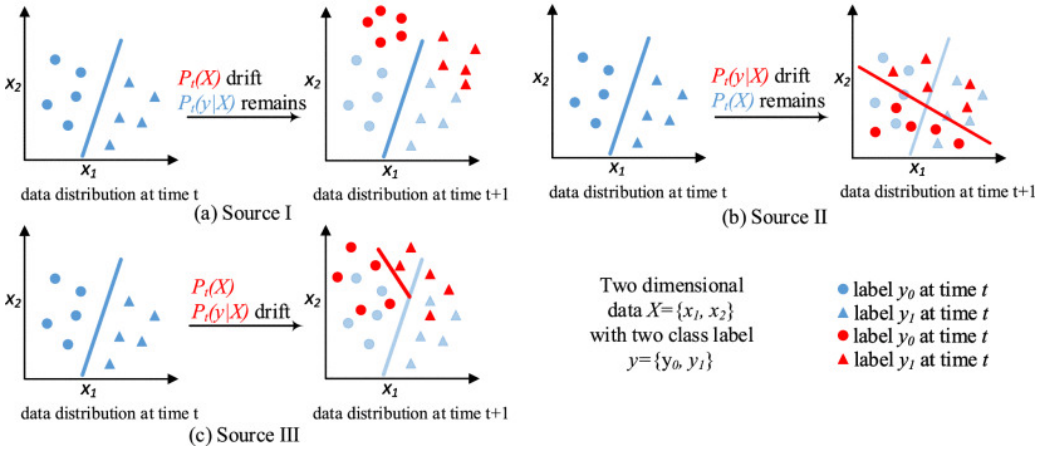


Figura 1: Tres tipos de 'concept drift' [7]

Otra característica del 'concept drift' a tener en cuenta:

- ocurre de manera abrupta: a partir de un instante temporal todas las instancias del stream lo sufren con el mismo grado de variación respecto a la situación previa;
- gradual: a partir de un instante se va intercalado la llegada de instancias sin y con 'concept drift'. Hasta que, a partir de un instante más avanzado, se estabiliza el stream y todas lo sufren;
- incremental: a partir de un instante todas las instancias del stream lo sufren, pero el grado en el que se da la variación aumenta incrementalmente. Hasta que, a partir de un instance más avanzado, se estabiliza el grado de variación respecto a la situación sin 'concept drift'.

Entiendo que los algoritmos más interesantes dentro del 'data stream analysis' son aquellos que incorporan mecanismos para modelar el citado 'concept drift'; no todos lo hacen. Veremos posteriormente ejemplos de éstos para cada tipo de escenario.

## Streaming Datasets

Cada aplicación de data streaming es única y merece una descripción propia: la frecuencia de generación de los datos, la existencia o no de 'concept drift', la disponibilidad de anotación en los datos, etc. Hacer 'setup' de una aplicación en un escenario real (e.g. IoT) requiere una adecuación específica. Los clásicos 'benchmark' que recogen aplicaciones reales los encuentras en este enlace. También es cierto que cualquier dataset real de

un tamaño muestral notable puede ser 'convertido-tratado' a streaming.

Si es tu primera aproximación al escenario del 'data streaming', considero que los datasets sintéticos, de los cuáles conocemos su función-distribución generadora y su tipo de 'concept drift', son el punto para comenzar el aprendizaje. `MOA` nos ofrece un interesente conjunto de *streams sintéticos*, que vienen a ser un 'benchmark' en la comunidad. En el siguiente enlace se describen con detalle. Mientras que algunos de ellos no sufren un 'concept drift' en su evolución y por ello las 'fronteras-boundaries' de los patrones no varían a lo largo del stream, aquellos que evolucionan y tienen 'concept drift' son los que suponen un mayor reto para los algoritmos. Para algunos streams se ofrecen ambas versiones: con y sin 'concept drift'.

Te describo un subconjunto de ellos, aquellos que llaman más mi atención; en el enlace previo tienes una descripción detallada:

- *HyperplaneGenerator*. Que el punto se encuentre 'sobre' o 'bajo' un hyperplano multidimensional hace que el etiquetado del caso sea positivo o negativo. Se alimenta el stream de casos al simular-muestrear el hyperplano. Se crea 'concept drift' al cambiar los pesos del hyperplano, cambiando así la orientación y posición del hyperplano.
- *RandomRBFGeneratorDrift*. Se comienza con un número clusters generados aleatoriamente: centroide, desviación, peso. La pertenencia al cluster define el etiquetado. Se alimenta el stream de casos al simular-muestrear la configuración de clusters, cada uno modelado con una distribución Gaussiana multidimensional. Se genera 'concept drift' moviendo los centroides a velocidad constante.
- *LEDGeneratorDrift*. Los 7 segmentos (variables binarias) de un display-LED producen el dígito concreto (variable clase). Cada variable tiene una probabilidad del 10% de invertir su valor. Se genera 'concept drift' haciendo 'swaping' entre 4 de las variables.
- *SEAGenerator*. La función etiquetadora reside en la suma de las dos varibles relevantes: si ésta supera un umbral, el etiquetado es positivo (negativo, en caso contrario). Se genera un 'concept drift' abrupto al cambiar el valor del umbral.
- *ConceptDriftStream*. Simula un 'concept drift' al hacer una transición 'smooth' entre dos streams cualesquiera que tengan el mismo número de variables y valores clase. Esta transición de un stream a otro se modela elegantemente mediante una función sigmoidea para la cual hay que fijar el 'point of change $p$' y 'length of change $w$'.

## Clasificación supervisada en data streams

En este escenario el las secuencias-vectores del stream llegan etiquetados, y bien por 'chunks' o individualmente. Éstos deben ser utilizados tanto para actualizar el modelo (i.e. lo único que se guarda en memoria), así como para evaluarlo. Se ha propuesto una amplia batería de algoritmos. Un listado quasi-exhaustivo lo tienes en este enlace, con las referencias bibliográficas concretas de cada algoritmo. Describiré un subconjunto de ellos: entre éstos, los más relevantes en la comunidad, a mi entender. Algunos incorporan mecanismos explícitos para modelar el 'concept drift'.

En muchas aplicaciones no será posible disponer del etiquetado de todas las secuencias. En la literatura nos encontramos aproximaciones de 'semi-supervised classification', así como de 'active learning'. En la primera aproximación se asume que una pequeña porción del stream llega etiquetada, pero la algorítmica intrínseca al aprendizaje semi-supervisado hace uso también de los casos sin etiquetar, basándose en alguna aproximación sobre la distribución de los datos (e.g. ' 'clustering assumption', muestras del mismo cluster comparten etiqueta). En 'active learning', es el propio algoritmo el que decide qué instancias solicitar al experto para que sean etiquetadas. En la sección 3.2 de la referencia [6] encontrarás referencias clave de cada escenario.

Antes de seguir con algorítmicas más elaboradas, ten en cuenta los siguientes dos clasificadores 'baseline'. *Majority class* predice la clase más frecuente en el histórico del stream. *No-change classifier*, para el siguiente caso a clasificar, lo predice con la clase real del caso previo en el stream: en algunas aplicaciones, debido a su naturaleza, puede ser una aproximación efectiva, ya que únicamente cometerá errores en los 'casos frontera' y rápidamente recupera el patrón de etiquetado del stream.

Como otras dos propuestas 'baseline', primero el popular algoritmo *naive Bayes*. Éste, al basarse en una estructura de dependencias probabilistas fija (i.e. no requiere aprendizaje estructural), computar las probabilidades y contadores que requiere se realiza de manera incremental: adecuado para un escenario streaming. En cambio, no incluye un mecanismo explícito para adaptarse con rapidez a un 'concept drift' abrupto si es que no limitamos el aprendizaje del modelo a una reciente ventana de datos. Como segundo 'baseline', la comunidad adapta para 'data stream analysis' el popular *k-vecinos más próximos* utilizando una 'sliding window' (ventana deslizante) de habitualmente los $1,000$ últimos casos del stream para buscar en ella los vecinos más cercanos y así responder de manera natural al 'concept drift'.

Los *árboles de clasificación* han sido la aproximación 'de facto' en 'data stream classification', y entre éstos los populares *Hoeffding trees*. Se basan en un resultado estadístico conocido como 'Hoeffding bound'. Éste aporta una cota superior a la probabilidad de que la suma de variables aleatorias independientes ($s$) se desvíe respecto a su valor esperado ($X$):

$$p(|X - s| > \epsilon) \leq \delta$$

En la literatura de 'data stream' se ha optado por fijar el valor de la diferencia, $\epsilon = \sqrt{\frac{R^2 \ln \frac{1}{\delta}}{2n}}$, siendo $n$ el número de casos y $R$ el rango de la variable (e.g. $\log c$ en el caso de la entropía como medida de 'split' en los árboles, siendo $c$ el número de clases). El valor de la cota superior, $\delta$, un valor de probabilidad bajo fijado por el usuario (e.g. 0.05).

Esta cota se utiliza en el siguiente contexto: para realizar un 'split' en una hoja del árbol y así incrementar su profundidad, se exige que la diferencia entre los scores (e.g. información mutua, Gini index, etc.) del mejor y segundo atributo sea como mínimo de este valor para proceder con el 'split'. Éste es el mecanismo para modelar el 'concept drift' del stream, y sólo proceder con una especialización del árbol (i.e. incrementar el nivel de profundidad) cuando el atributo candidato a hacer el 'split' es para ello notablemente mejor (i.e. Hoeffding bound) que el resto. Según avanza el stream los estadísticos previos se actualizan incrementalmente de manera natural en cada hoja final del árbol y para cada atributo predictor.

Una actualización de este algoritmo es el popular *Very Fast Decision Tree (VFDT)* [3], ya incorporada en MOA a la implementación del *Hoeffding Tree*. Para acelerar el proceso, en vez de actualizar los estadísticos necesarios a la llegada de cada instancia individual, este proceso se realiza con la llegada de una ventana de $n_{min}$ número de instancias. VFDT también desactiva ('dropout') los nodos con alto error; y puede inicializarse su aprendizaje a partir de un árbol aprendido sobre una muestra 'batch' etiquetada.

Su popular versión *Hoeffding Adaptive Tree* incorpora otros mecanismos para modelar el 'concept drift'. Con la llegada de cada $T_0$ instancias, a partir de éstas se re-evalúa si cada 'splitting attribute' es el mejor en cada

nodo del árbol. De no ser así, las siguientes $T_1$ instancias del stream se utilizan para crear un sub-árbol alternativo a partir de dicho nodo actualizado. Y las siguientes $T_2$ instancias se utilizan para evaluar si el sub-árbol alternativo es mejor que el previo: de ser así, se reemplaza finalmente.

Una alternativa flexible e independiente de cualquier algoritmo de clasificación concreto, *SingleClassifierDrift* detecta un 'concept drift' a partir de monitorizar la evolución del error del clasificador escogido: eliminando el modelo previo y aprendiendo uno nuevo. La literatura nos ofrece múltiples formas de monitorizar el error y alertarnos de un cambio en su comportamiento. Entre éstas, 'Drift Detection Method' (DDM) [4] es de las más populares e intuitivas. Declara 'concept drift' cuando:

$$p_t + s_t \geq p_{min} + 3 \cdot s_{min}$$

donde $p_t$ es el error del modelo actual aprendido sobre las últimas $t$ instancias del 'stream' desde el último cambio; $s_t$ la desviación standard del error que, al ser éste modelado por una distribución binomial, $s_t = \sqrt{\frac{p_t(1-p_t)}{t}}$; $p_{min}$ el mínimo error observado en el 'stream' a partir del tiempo $t$. En el momento de declararse el 'concept drift', el modelo en vigor hasta el tiempo $t$ es descartado y el nuevo modelo en vigor es aquel aprendido con los vectores desde que se declaró un 'warning'; éste fue declarado cuando:

$$p_t + s_t \geq p_{min} + 2 \cdot s_{min}$$

en anticipación a un posible y cercano 'concept drift' futuro. Los valores $p_t$, $p_{min}$ son reseteados y el 'stream' sigue su curso.

**→ Evaluación de clasificadores en data stream**

Antes de seguir con otros escenarios de aprendizaje, no podemos dar ese salto sin tratar la evaluación de clasificadores. Las técnicas de validación se basan en las clásicas que conocemos del 'offline learning': pero incorporan necesarios mecanismos y matices para adaptarlas al 'online learning'. Estamos en una aplicación de llegada continua (e infinita) de datos, y no podemos esperar a disponer de todos ellos para evaluar el modelo de clasificación que llevamos aprendido; y es imprescindible crear una monitorización-visualización del 'performance estimado' del modelo a lo largo del tiempo: 'i.e. picture of accuracy along time'.

*HoldOut.* Disponemos tanto de un 'stream de train' como otro 'stream de test': disjuntos, no solapados. Mientras que el 'stream de train' fluye entrenando-actualizando el modelo, periódicamente y con una frecuencia fijada, evaluamos su 'performance' en el 'stream de test'.

*Interleaved test-then-train.* Cada instancia individual llegada en el stream, y antes de proceder con la siguiente, es utilizada primero para evaluar el modelo, y posteriormente para re-entrenarlo-actualizarlo. Se trata de una evaluación honesta: en el momento de evaluación, la instancia no ha sido 'vista' por el modelo. Se trata de actualizar incrementalmente, de forma 'smooth-suavizada', el 'picture of performance'.

*Prequential.* Se diferencia del método previo en dar un mayor peso para el cálculo del 'score' de evaluación a ejemplos recientes. Esto se realiza mediante un 'decay factor' individualizado para los casos previos o una ventana dezlizante ('sliding window').

*Interleaved chunks.* Como el método 'interleaved' previo pero, en vez de con instancias individuales, con 'chunks' de instancias consecutivas del 'stream'.

**→ MOA software: clasificación supervisada + evaluación**

Antes de pasar con los siguientes escenarios de modelado, prefiero 'tocar tierra' con el software y ver cómo se ejecutan los conceptos que hemos visto hasta ahora en clasificación supervisada: ya tenemos 'streaming datasets', conocemos varias técnicas de clasificación supervisada en 'streaming', y nociones de cómo evaluar los modelos. No hay excusa.

MOA [1] es un software fantástico para 'data stream' analysis. Y aunque están brotando iniciativas en python y R a las que habrá que seguir la pista, es a día de hoy el 'software de facto'. Escrito en Java, podemos lanzar su GUI con la orden `java -jar moa.jar`. Nos ofrece tres modos de uso: desde intuitivos menús desplegables y pestañas en la propia GUI, desde líneas de comandos tanto en la propia GUI o el prompt del sistema, o desde código Java externo interactuando con su API.

Mediante el botón 'Configure' del GUI y sucesivas pestañas y botones, podemos llevar adelante el aprendizaje: especial atención a los conceptos-botones de 'learner' (i.e. algoritmo de clasificación) y 'stream' (el dataset). El botón de 'Edit' permite cambiar la selección y los parámetros propios correspondientes. El botón de 'Help' ofrece una breve ayuda sobre cada parámetro. Al clickar en él puedes observar que cada parámetro tiene un identificador concreto. Por ejemplo, `-l learner` o `-s stream`. Al ir cambiando con los botones tus preferencias, se va creando un comando al lado del botón 'Configure', parametrizado con estas opciones precedidas del símbolo negativo: esta orden será la que se ejecutará (botón 'Run'). Ésta será la forma de uso que mostraremos en los siguientes ejemplos: accedes a ella clickando con el botón derecho sobre la propia orden, y posteriormente

`Copy configuration to the clipboard` → `Enter configuration` → `Ctrl+V`

Ahí tienes el comando y puedes editarlo. Cuando tengas un poco de dominio, el comando lo editarás por completo. No te asustes. Es sencillo. Cada software tiene sus peculiaridades.

Evaluamos mediante *HoldOut* el clasificador *HoeffingAdaptiveTree* (`-l`). La evaluación se realiza sobre las primeras $100,000$ instancias del stream (`-n`), con 'snapshots' del performance cada $500,000$ muestras del ' training stream' (`-f`), y éste formado por $7,500,000$ vectores (`-i`). Sobre el stream *HyperplaneGenerator* (`-s`).

```
EvaluatePeriodicHeldOutTest -l trees.HoeffdingTree -s generators.HyperplaneGenerator -n 100000
   -i 7500000 -f 500000
```

Este mismo comando puedes pegarlo en el 'command line' de MOA, y editarlo, clickando con el botón derecho y 'enter configuration'. Fíjate en la salida en forma de tabla: sus columnas, llenas de 'scores'. Es posible exportarla y estudiarla con detenimiento en otro programa (e.g. hoja de cálculo, `R`, etc.). Desde la base del comando previo, éste puede ser editado y por ejemplo comparar sobre el mismo 'stream setting' el comportamiento de otro tipo de clasificador. No hay mucho más misterio. Ésta es la base: puedes cambiar el procedimiento de evaluación (e.g. `EvaluateInterleavedTestThenTrain`), el algoritmo de clasificación (e.g. `bayes.NaiveBayes`), el 'stream dataset' (e.g. `RandomRBFGeneratorDrift`)... una vez partes de esta base, tienes todo un mundo de recursos en MOA para ser descubiertos por ti mismo, tanto en el 'command line' como en el GUI.

Si quieres cambiar los parámetros del algoritmo de clasificación o el 'stream dataset' te recomiendo hacerlo sobre el GUI de MOA y sus botones de edición. No son pocos los parámetros: una breve ayuda sobre éstos con el botón *Help*.

## Clustering en data streams

Una primera familia de algoritmos de clustering disponibles en MOA pertenecen a la familia del *clustering particional* fundamentado en el popular 'k-means'. Aquí tenemos algoritmos como *BIRCH (Balanced Iterative Recuding and Clustering using Hierarchies), CLUSTREAM, CLUSTREE* y *StreamKM++*. Para adaptarse al

escenario del 'data streaming', se basan en el concepto de 'microcluster'. Éste viene a ser una estructura cohesionada de un número reducido de puntos para los cuales se almacenan estadísticos como su número de casos, centroide, radio, nivel de cohesión interna, etc.: esto es, vienen a ser 'hyperesferas' en espacios d-dimensionales. Los 'microcluters' se actualizan 'online' en el avance del 'stream', siendo los casos asignados al 'microcluster' más cercano.

Algunos de estos algoritmos almacenan el conjunto de 'microclusters' en una estructura de árbol. Cuando un nuevo caso del 'stream' llega a una hoja del árbol y no se encuentra dentro del radio del 'microcluster' ahí almacenado, entonces se crea una nueva hoja para dicho punto en la estructura de árbol.

Todos los algoritmos incorporan mecanismos de unión de 'microclusters' cercanos, así como de borrado de 'microclusters' obsoletos: aquellos más antiguos o sin actividad de actualización reciente. La unión final de 'microclusters' en la última etapa del algoritmo configura la partición final de clusters.

Una segunda familia de algoritmos son los basados en *densidades*: *DBSCAN, Den-Stream.* No sólo basándose en el concepto de distancia-cercanía, también explotan dinámicamente la *densidad de las conexiones entre puntos.* El $\epsilon$-'neighborhood' de un vector $p$ está formado por aquellos puntos a distancia menor que $\epsilon$: y el punto $p$ es considerado 'core point' si al menos recoge una fracción $\mu$ de vectores en este vecindario. Los puntos no accesibles desde 'core points' son considerados 'outliers'. En la llegada de casos en el 'stream', éstos se asignan a dichos vecindarios: en caso de no ser posible, se trata de unirlos al de los 'outliers'. Éstos últimos dejan de ser considerados 'outliers' cuando recogen un número mínimo de puntos en su vecindario. Para cumplir con los requisitos de memoria del sistema, los puntos decrecen exponencialmente su peso mediante un 'decay factor', hasta el momento de desaparecer del modelo de clustering dinámico.

→ **MOA software: clustering**

El software no tiene en este escenario 'command line', y nos ofrece un intuitivo GUI. Tras escoger el 'data stream' concreto, nos ofrece la posibilidad de lanzar dos algoritmos en paralelo y así compararlos. Te recomiendo empezar por ejecutar sólo uno. Por defecto nos ofrece el ya expuesto 'stream' *RandomRBFGeneratorDrift*: con su parámetro `noiseLevel` implementa un 'concept drift' moviendo los centroides de los clusters a lo largo del tiempo. Al ser un 'stream' en 2-D, podemos utilizar con intuición las dos pestañas que nos ofrece MOA: `Setup` y `Visualization`. Mientras que en la primera parametrizamos el algoritmo a ejecutar (o pareja de métodos), en la segunda nos ofrece una evolución temporal de la ejecución. En ésta, junto a una larga batería de 'scores', podemos observar la evolución temporal de los clusters aprendidos (`Clustering`), *versus* los clusters reales (`Ground truth`).

Ten en cuenta que en el 'stream' de *RandomRBFGeneratorDrift* es conocida la clase-grupo de pertenencia de cada punto (esto no es así en una aplicación realista de clustering): este `GroundTruth` es utilizado para 'evaluar' la configuración de clustering aprendieda.

Los algoritmos 'sufren mucho' cuando el número de clusters reales no es el de la $k$, i.e. número de clusters a aprender. En la pestaña de visualización, la ejecución, por defecto, se pausa cada $5,000$ puntos del 'stream': debes pulsar en el botón 're-start' para que siga fluyendo el 'stream'.

Todo un 'playground' para jugar y aprender.

# Bibliografía

[1] A. Bifet, R. Gavaldà, G. Holmes, and B. Pfahringer. *Machine Learning for Data Streams with Practical Examples in MOA*. MIT Press, 2018. `https://moa.cms.waikato.ac.nz/book/`.

[2] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer. MOA: massive online analysis. *Journal of Machine Learning Research*, 11:1601–1604, 2010.

[3] P. Domingos and G. Hulten. Mining high-speed data streams. In *Proceedings of Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80, 2000.

[4] J. Gama, P. Medas, G. Castillo, and P.P. Rodrigues. Learning with drift detection. In *Proceedings of the 17th Brazilian Symposium on Artificial Intelligence*, pages 286–295, 2004.

[5] J. Gama and M. Mehdat, editors. *Learning from Data Streams*. Springer, 2007.

[6] H.M. Gomes, J. Read, A. Bifet, J.P. Barddal, and J. Gama. Machine learning for streaming data: state of the art, challenges, and opportunities. *ACM SIGKDD Explorations Newsletter*, 21(2):6–22, 2019.

[7] J. Lu, A. Liu, F. Dong, F. Gu, J. Gama, and G. Zhang. Learning under concept drift: a review. *IEEE Transactions on Knowledge and Data Engineering*, 31(12):2346–2363, 2019.