

# Minería de Data Streams: Conceptos y Principales Técnicas

Francisco J. Ferrer–Troyano<sup>1</sup> y Jesús S. Aguilar–Ruiz<sup>1</sup>

<sup>1</sup> Universidad de Sevilla  
aguilar@lsi.us.es

<sup>2</sup> Universidad de Sevilla  
ferrer@lsi.us.es

**Resumen** La evolución del incremento de volumen de los datos y la velocidad a la que éstos se reciben están propiciando el desarrollo de nuevas técnicas de minería de datos capaces de generar modelos de conocimiento (reglas de decisión, árboles de decisión, etc.) a partir de datos que se reciben a gran velocidad, los cuales no pueden ser almacenados todos ellos para ser analizados de forma global. Las técnicas para data streams abordan directamente las características relacionadas con el tamaño inmanejable de los datos, su velocidad de llegada y las limitaciones en cuanto a recursos para diseñar algoritmos eficientes. A continuación se presentan los conceptos más importantes y las técnicas más relevantes dentro del área de data streams.

## 1 Introducción

Los sistemas de decisión *off-line* procesan repetidas veces el conjunto de entrenamiento para construir un único modelo final. Es por ello que los esquemas inductivos en los que se basan parten de tres supuestos:

- Todos los ejemplos necesarios para describir el dominio del problema están disponibles antes del proceso de aprendizaje.
- Todos los ejemplos de entrenamiento pueden ser cargados en memoria.
- Tras procesar debidamente el conjunto de entrenamiento, el aprendizaje puede darse por finalizado.

El éxito de las técnicas *off-line* se sustenta en el hecho de que, durante décadas, la cantidad de datos disponibles para el análisis era tan reducida que podía ser fácilmente almacenada en simples ficheros planos. Sin embargo, tras la celebración del primer congreso internacional de KDD [1], la comunidad investigadora en minería de datos destacó la necesidad de desarrollar nuevos sistemas capaces de modelar bases de datos reales de gran escala (*very large databases*). Para hacer frente a volúmenes de tal magnitud, a finales de la década de los 90 aparecieron diversas propuestas para adaptar y hacer escalables los principales sistemas de decisión basados en reglas [2]. Aun así, un número creciente de aplicaciones cuestiona diariamente la capacidad y utilidad de las técnicas escalables.

Impulsado por la automatización en los procesos de adquisición y almacenamiento de datos, en la actualidad ninguna de las tres condiciones anteriores puede aceptarse en numerosos dominios de interés donde los datos proceden de entornos dinámicos y van siendo adquiridos a lo largo del tiempo. Este hecho obliga a los sistemas KDD a procesarlos secuencialmente y a adaptar el modelo, conforme varía su distribución de probabilidad, en sucesivos episodios, de forma *incremental*. De mayor ubicuidad hoy día que el aprendizaje por lotes, ejemplo de cuatro grandes áreas de extensa aplicación mediante procesos incrementales, no exclusivas a tareas de aprendizaje y minería de datos, son [3]:

- **Perfiles de usuario y detección de intrusos.** Diversos estudios coinciden en el hecho de que el comportamiento y los intereses de un usuario varían radicalmente cada seis meses. En general, la detección de intrusos en redes y la clasificación de usuarios requiere la monitorización individual de cada una de las entradas en el sistema [4].
- **Robótica de exploración.** Aun en el más sencillo de los casos, el entorno de un robot de exploración es a menudo imprevisible. Así por ejemplo, tomando como objetivo una navegación libre de colisiones, un robot debe ser capaz de reaccionar y adaptarse incrementalmente a las nuevas señales del entorno [5].
- **Agentes Inteligentes.** Definidos a muy grandes rasgos como aplicaciones software con capacidad reactiva y proactiva, al igual que en la robótica de exploración el carácter incremental es inherente al proceso de aprendizaje de todo agente. Ejemplo de aplicaciones basadas en agentes donde se lleva a cabo un proceso de aprendizaje incremental son las interfaces de usuario inteligentes [6] y la gestión del tráfico y optimización del rendimiento en redes.
- **Estimación de métricas en proyectos de desarrollo de software.** La estimación del coste, el esfuerzo y la duración de un proyecto software es en gran medida cuestión de experiencia. Debido a la duración de los mismos, por muy útiles que sean los datos disponibles, éstos van siendo generados en etapas de tiempo relativamente largas, por lo que en principio podría parecer un problema ajeno al aprendizaje incremental. De hecho, varios estudios coinciden en que la construcción de una base de conocimiento adecuada para obtener una estimación con un nivel de certeza satisfactorio puede llevar hasta tres años. Aun así, en la toma de decisiones el interés está en poder disponer de distintos modelos para cada una de las fases del proyecto y poder así reajustar predicciones a corto plazo en función de los cambios ocurridos en modelos pasados y de las características de los datos actuales. En este sentido, el carácter del problema es puramente incremental [7].

Aunque abordadas también durante años y cada vez menos por sistemas *off-line*, otras áreas de amplia y más reciente aplicación para sistemas incrementales son el reconocimiento de patrones de compra en grandes almacenes [8], modelos de comportamiento para movimientos bursátiles, organización y rankings de *e-mails* [9], análisis de imágenes en climatología y control medioambiental [10] y planificación de procesos [11]. Sin embargo, tanto las recientes ampliaciones

escalables de los sistemas *off-line* como muchas de las principales técnicas de carácter incremental están pasando en pocos años a ser impracticables en éstas y otras muchas áreas gracias a los recientes avances en telecomunicaciones inalámbricas y dispositivos empotrados. Diariamente, el número de fuentes y de datos disponibles para su análisis aumenta vertiginosamente, trasladándose inmediatamente a un incremento en la velocidad con la que los nuevos ejemplos de entrenamiento son recibidos por los sistemas KDD en forma de flujos o secuencias permanentes. Ejemplo de estos flujos de datos continuos o *data streams* son eventos y *logs* en redes, datos vía satélite y registros en telecomunicaciones o transacciones comerciales y financieras. Bajo tales circunstancias, no es posible recopilar y filtrar a tiempo todos los ejemplos necesarios para el proceso de aprendizaje, exigiendo a los sistemas KDD no sólo a operar *on-line* de forma continua sino a procesar cada *item* en tiempo real [12,13]. Y aun siendo posible almacenar todos los ejemplos en disco, las limitaciones de memoria y las restricciones en el tiempo de respuesta quiebran la lógica inductiva *multi-pass* a la que atienden los algoritmos escalables y muchos de los algoritmos incrementales [14,15].

## 2 Aprendizaje Incremental y Sistemas On-Line

El adjetivo incremental ha sido aplicado tanto a la naturaleza del problema como al esquema de aprendizaje, provocando la confusión entre dos conceptos diferentes. Una tarea de aprendizaje se dice de carácter incremental si el conjunto de entrenamiento no está disponible a priori sino que, generados a lo largo del tiempo, los ejemplos van siendo procesados secuencialmente obligando al sistema a aprender en episodios sucesivos. A este respecto es importante señalar los tres factores principales que determinan el diseño de un algoritmo incremental:

- **La influencia del orden de llegada.** Aunque los ejemplos no son necesariamente dependientes del tiempo (como lo son por ejemplo en el caso de las series temporales), según el problema puede ser necesario tener en cuenta su momento de llegada, integrando la dimensión temporal bien como un atributo del conjunto de entrenamiento, o bien como un parámetro relevante para el proceso de aprendizaje.
- **La adaptación al cambio.** Los sistemas de aprendizaje por lotes asumen que el conjunto de entrenamiento disponible encierra toda la realidad. Este supuesto es conocido como *mundo cerrado* frente a su opuesto *mundo abierto* del que parten los algoritmos incrementales. Pese a ser en teoría altamente conveniente pues simplifica en gran medida el proceso de aprendizaje, dicho supuesto no se sostiene en la mayoría de las aplicaciones reales, donde, independientemente del volumen de datos disponible, éstos generalmente presentan incertidumbre debido al ruido o a valores nulos, incompletos o inconsistentes. Además, a medida que nuevos datos van siendo adquiridos la función objetivo puede variar, de forma que reglas válidas para el pasado han dejado de serlo en la actualidad.

- **La curva de aprendizaje.** En general los sistemas de aprendizaje parten de un modelo vacío y van ampliándolo, actualizándolo y simplificándolo con los nuevos ejemplos procesados. Como resultado, aunque un sistema incremental puede realizar predicciones en todo momento, éstas no poseen un nivel de confianza satisfactorio en épocas tempranas.

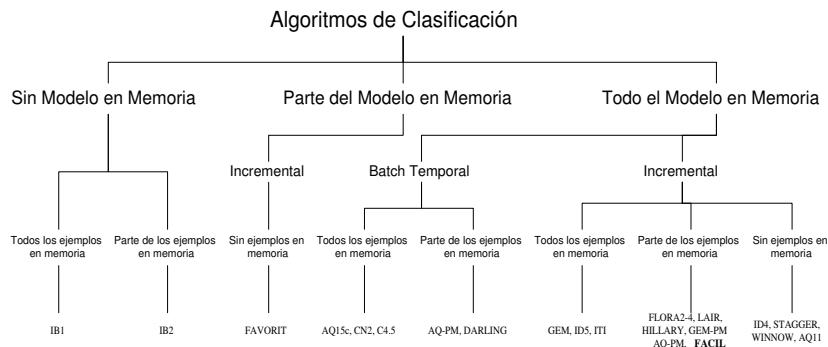
Es importante señalar que no todos los sistemas *on-line* se basan en un esquema de aprendizaje incremental.

En general, un sistema *on-line* es aquél capaz de dar respuesta en todo momento. Aceptando el segundo supuesto del aprendizaje por lotes en el dominio de un problema incremental, en principio éste podría abordarse mediante un sistema *off-line*. Bastaría para ello con desechar el modelo obtenido y aplicar el algoritmo de aprendizaje cada vez que se reciben nuevos ejemplos, añadiendo éstos al conjunto de entrenamiento [16,17,18,19]. La única ventaja de este esquema - conocido como *temporal-batch* - es que no se ve influenciado por el orden de llegada de los ejemplos. Sin embargo, junto con su alta ineficiencia, el gran inconveniente de estas técnicas - *full instance memory systems* - es su pobre rendimiento ya que, en dominios dinámicos reales, no se cumple el tercer supuesto del aprendizaje por lotes. Debido a que la función objetivo cambia a lo largo del tiempo, dichos algoritmos intentan erróneamente mantener la consistencia entre un modelo *obsoleto* y nuevos ejemplos que describen a nuevos conceptos.

Por contra, un algoritmo de aprendizaje puramente incremental tan sólo tiene en cuenta los ejemplos recibidos en cada instante.

**Definición 21 (Algoritmo de Aprendizaje Incremental)** Sea  $T_t = \{(\vec{x}, y) : y = f(\vec{x})\}$  el conjunto de ejemplos de entrenamiento disponibles en un instante  $t \in \langle 1, 2, \dots \rangle$ . Un algoritmo de aprendizaje se dice puramente incremental si a partir de una secuencia  $\langle T_1, T_2, \dots, T_i \rangle$  produce una secuencia de hipótesis  $\mathcal{H} = \langle \mathcal{H}_0, \mathcal{H}_1, \dots, \mathcal{H}_i \rangle$  donde la hipótesis actual  $\mathcal{H}_i$  es función de la hipótesis anterior  $\mathcal{H}_{i-1}$  y el conjunto de ejemplos leídos en el instante actual  $T_i$ .

Paralelamente, todo sistema cuyo esquema de aprendizaje es puramente incremental (también *no instance memory systems*) puede aplicarse a un problema no incremental. En este caso, el bajo rendimiento no se refleja en el coste computacional sino en la exactitud del modelo. Al utilizar únicamente los ejemplos recibidos en cada instante, los algoritmos incrementales no aprovechan toda la información inherente al conjunto de entrenamiento, a priori disponible por completo. Para tales dominios se dice que los algoritmos incrementales padecen de *miopía*, induciendo únicamente información local cuando se dispone de información global. Por otro lado, si el tiempo no es un factor relevante en el dominio del problema, el rendimiento obtenido por tales sistemas se ve fuertemente influenciado por el orden de llegada de los mismos. En cualquier caso, el coste computacional de un algoritmo incremental en la resolución de un problema incremental será siempre inferior al coste computacional de un algoritmo no incremental.



**Figura 1.** Clasificación de las principales técnicas del aprendizaje supervisado.

En [17] se demuestra que, dados  $m$  atributos y  $n$  ejemplos de entrenamiento, la versión incremental de ID3 (ID5) debe examinar  $\mathcal{O}(nm^2)$  atributos, mientras que ID3 re-entrenado con cada nueva instancia termina evaluando  $\mathcal{O}(n^2m^2)$  atributos. En [20] se exponen diferentes esquemas para adaptar a problemas incrementales técnicas *off-line* basadas en vectores de soporte. En principio, como solución inicial cabría pensar en descartar todos los ejemplos retenidos cuando se detecta un cambio en la función objetivo. Ésta sería una solución válida únicamente cuando los ejemplos pasados describen conceptos disjuntos con respecto a los descritos por los nuevos ejemplos. Ahora bien, si los conceptos pasados *solapan* con los nuevos, puede ser útil utilizar algunos de los ejemplos pasados, ayudando a converger con mayor rapidez y exactitud hacia la nueva función objetivo. Por este motivo, un tercer esquema más extendido en la literatura consiste en retener un subconjunto de ejemplos  $\mathcal{T}'$  pertenecientes a episodios pasados y a su vez consistentes con la hipótesis anterior  $\mathcal{H}_{i-1}$ , de forma que la hipótesis actual  $\mathcal{H}_i$  es función de  $\mathcal{H}_{i-1}$ , de los ejemplos recibidos en el instante actual  $\mathcal{T}_i$  y de un subconjunto de ejemplos de  $\mathcal{T}'$  (*partial instance memory systems*) [21,22]. Mediante este enfoque se garantiza una mayor rapidez en la convergencia y a él pertenece nuestra propuesta. La figura 1 muestra una clasificación de las principales técnicas del aprendizaje supervisado en función de los ejemplos retenidos en memoria.

### 3 La influencia del contexto

El principal problema al que se enfrentan los sistemas de aprendizaje en dominios de carácter incremental es el hecho de que la función objetivo puede depender del contexto, el cual no es recogido mediante los atributos de los datos de entrada. Un clásico ejemplo de ello es el problema de la predicción climatológica, donde las reglas a generar pueden variar radicalmente entre las distintas estaciones. Otro ejemplo es la obtención de patrones de compra entre los clientes de las grandes superficies, susceptibles a variaciones permanentes en función del

día de la semana, la época del año, factores económicos como la inflación, la disponibilidad de los productos, etc. En tales situaciones, la causa del cambio se dice oculta y el problema se conoce como *hidden context*. En consecuencia, cambios ocurridos en el contexto pueden inducir variaciones en la función objetivo, dando lugar a lo que se conoce como *concept drift*<sup>3</sup> [23,24]. Un factor decisivo en el rendimiento del sistema es el balance o *trade-off* entre la robustez al ruido y la sensibilidad al cambio [24].

Para agravar aun más si cabe las dificultades del aprendizaje incremental, en muchos dominios es de esperar que el contexto sea recurrente debido tanto a fenómenos cíclicos - p.e. las estaciones del año para la predicción climatológica - como a fenómenos irregulares - p.e. la tasa de inflación o la atmósfera de mercado para el modelado de perfiles de compra - [25]. Siendo muy reducido el número de técnicas existentes en la literatura diseñadas para poder adaptarse a contextos recurrentes (FLORA3 [26], PECS [27], SPLICE [25] o Local Weights and Batch Selection [20]), en general basan su estrategia en retener modelos pasados en memoria de forma que pueden ser revisados en todo momento y volver de actualidad con lo que se acelera la adaptación al cambio. Por tanto, junto a la robustez y la sensibilidad anteriormente mencionadas, el *trade-off* de un sistema incremental lo completa la capacidad de reconocer contextos recurrentes.

## 4 Taxonomía del cambio

En función de la frecuencia con la que los nuevos ejemplos descriptivos de la nueva función objetivo son recibidos, en general se distinguen en la literatura dos tipos de cambio o *concept drift*: (1) **abrupto** (repentino, instantáneo) y (2) **gradual**. En [28] Stanley divide a su vez el cambio gradual entre moderado y lento. Paralelamente, la dependencia del contexto no sólo puede inducir variaciones en la función objetivo sino que además puede cambiar la propia distribución de los atributos de los ejemplos de entrenamiento. Cuando el cambio contextual afecta únicamente a los datos de entrada éste se dice **virtual** [26,27] (*sampling shift*), mientras que el cambio es **real** cuando únicamente induce un movimiento del concepto objetivo (*concept shift*). En la práctica es irrelevante el tipo del cambio ya que ambos producen un impacto en el modelo, en cuanto aumentan el error del mismo con respecto a los ejemplos actuales y lleva a la necesidad de revisarlo constantemente.

El cambio virtual está presente en todas las tareas del aprendizaje incremental, no siendo excluyente con respecto al cambio real y pudiendo suceder ambos simultáneamente. Por ejemplo, en la clasificación de *spam*, las reglas de usuario que determinan qué es correo no deseado suelen permanecer invariantes a lo largo de periodos de tiempo relativamente largos. Sin embargo, la frecuencia relativa de los diferentes tipos de *spam* puede cambiar de forma drástica y sucesiva durante dichos periodos. Este último cambio es virtual y puede provocar un cambio real en la definición de las reglas de usuario.

---

<sup>3</sup> también *changing concepts, time-varying concepts o non-stationary environments*

## 5 Aprendizaje Incremental de Reglas

En general, el distintivo principal entre las técnicas incrementales basadas en reglas viene dado por el método empleado para determinar, según los cambios en la función objetivo, qué ejemplos son relevantes en cada momento. La importancia de este aspecto radica en la influencia de ejemplos *obsoletos* sobre el modelo actual, cuyos efectos reducen la exactitud y consistencia del mismo.

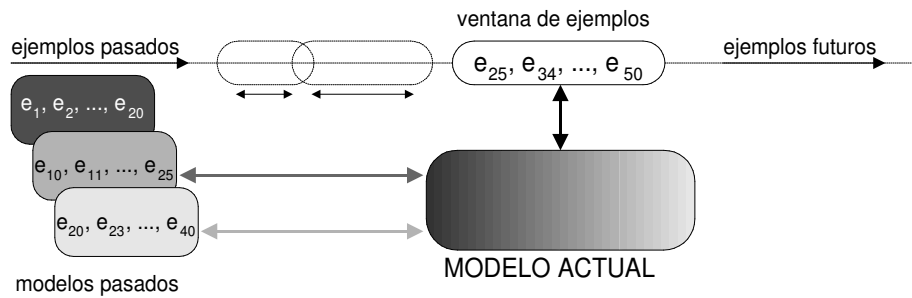
Entre las propuestas existentes en la literatura, el esquema de ventana es el más utilizado hasta la fecha - FLORA [29], FRANN - [24] o TMF [27] - y basa la generalización y actualización de las mismas en la consistencia con respecto a los últimos ejemplos recibidos. De tamaño fijo o variable, la ventana observa únicamente a los ejemplos más recientes (figura 2), por lo que tanto el modelo como las predicciones se limitan a un futuro cercano. El esquema de ventana de tamaño fijo consiste simplemente en *olvidar* aquellos ejemplos recibidos pasado un tiempo constante  $T$ . El principal problema de este esquema es que el rendimiento depende en alto grado del valor de  $T$ , ya que el patrón de cambio de la función objetivo  $f$  es desconocido, dando lugar al siguiente dilema:

- Si  $T$  es mayor que la frecuencia o la persistencia del cambio, los ejemplos de la ventana describirán una función objetivo *antigua*. Una tasa más alta bajaría la exactitud del modelo actualizado como es apoyada por una menor cantidad de datos del entrenamiento.
- Si  $T$  es menor que la persistencia del cambio, los ejemplos de la ventana serán insuficientes y, debido al sobreajuste, el modelo arrastrará una fuerte varianza. Una tarifa más baja haría el modelo menos sensible a la tendencia actual y evitaría que descubriera patrones transitorios.

Por contra, dicha dependencia es eliminada si el tamaño de la ventana puede ajustarse de forma dinámica - ventana deslizante -, creciendo cuando el modelo gana exactitud y decreciendo en caso contrario. De esta forma, el cambio en  $f$  es detectado por un aumento en la inconsistencia entre el modelo y los ejemplos de la ventana, reduciéndose el tamaño de la misma para así descartar ejemplos no consistentes. Aún así, ambos esquemas presentan dos severos inconvenientes:

- No puede detectar ni modelar cambios continuos de alta frecuencia o de frecuencias variables, ya que el modelo sería actualizado constantemente y no podría por tanto converger a una función consistente con los ejemplos observados.
- Alta sensibilidad a cambios virtuales [30].

En [31] Kuh et al. determinan la máxima frecuencia  $t$  con la que pueden cambiar los conceptos objetivo para poder llevar a cabo el aprendizaje, determinando en función de  $t$  un límite inferior para el tamaño de la ventana a utilizar. Hembold & Long [32] establecen límites sobre la magnitud tolerable asumiendo cambios persistentes y a su vez graduales. La magnitud  $\epsilon$  es definida como la probabilidad de que dos conceptos sucesivos no coincidan en un ejemplo aleatorio. Los resultados obtenidos en [32] y [31] coinciden en que basta con una ventana fija



**Figura 2.** Selección de ejemplos relevantes mediante el esquema de ventana.

que contenga a los ejemplos más recientes. En la práctica sin embargo no puede garantizarse que tales restricciones se sostengan. De forma análoga, ventanas del tamaño dado por los límites teóricos encontrados son impracticables. En [30] se propone una técnica con múltiples ventanas para evitar los inconvenientes mencionados anteriormente derivados del uso de una única ventana.

Junto con los métodos y heurísticas empleados para determinar qué ejemplos retener en cada momento, el segundo distintivo entre los algoritmos incrementales viene dado por la lógica que gobierna la actualización y revisión del modelo, diferenciándose tres casos en la mayoría de las técnicas:

- **Éxito:** un nuevo ejemplo es cubierto por una de las reglas generadas para su misma etiqueta. En general existen dos alternativas a este caso: se actualizan las estadísticas sobre los datos de la regla que cubre al ejemplo o, en base a una heurística, se busca otra regla que tras describirlo pase a tener mayor precisión.
- **Novedad:** un nuevo ejemplo no es cubierto por ninguna de las reglas del modelo. En este caso lo habitual es buscar, en base a una heurística, la mejor regla a generalizar para describir al ejemplo.
- **Anomalía:** una o varias reglas del modelo son inconsistentes con un nuevo ejemplo (es cubierto por reglas asociadas a una etiqueta distinta). En este último caso el modelo debe revisarse de forma que las reglas vuelvan a ser consistentes y el nuevo ejemplo sea contemplado en el modelo (descrito por una nueva regla o una regla anterior que es generalizada).

Paralelamente, ya que dentro del aprendizaje incremental el entrenamiento y el test se realizan simultáneamente, la clasificación está sujeta a dos tipos de errores:

- **Falsas Alarmas:** comportamientos normales que son incorrectamente identificados como hostiles.
- **Falsas Aceptaciones:** comportamientos hostiles que son incorrectamente identificados como normales.

## 6 Aprendizaje en un entorno data streams

En teoría, un *data stream* es una secuencia de items  $\langle \dots e_i \dots \rangle$  procesados en orden creciente de llegada con respecto a sus índices  $i$ . En la práctica, procedentes de una gran diversidad de fuentes, los datos son recibidos a gran velocidad, presentan una elevada susceptibilidad al ruido debido al tráfico permanente entre dichas fuentes y poseen una fuerte dependencia del entorno, estando por tanto sujetos al cambio.

En el marco del aprendizaje supervisado, el modelado y la clasificación de *data streams* puede considerarse subárea del aprendizaje incremental donde la velocidad de llegada de los ejemplos y la dependencia del contexto exigen una lógica algorítmica compleja y altamente eficiente para tomar decisiones críticas en tiempo real en cuanto a la adaptación al cambio, el filtrado del ruido, la resistencia a cambios virtuales, la captura de patrones recurrentes y la detección de las últimas tendencias o fenómenos de comportamiento transitorios. Hasta la fecha, las estrategias empleadas por los principales sistemas desarrollados para hacer frente a todas estas decisiones de acuerdo con la velocidad de recepción de los ejemplos, pueden dividirse en cinco categorías [33]:

- **Reducción del número de datos de entrada a procesar:** mediante muestreo, filtrado, agregación y balance de carga. A diferencia del muestreo -basado en medidas estadísticas- el filtrado es un muestreo semántico donde los ejemplos son seleccionados en base a las medidas de interés definidas por el usuario. Las técnicas de agregación utilizan medidas tanto estadísticas como de interés para reducir el número de ejemplos y el número de atributos creando un conjunto de datos descrito a mayor nivel conceptual. Por último, los esquemas basados en balance de carga han sido propuestos en su mayoría para consultas de *streams* más que como técnicas de minería de datos [34,35,36,37]. En esencia, consisten en descartar directamente distintos grupos de ejemplos consecutivos.
- **Análisis a bordo.** Para reducir el tráfico de datos entre las distintas fuentes, las técnicas basadas en esta estrategia pueden englobarse dentro de la minería de datos distribuida, ya que reparten el proceso de aprendizaje entre las distintas fuentes en vez de centralizarlo en una única máquina. Sistemas basados en esta arquitectura son por ejemplo VEDAS [38], EVE [39] y Diamond Eye [40]. El principal inconveniente de este enfoque es la asunción de suficientes recursos en cada uno de los centros de adquisición y generación de datos.
- **Aumento del nivel conceptual del modelo de salida:** clasificando los ejemplos de entrada en un número fijo y reducido de categorías definidas según medidas de interés dadas por el usuario y reemplazando cada ejemplo por la etiqueta correspondiente - de forma similar a un proceso de discretización de valores continuos.
- **Adaptación del nivel de detalle del modelo de salida:** mediante un parámetro de control como parte de la lógica algorítmica del proceso de aprendizaje que determina la frecuencia de respuesta y la creación de reglas

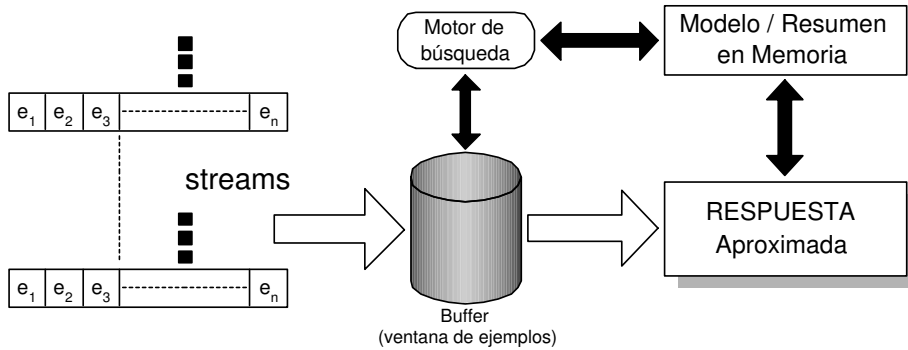
en función de 1) la velocidad de recepción de ejemplos, 2) la cantidad de memoria disponible y 3) el tiempo restante para consumir dicha memoria antes de actualizar de forma incremental el modelo de salida. La cantidad de patrones a generar, la memoria necesaria para los mismos y el tiempo requerido para cada nueva ampliación y adaptación del modelo se fija a partir de varios parámetros de usuario que reflejan las medidas de interés.

- **Algoritmos Aproximados.** Hasta la fecha es la estrategia más abordada en la literatura para el diseño de sistemas de decisión [41,42,43,44,45,46]. Las técnicas basadas en este esquema suponen las de mayor aportación y complejidad algorítmica ya que, independientes de los recursos disponibles, garantizan, antes del proceso de aprendizaje:
  - Un límite teórico para el tiempo máximo de procesamiento de cada ejemplo. Dicho límite conlleva, en la práctica, a procesar cada ejemplo, a lo sumo, una vez. Se denominan también algoritmos de una única pasada (*one pass mining algorithms*). Una vez es utilizado un ejemplo para decidir cambios en el modelo, éste es descartado o almacenado en disco, de forma que no puede ser recuperado fácilmente - en tiempo aceptable.
  - Puesto que el tamaño del *stream* es ilimitado, potencialmente infinito, exigen determinar de antemano una cantidad de memoria fija e independiente del número total de ejemplos a recibir, tanto para el modelo como para los ejemplos relevantes a retener en cada episodio de aprendizaje - el tamaño de la ventana.
  - Una cota de error para el modelo de salida en función del tiempo empleado para procesar cada ejemplo y del número de nuevos ejemplos que son necesarios almacenar en memoria para realizar cada actualización incremental del modelo. Ambos valores suelen ser fijados generalmente mediante parámetros de usuario.
  - Un modelo de salida asintóticamente idéntico, a medida que aumenta el número de ejemplos, a la obtenida por un algoritmo *por lotes* sin la presencia de *concept drift* en los datos de entrada, donde no tiene efecto el orden de llegada de los mismos.

La figura 3 ilustra la arquitectura y el enfoque seguido por los algoritmos de una única pasada, donde el objetivo no es proporcionar un modelo de gran exactitud sino garantizar una cota inferior de la misma en función de las exigencias del usuario.

La importancia de esta nueva filosofía, ausente en las técnicas de decisión por lotes en las que durante décadas el único objetivo ha sido conseguir, cueste lo que cueste, la mayor precisión posible, se refleja ya hoy día en muchos sistemas KDD capaces de gestionar flujos de datos continuos - mencionados al principio del presente capítulo - proporcionando:

- Técnicas de procesamiento de consultas aproximadas para evaluar consultas que requieran memoria ilimitada.
- Técnicas de procesamiento de consultas basadas en ventana deslizante, tanto como técnica de aproximación como opción al lenguaje de consultas.



**Figura 3.** Modelo de computación para *data streams*.

- Técnicas de muestreo para la gestión de situaciones donde la velocidad de llegada de los ejemplos es mayor que la velocidad de procesamiento de consultas.
- Definición e implementación de operadores blocking (p.e. agregación u ordenación) en presencia de *streams* interminables.

Nuestra propuesta se enmarca dentro de este último grupo de técnicas *aproximadas*.

## 7 Técnicas de Inducción Incremental de Reglas

### 7.1 AQ11

AQ11 [47] es el primer algoritmo de aprendizaje incremental basado en conjuntos de reglas descritas en FND. Supone una extensión del esquema de aprendizaje por lotes de AQVAL/1 [48] y a diferencia de éste, las reglas generadas en cada instante de tiempo  $t$  son inducidas a partir de las obtenidas en el instante anterior  $t - 1$  y de los nuevos ejemplos de entrenamiento recibidos en dicho instante  $t$ . AQ11 no retiene en memoria ningún ejemplo de entrenamiento y *confía* en el conjunto de reglas obtenido tras cada episodio o ciclo de aprendizaje. Por cada nuevo ejemplo cubierto por una regla de distinta etiqueta, AQ11 especializa tales reglas de forma iterativa hasta que todo ejemplo mal clasificado deja de ser cubierto. Posteriormente, siguiendo el esquema de aprendizaje de AQ, las reglas anteriormente especializadas son generalizadas a partir de los nuevos ejemplos pertenecientes a su misma etiqueta. De forma análoga, para generalizar un nuevo ejemplo no cubierto, AQ11 añade una regla específica que lo describe y a continuación, utiliza el esquema de inducción de AQ para obtener un nuevo conjunto de reglas consistentes que lo cubran.

Junto a la inflexibilidad del modelo, completo y consistente únicamente con los ejemplos recibidos en dicho instante  $t$ , el principal inconveniente de AQ11 es

su coste computacional debido a que el esquema inductivo de AQ es aplicado tantas veces como reglas sean especializadas en cada instante  $t$ .

## 7.2 GEM

GEM [49] (*Generalization of Examples by Machine*) supone una variante de AQ11 para mejorar la completitud y la consistencia del modelo cuando no existen restricciones de memoria, reteniendo todos los ejemplos de entrenamiento. El esquema de aprendizaje de GEM es muy similar al de AQ11, exceptuando que cada proceso de especialización y generalización tiene en cuenta todos los ejemplos recibidos independientemente de su momento de llegada. Mediante GEM, cuando un nuevo ejemplo es cubierto por una regla de distinta etiqueta, ésta es especializada según AQ11, utilizando para ello los nuevos ejemplos mal clasificados por dicha regla y los ejemplos de su misma etiqueta que, recibidos anteriormente, son actualmente cubiertos por otras reglas del modelo. Ahora bien, dicha especialización puede hacer que tal regla deje de cubrir a ejemplos que anteriormente cubría. Para contrarrestar la no completitud del modelo, GEM emplea nuevamente el esquema de inducción de AQ y generaliza dichas reglas de forma iterativa en base a todos los ejemplos recibidos de su misma etiqueta. De este modo, el modelo es completo y consistente con los nuevos ejemplos y con los recibidos en instantes pasados.

Al igual que en AQ, las reglas en GEM son descritas en FNC y son reparadas cada vez que un nuevo ejemplo es mal clasificado. Sin embargo, en vez de modificar una regla por completo cada vez que es inconsistente, GEM modifica sólo una pequeña parte de ella. Cuando una regla pierde la consistencia, las excepciones a la misma son descritas mediante términos conjuntivos individuales pertenecientes a su antecedente. Cada uno de estos términos *defectuosos* es sometido a una nuevo proceso de generalización en base a los ejemplos positivos que cubre y a los ejemplos negativos que lo provocaron. Claramente, GEM es impracticable en un entorno de data streams puesto que necesita almacenar todos los ejemplos recibidos.

Tras una comparación empírica frente a AQ en tres dominios distintos, los resultados obtenidos reflejaron que:

- en AQ11 y GEM la complejidad del modelo es mayor que en AQ.
- en AQ11 y GEM la exactitud es ligeramente inferior que en AQ.
- en AQ11 y GEM la actualización del modelo supone un coste computacional inferior que en AQ.

## 7.3 STAGGER

STAGGER [50] supone el primer algoritmo de aprendizaje incremental de reglas diseñado para ser robusto frente al ruido y poder modelar conceptos en entornos no estacionarios. Puesto que el ruido y los cambios en la función objetivo son imprevisibles, STAGGER no intenta mantener la consistencia de las reglas cada

vez que un nuevo ejemplo es mal clasificado. Por contra, intenta evitar permanentes y repentinas modificaciones que, sin tener certeza de su necesidad, pueden incrementar duramente el coste computacional.

Bajo tal planteamiento, STAGGER se aparta de las anteriores propuestas y mantiene una política conservadora, modificando las reglas únicamente cuando éstas poseen un alto grado de inconsistencia. STAGGER permite además descartar tales cambios a medida que nuevos ejemplos van siendo recibidos y convierten a las reglas modificadas en erróneas. Para ello, las reglas son almacenadas junto a un resumen estadístico sobre la importancia de sus términos (condiciones que constituyen a la regla). Al igual que GEM, los cambios se realizan a nivel de término (distintos componentes de una regla). Sin embargo, a diferencia de GEM y AQ11, STAGGER no utiliza ni almacena ejemplos pasados mal clasificados, sino que las estadísticas calculadas sirven para representar al conjunto de entrenamiento y guiar el proceso de *revisión*. En este sentido, ambas técnicas comparten el mismo supuesto y la misma estrategia de aprendizaje: la evidencia estadística y una política de actualización conservadora. Al revisar y realizar las modificaciones mediante primitivas *chunking* [51] (agrupamiento por categorías), no sólo se garantiza la robustez frente al ruido sino que además permite al sistema detectar cambios del entorno a largo plazo.

#### 7.4 FLORA

FLORA supone el primer sistema de aprendizaje puramente incremental capaz de adaptarse de forma automática tanto a cambios ocurridos en el entorno (*hidden context*) como a variaciones frecuentes y recurrentes en la función objetivo (*concept drift*). A partir del esquema de aprendizaje común, se han desarrollado posteriormente diferentes mejoras para una más rápida adaptación a cambios bruscos y una mayor robustez frente al ruido [52,26,53,24]. La familia de algoritmos FLORA obtiene un conjunto de reglas descritas en FND a partir de un esquema de ventana temporal basado en el tiempo de llegada.

Los ejemplos de entrenamiento son procesados secuencialmente y van ocupando posiciones consecutivas en la ventana, siendo descartados tras superar un tiempo umbral. Dicho umbral varía dinámicamente en función del rendimiento del sistema, la exactitud del modelo y la complejidad del mismo (medida en el número de condiciones que constituyen cada regla). Cuando existen fluctuaciones en el rendimiento, el sistema reduce el tamaño de la ventana bajo la presunción de que ello es debido a un cambio en la función objetivo, intentando así descartar cuanto antes aquellos ejemplos que representan conceptos *obsoletos*. A medida que las reglas ganan consistencia y el modelo se estabiliza el tamaño de la ventana se incrementa. Para ello cada regla tiene asociada el porcentaje de ejemplos cubiertos con respecto al tamaño de la ventana.

El aprendizaje se lleva a cabo mediante la incorporación y actualización de reglas que describan a los nuevos ejemplos de la ventana, junto con la eliminación de aquéllas referentes a ejemplos para los que ya expiró el tiempo de ventana y fueron excluidos de la misma. Basado en principios de la teoría de *Rough Sets*

(*FLOating Rough Approximation*), dicho proceso hace uso de dos conjuntos de ejemplos por etiqueta  $\mathcal{C}_k$ :

- *ADES* (*A*ccepted *D*escriptors), el conjunto de todas las reglas consistentes con  $\mathcal{C}_k$  (descripción por defecto de todos los ejemplos positivos con respecto a  $\mathcal{C}_k$ ).
- *PDES* (*P*otential *D*escriptors), el conjunto completo de todas las reglas candidatas, no necesariamente consistentes, para describir a los ejemplos asociados a  $\mathcal{C}_k$  (descripción por exceso).

Cada vez que un nuevo ejemplo es recibido se intenta clasificar con las reglas de todos los conjuntos *ADES*. Si el ejemplo no es cubierto por ninguna regla, se intenta generalizar alguna de las pertenecientes al conjunto *ADES* asociado a la etiqueta de dicho ejemplo. Si esto no es posible, entonces se añade una nueva regla para que lo describa. Si el nuevo ejemplo resulta ser cubierto por alguna regla perteneciente a un conjunto *ADES* de distinta etiqueta asociada, dicha regla pasa a formar parte del correspondiente conjunto *PDES*.

A medida que el número de ejemplos procesados crece, *ADES* contendrá las reglas más generales que consistentes con los ejemplos positivos, mientras que *PDES* almacenará reglas más específicas y ligeramente inconsistentes, candidatas a volver a ser válidas en el futuro. De esta forma, el conjunto *ADES* es utilizado para clasificar cada nuevo ejemplo recibido. Si un ejemplo pertenece a una regularidad, entonces su frecuencia relativa (con respecto al tamaño del conjunto de entrenamiento) será elevada y aparecerá constantemente en la ventana, por lo que si fue descartado, pasará de estar en *PDES* a estar nuevamente en *ADES*.

## 7.5 AQ-PM

Cuando el tiempo no interviene en la variabilidad de la función objetivo, en aquellos sistemas como AQ11 donde ningún ejemplo es retenido en memoria la exactitud del modelo resultante se ve altamente influenciada por el orden de llegada de los mismos. AQ-PM [16,4] (*P*artial *M*emory) supone la adaptación a dominios de tales características manteniendo del esquema de aprendizaje AQ. El planteamiento propuesto consiste en retener únicamente aquellos ejemplos que definen los límites descriptivos de las reglas que van siendo generadas. Dichos ejemplos, denominados extremos, pueden estar localizados en los vértices, en las aristas o en las superficies de cada uno de los hiperrectángulos paralelos a los ejes que representa cada regla. Existen pues tres variantes del algoritmo. Como segunda novedad, el esquema de inducción de reglas empleado por AQ-PM es una extensión del algoritmo por lotes de AQ15c. AQ15c es a su vez extensión de AQ por lo que AQ-PM no es puramente incremental. La principal diferencia entre ambos reside únicamente en la heurística de distancia utilizada para estimar la proximidad o el grado de pertenencia de un ejemplo a una regla. Por tanto, AQ-PM actúa como un *temporal-batch learner* puesto que reemplaza por completo las reglas obtenidas en un instante  $t - 1$  con aquellas inducidas a partir de los

nuevos ejemplos recibidos en el instante  $t$  y los ejemplos extremos retenidos, los cuales son actualizados tras cada proceso de inducción. AQ-PM permite además aplicar distintas políticas de ventana mediante parámetros de usuario basadas en tiempo, frecuencia o influencia.

## 7.6 SPLICE

SPLICE [25] es un sistema de meta-aprendizaje basado en *clustering* contextual para detectar cambios causados por variaciones en el entorno, de forma que cada concepto aprendido es inicialmente asociado a un contexto diferente. Asumiendo la consistencia en los datos de entrada, el sistema agrupa subsecuencias de ejemplos consecutivos en distintos intervalos, de forma que ejemplos pertenecientes al mismo intervalo describen al mismo concepto. En una segunda fase se lleva a cabo el *clustering* contextual, que consiste en unir intervalos similares en base a una medida de similitud basada en distancia. Finalmente, los *clusters* obtenidos son utilizados para inducir el modelo final de reglas consistentes con los datos de entrada.

## 7.7 AQ11-PM y GEM-PM

AQ11-PM y GEM-PM [21,22] son sencillas versiones de sus predecesores AQ11 Y GEM que emplean el método de selección de ejemplos extremos de AQ-PM, aprendiendo exclusivamente de ellos. Cuando se reciben nuevos ejemplos, AQ11-PM utiliza el esquema de aprendizaje de AQ11 para actualizar las reglas de forma que éstos sean cubiertos sin que el modelo pierda la consistencia. Si tras la actualización alguna de estas reglas fue generalizada, AQ11-PM modifica el conjunto de ejemplos de la ventana usando los nuevos ejemplos recibidos, de forma que todos ellos sean extremos, según el método de AQ-PM. A diferencia de FLORA, AQ11-PM permite almacenar ejemplos no consecutivos del stream de entrada en la ventana de entrenamiento, pudiendo descartarlos tras un periodo de tiempo. Análogamente, GEM-PM utiliza el esquema de aprendizaje de GEM, por lo que las nuevas reglas no son únicamente inducidas para ser consistentes con sólo los ejemplos actuales sino también se tienen en cuenta los recibidos en instantes anteriores. El principal inconveniente para aplicar AQ11-PM en un entorno de data streams es que el número de ejemplos retenidos en memoria crece exponencialmente con respecto al número de atributos. Dadas  $n$  reglas,  $m$  atributos y un tamaño medio de selector  $s$  (valores discretos asociados a cada una de las condiciones que forma el antecedente de una regla), en el peor de los casos AQ11-PM almacena  $\mathcal{O}(ns^m)$ . Cuando los ejemplos extremos pertenecen a los vértices de la regla el número de ellos será de orden  $\mathcal{O}(n2^m)$ . Si pertenecen a las aristas será  $\mathcal{O}(nm2^{m-1}s)$  y si pertenecen a las caras se tiene  $\mathcal{O}(nm2^{m-1}s^2)$ . En el mejor de los casos, cuando todos los ejemplos de recibidos corresponden a los vértices opuestos de las reglas, el número de ejemplos en memoria es de orden  $\mathcal{O}(n)$ .

## 8 Otras técnicas de Inducción Incremental

IB2 [54] es una sencilla aplicación incremental de la técnica del vecino más cercano para detectar las fronteras de decisión. Cuando un nuevo ejemplo recibido es clasificado correctamente mediante los ejemplos retenidos en memoria hasta ese momento, IB2 lo descarta; en caso contrario es retenido para ayudar a clasificar a nuevos ejemplos. IB2 es una técnica exclusivamente de clasificación y no permite el modelado de los datos de entrada, estando además limitada a entornos estacionarios donde no ocurren efectos severos en el orden de llegada de los ejemplos. AQ-PM se inspira en la lógica de IB2 para seleccionar los ejemplos extremos pero utiliza además para ello las reglas actuales del modelo. Otros sistemas que no retienen en memoria ningún ejemplo de entrenamiento son ARCH [55], *Winnow* [11,56] y *Weighted Majority* [57]. *MetaL(B)* y *MetaL(IB)* [58] son sistemas basados en *naïve-Bayes* y vecinos más cercanos respectivamente. Utilizan una ventana de tamaño fijo y al igual que FLORA3, modelan entornos con cambios recurrentes. *MetaL(IB)* utiliza además técnicas de ponderación y selección de ejemplos basadas en distancia para retener aquellos ejemplos más relevantes.

Las técnicas basadas en máquinas de soporte vectorial presentan una lógica similar a IB2 y al esquema de retención de ejemplos extremos de la familia AQ-PM. Dichas técnicas obtienen los ejemplos extremos al transformar el conjunto de entrenamiento original en un espacio de mayor dimensionalidad donde los ejemplos pertenecientes a etiquetas diferentes pueden ser linealmente separables y son separados mediante aquellos hiperplanos que dejan un margen mayor. Syed, Liu & Sung proponen en [59] un algoritmo incremental basado en máquinas de soporte vectorial y evalúan la capacidad del mismo para modelar dominios no estacionarios y detectar desplazamientos virtuales en la función objetivo. Los autores diferencian entre el desplazamiento real de la función objetivo y el percibido o virtual cuando se aplica un muestreo a los datos de entrada. Evalúan su propuesta con varios conjuntos del almacén UCI [60] aplicando validación cruzada de forma incremental.

Otras técnicas incrementales que no utilizan una política de ventana temporal, reteniendo ejemplos anteriores al instante actual son LAIR [61] y HILLARY [62]. Diseñadas para modelar únicamente conjuntos de dos etiquetas, LAIR retiene únicamente el primer ejemplo positivo recibido mientras que HILLARY retiene únicamente ejemplos negativos.

ITI [19] (versión incremental de C4.5), DARLING [63] y FAVORIT [64] son técnicas basadas en árboles de decisión. La idea común en ellas es asociar a cada hoja del árbol un subconjunto de ejemplos que son retenidos en memoria hasta que dicha hoja de ser válida. DARLING asigna un peso inicial a cada ejemplo recibido que va decreciendo conforme el número de vecinos de distinta etiqueta aumenta. Cuando el peso del ejemplo supera un umbral, es eliminado. Al contrario que AQ-PM, DARLING elimina de la memoria los ejemplos extremos y retiene únicamente los centrales, siempre y cuando la función objetivo es estacionaria y no existe influencia en el orden de llegada de los ejemplos. FAVORIT es un sistema que no retiene en memoria ningún ejemplo, asociando de forma similar un peso a cada nodo del árbol. A diferencia de DARLING, los nodos son

eliminados cuando no reciben ejemplos que aumenten su peso tras pasar un periodo de tiempo. Por el contrario, cuando un nodo recibe un gran número de ejemplos y su peso supera un umbral, dicho nodo permanece fijo en el árbol.

## 9 Sistemas de Decisión para data streams

A fecha de hoy, las principales técnicas incrementales basadas en reglas no han sido adaptadas para su aplicación en entornos *data streams*. Esto es debido principalmente a la complejidad computacional del algoritmo de búsqueda empleado y a que fueron diseñadas inicialmente para el aprendizaje de conceptos, por lo que han sido únicamente evaluadas en dominios con atributos simbólicos y discretos. Actualmente, las nuevas propuestas desarrolladas para la clasificación y el modelado de *data streams* etiquetados están basadas en árboles de decisión y métodos *ensembles* (de ensamblaje).

### 9.1 La familia VFDT

Entre todas las técnicas de decisión incrementales, la familia de algoritmos VFDT (*Very Fast Decision Tree*) parece estar llamada a convertirse, al igual que C4.5 dentro del aprendizaje por lotes, en el estándar para la clasificación de data streams mediante árboles [41,42,43,44,65].

Limitado inicialmente a *streams* de ejemplos con todos sus atributos simbólicos y provenientes de entornos estacionarios, VFDT [41] parte de la observación de Catlett [66] para sistemas *off-line* basados en árboles de decisión, según la cual, dado el esquema *top-down* seguido por estas técnicas para construir el modelo, no es necesario utilizar todos los ejemplos de entrenamiento para encontrar la mejor pareja (atributo,valor) como siguiente test-nodo del árbol, sino únicamente un subconjunto de aquellos no descritos previamente mediante cualquier predicado extraído de nodos anteriores.

Bajo la observación anterior, VFDT utiliza la inecuación de Hoeffding [67] como cota de error para determinar el número de nuevos ejemplos necesarios que garantiza continuar la expansión del árbol de forma incremental sin dañar la precisión del mismo.

**Teorema 91** *Dada  $r$  una variable aleatoria que se mueve en un rango  $R$  y la media  $\bar{r}$  de dicha variable para  $n$  observaciones independientes, la media de  $r$  es  $\bar{r} - \epsilon$  con una probabilidad de  $1 - \delta$ , de forma que:*

$$\epsilon = \sqrt{\frac{R^2 \ln \frac{1}{\delta}}{2n}} \quad (9.1)$$

A partir de la ecuación anterior, VFDT determina la relación entre la media  $\bar{r}$  de una variable aleatoria  $r$  y el criterio de división de un nodo en un árbol de decisión. Sea  $G$  la heurística a maximizar para elegir el mejor test en un nodo. Sean respectivamente  $\mathcal{A}_1$  y  $\mathcal{A}_2$  el mejor y el segundo mejor atributo según  $G$

para  $n$  ejemplos. La diferencia media de dicha heurística para todos los valores de ambos atributos, tras observar  $n$  ejemplos, es:

$$\Delta\bar{G} = \bar{G}(\mathcal{A}_1) - \bar{G}(\mathcal{A}_2) \geq 0$$

De lo anterior sigue que, observados  $n$  ejemplos y dado un valor de  $\delta$  deseado,  $\mathcal{A}_1$  es la mejor elección con una probabilidad de  $1 - \delta$ , de forma que:

$$\Delta\bar{G} > \epsilon^2$$

Por lo tanto, si  $\Delta\bar{G} > \epsilon$ , entonces  $\Delta(G) \geq \Delta\bar{G} - \epsilon$  con una probabilidad  $1 - \delta$  y  $\mathcal{A}_1$  es la mejor elección. Esto es, la probabilidad de que el criterio de Hoeffding y el de un algoritmo convencional elijan diferentes tests en cualquier nodo, decrece exponencialmente respecto al número de ejemplos de partida.

Junto al atractivo de la cota de error de Hoeffding, la gran ventaja de VFDT es que tanto el tiempo como la memoria necesaria para construir el modelo es lineal y no depende del número de ejemplos del stream sino del tamaño  $n$ . Cuanto menor sea  $n$ , menor será el coste computacional. Cuanto mayor sea el error permitido, menor será el tamaño del bloque. Paralelamente, ya que el error  $\epsilon$  es función monótona decreciente, para cada nodo sólo es necesario retener en memoria los nuevos ejemplos que van siendo leídos hasta obtener  $\epsilon < \Delta\bar{G}$ .

Si  $m$  es el número de atributos,  $v$  el máximo número de valores por atributo y  $k$  el número de etiquetas de clase, el algoritmo para generar un árbol de Hoeffding necesita memoria de orden  $\mathcal{O}(mvk)$  para almacenar las estadísticas necesarias para cada hoja, por lo que si  $h$  es el número de hojas, la memoria necesaria para todo el árbol es de orden  $\mathcal{O}(mvkh)$ . Luego la complejidad computacional es independiente del número de ejemplos que sean vistos.

A partir de las inecuaciones de Hoeffding, VFDT garantiza además que, con el número de ejemplos necesarios, el modelo de salida generado es asintóticamente idéntico al generado por un algoritmo de aprendizaje por lotes convencional. VFDT [41] es la primera propuesta, dirigida a conceptos estáticos y CVFDT [44] es una extensión del anterior que, mediante un bosque de árboles, modela dominios no estacionarios eliminando árboles obsoletos. Ambas propuestas están limitadas a atributos simbólicos y en [45] se propone una extensión para procesar atributos numéricos mediante un esquema de discretización sin pérdida de precisión. Gama et al. [42,43] han propuesto recientemente UFFT (*Ultra Fast Forest Trees*) y VFDTc, sistemas que extienden a VFDT en dos direcciones: capacidad para tratar con atributos numéricos directamente y la aplicación de naïve-Bayes en las hojas del árbol.

El principal inconveniente de estos sistemas es que pequeñas variaciones en la función objetivo pueden suponer complejas modificaciones en el árbol de decisión de salida, comprometiendo severamente el coste computacional y la eficiencia en el aprendizaje de *streams* de alta velocidad [46].

## 9.2 Sistemas basados en ensamblaje de modelos

Ampliamente utilizadas en el aprendizaje *off-line*, *Bagging* (*Bootstrap aggregating*) [68] y *Boosting* [69,70] son dos estrategias de propósito general para incre-

mentar la exactitud de los modelos inducidos por cualquier técnica de aprendizaje. Mediante heurísticas de votación y ponderación, ambas técnicas van creando modelos intermedios en base a los cuales formar un único modelo final cuya exactitud mejore la exactitud de cualquiera de ellos.

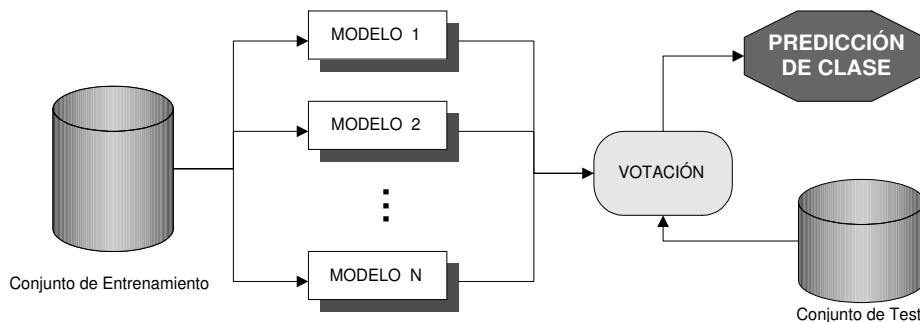
Mediante *Bagging* el modelo final es compuesto a partir de las reglas más frecuentes dentro de los modelos individuales. A partir de dos parámetros de usuario  $k$  y  $q$ , se realizan  $k$  muestras con reemplazo de tamaño  $q$  a partir del conjunto de entrenamiento original. Para cada muestra se aplica un clasificador distinto de forma que cada ejemplo de test es clasificado  $k$  veces, una vez para cada modelo. Debido al reemplazo en el muestreo puede que no todos los ejemplos del conjunto original sean seleccionados y que algunos aparezcan en varias muestras. Puesto que la probabilidad de que un ejemplo en un conjunto de tamaño  $n$  sea seleccionado es  $\frac{1}{n}$ , la probabilidad de que no lo sea es  $1 - \frac{1}{n}$ , y la probabilidad de que no lo sea tras  $k$  intentos será  $(1 - \frac{1}{n})^k$ , luego la probabilidad para todo ejemplo de ser seleccionado es  $1 - (1 - \frac{1}{n})^k$ . Cuando el tamaño del conjunto de entrenamiento ( $n$ ) tiende a infinito, la probabilidad anterior se acerca a  $1 - \frac{1}{e} = 63.2\%$ , lo que implica que cada muestra contenga sólo el 63.2% de ejemplos distintos.

Similar a *Bagging*, mediante *Boosting* se generan varios clasificadores y son votados de acuerdo a su tasa de error. Sin embargo, a diferencia de *Bagging*, no se obtienen a partir de diferentes muestras sino secuencialmente sobre el mismo conjunto de entrenamiento. Cada ejemplo del conjunto de entrenamiento tiene inicialmente asignado un peso 1 y secuencialmente cada clasificador modifica el peso de aquellos que son clasificados incorrectamente con un factor inversamente proporcional al error global del conjunto de entrenamiento ( $1/(2\varepsilon_i)$ ). Así, conjuntos de entrenamiento con un error reducido (igual o inferior al 0.1%) provocarán que los pesos de los ejemplos mal clasificados aumenten en varios órdenes de magnitud.

El objetivo perseguido es que tras el aprendizaje de un clasificador  $\mathcal{M}_i$ , el siguiente  $\mathcal{M}_{i+1}$  preste más atención a los errores de clasificación cometidos por todos sus anteriores. A su vez,  $\mathcal{M}_i$  obtiene un número de votos en función del número de aciertos obtenidos. Al final, el último clasificador combina los modelos de cada uno de los clasificadores anteriores en función del peso (número de votos) de cada uno. Una variante significativa de *Boosting* es *Adaboost* (***Adaptive boosting***) [71], diseñado para clasificadores poco robustos para los cuales una pequeña variación en los datos de entrada provoca un fuerte cambio en el modelo. Aun así, empíricamente se ha demostrado que para clasificadores con una tasa de error cercana al 50% *Adaboost* no es recomendable.

Como característica común, ambos métodos eliminan ejemplos con pesos pequeños para evitar errores de *underfitting* (subajuste o modelado pobre e insuficiente). Las diferencias más significativas entre ambos métodos son tres:

- En general, *Adaboost* es mejor que *Bagging* pero no siempre uniformemente mejor que cada clasificador, mientras que *Bagging* sí.
- En *Boosting*, si un clasificador tiene error cero, recibe recompensa infinita y es el único ganador, mientras que *Bagging* no.



**Figura 4.** Incremento de la exactitud mediante ensamblaje de modelos.

- *Bagging* sin poda reduce el error, mientras que *Adaboost* lo aumenta.

Con un gran número de variantes, en general todos los métodos de *ensamblaje* basados en *Bagging* y *Boosting* presentan las siguientes ventajas e inconvenientes con respecto a los sistemas de decisión autónomos:

#### **Ventajas:**

- Los ejemplos pueden no proporcionar la información suficiente para determinar el clasificador más idóneo al problema.
- El conjunto de datos puede no contener a la función objetivo.
- El algoritmo seleccionado puede no ser adecuado al problema, pero puede verse reforzado al modelar el conjunto.

#### **Inconvenientes:**

- Un tratamiento inadecuado del ruido provocaría una ponderación exagerada de los ejemplos incorrectamente clasificados.
- Incremento en el coste computacional y los requisitos de memoria, haciendo necesario la aplicación de métodos de eliminación de redundancias y paralelización de procesos.
- La comprensibilidad del modelo paga un alto precio por el incremento en la exactitud.

Contrariamente a lo que podría parecer dada la complejidad computacional de los mismos, los métodos de ensamblaje han recibido en los últimos cinco años una gran atención para el modelado y la clasificación de *data streams*. En general, y aunque dentro de un dominio incremental, las nuevas propuestas siguen el mismo esquema seguido por las técnicas de ensamblaje aplicadas en el aprendizaje por lotes, basándose en heurísticas y medidas de interés para eliminar, reactivar o añadir dinámicamente nuevos algoritmos de aprendizaje en respuesta a las variaciones ocurridas en la consistencia del modelo con respecto a los nuevos ejemplos recibidos.

Siendo muy reducido el número de propuestas aparecidas en la literatura, entre las principales cabe destacar como pionera SEA [72], la cual utiliza una cantidad de memoria fija e independiente del número de ejemplos y garantiza una cota máxima para el tiempo necesario por cada ejemplo. SEA utiliza una ventana de tamaño fijo donde todos los ejemplos son consecutivos y reemplazados en bloque (*chunks*). Bajo el mismo esquema de ventana, Wang et al. [46] proponen un método basado en ponderación de clasificadores en función de la precisión obtenida por los mismos al utilizar como test los propios ejemplos de entrenamiento. Kolter & Maloof [73] proponen DWM, basado en el algoritmo de ponderación por mayoría de Littlestone [57] para ensamblar modelos de distinta edad.

A diferencia de los métodos basados en árboles de decisión, los métodos de ensamblaje basados en ponderación de ejemplos evitan revisiones innecesarias en beneficio de la eficiencia en el proceso de aprendizaje. Aún así, obtienen un rendimiento considerablemente inferior al obtenido por las técnicas basadas en ventana deslizante [20]. Esto es debido a que las segundas construyen un modelo de reglas mientras que las primeras están principalmente dirigidas a la unión de árboles de decisión, los cuales poseen una sensibilidad mucho mayor al sobreajuste.

Paralelamente, el coste computacional de las técnicas de ensamblaje está expuesto a un alto riesgo frente a cambios virtuales en el *stream*, llevándolos a ponderaciones, ampliaciones, reducciones y reactivaciones innecesarias en los sistemas miembros. Por contra, las técnicas de decisión basadas en conjuntos de reglas se benefician de que el modelo no está estructurado jerárquicamente, consiguiendo así que las descripciones puedan ser modificadas o eliminadas cuando dejan de tener validez sin dañar el coste computacional.

Por otro lado, pese a aumentar la exactitud del modelo final, el ensamblaje presenta como gran inconveniente la nula interpretabilidad del modelo generado, el cual es generalmente presentado al usuario como una caja negra. Esto es debido a que la lógica interna que cohesionaba a los subárboles—partes difícilmente puede ser descrita o interpretada. Para posibilitar la comprensión de modelos tan complejos y poder aplicar el conocimiento obtenido en la toma de decisiones, las técnicas de visualización juegan un papel imprescindible.

Tanto evitar realizar revisiones innecesarias como facilitar la comprensión de modelos complejos han sido objetivos perseguidos en el desarrollo de nuestra herramienta FACIL, la cual se describe a continuación.

## 10 FACIL

FACIL es una técnica híbrida que intenta aprovechar las ventajas de varios enfoques de aprendizaje descritos en capítulos anteriores. Las razones que han motivado un diseño tan ecléctico y que se describen a continuación atienden a dos aspectos muy distintos:

- En cuanto al modelo de conocimiento: integración de reglas y vecinos.

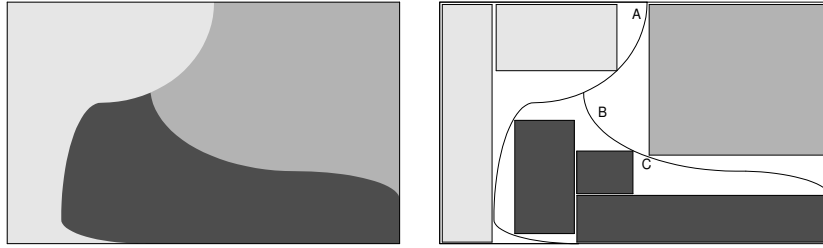
- En cuanto el esquema de aprendizaje: múltiples ventanas deslizantes e independientes.

### 10.1 Integración de reglas de decisión y vecinos más cercanos

Como principal ventaja frente a los árboles de decisión, los algoritmos de inducción de reglas no modelan todo el espacio sino únicamente aquellas regiones para las que existe una fuerte evidencia, pudiendo además ordenar las reglas obtenidas y presentar al usuario un modelo estructurado en función de ésta y según las medidas de interés. Indirectamente, este esquema de cubrimiento progresivo no padece la alta sensibilidad al sobre-ajuste presente en el esquema de modelado recursivo por particiones de los sistemas basados en árboles, el cual añade dos severos inconvenientes para modelar *data streams*.

- Por un lado, para decidir cuál es la mejor partición - el par *atributo-valor* a partir del cual continuar la ramificación del árbol - es necesario disponer en cada ciclo de aprendizaje de un número de ejemplos directamente proporcional a la exactitud deseada para el modelo. Si los datos presentan valores continuos y una elevada dimensionalidad, la cantidad de memoria requerida alcanzará cotas muy altas. Mediante técnicas de muestreo y heurísticas de evaluación de atributos, los sistemas existentes basados en ensamblaje de árboles (sección 9.1) intentan filtrar aquellos ejemplos cuyos valores describen con mayor independencia la nueva partición a modelar. El problema de esta segunda estrategia es que el árbol es construido por partes y ensamblado al final, comprometiendo el coste computacional del proceso. Si a esto añadimos el tiempo requerido para la poda, la actualización y la presentación del conocimiento en tiempo real puede llegar a ser un objetivo muy difícil de cumplir.
- Por otro lado, ante nuevos cambios en la función objetivo, varias ramas y subárboles del árbol global habrán quedado *obsoletos*. Puesto que las fases de entrenamiento y test se llevan a cabo simultáneamente en el aprendizaje incremental, evaluar la validez de cierta rama exigirá retener en memoria un alto número de ejemplos, ya que, a priori, los últimos ejemplos no tienen por qué pertenecer a la partición que está cambiando y que describe erróneamente dicha rama. Es más, debido al coste computacional que requiere reconstruir el modelo, puede ser incluso necesario descartar por completo el subárbol y empezar desde cero una nueva ramificación.

Si bien el aprendizaje mediante reglas adolece con menor grado de los dos problemas anteriores, su principal inconveniente - presente también en los árboles - aparece en pos de la sencillez tanto del algoritmo a implementar como del modelo final a presentar al usuario. Para facilitar su comprensión, las reglas aprendidas son generalmente hiperrectángulos con aristas paralelas a los ejes (figura 5), lo que impide modelar con precisión regiones del espacio cuyas fronteras de decisión corresponden a hiperplanos oblicuos o superficies complejas, usual en bases de datos con atributos continuos. Para tales dominios, el vecino más cercano sigue



**Figura 5.** Modelado mediante reglas de decisión consistentes.

siendo una de las estrategias más sencillas y eficaces, proporcionando resultados muy satisfactorios [74,75]. Sin embargo, junto a la sensibilidad al parámetro  $k$  y la dependencia a la función distancia utilizada, el vecino más cercano presenta tres severos inconvenientes para ser aplicado directamente como método de clasificación. Estos son:

- Alto coste computacional.
- Alta sensibilidad al ruido y a la dimensionalidad.
- Alta sensibilidad al orden de llegada de los ejemplos.

Puesto que por cada nuevo ejemplo recibido sería necesario calcular su distancia a los  $n$  retenidos en memoria en cada momento ciclo de aprendizaje, determinar si el nuevo ejemplo debe ser retenido o no - pertenece o no a una frontera de decisión - llevaría un coste computacional de orden  $\mathcal{O}(mn^2)$ , siendo  $m$  el número de atributos. Normalmente, la descripción de fenómenos complejos a la que atienden los *data streams* exigen una dimensionalidad muy elevada, por lo que identificar las numerosas fronteras de decisión que separan las distintas etiquetas para poder realizar predicciones con precisión supondría retener en memoria un número de ejemplos más allá de lo posible. Dejando a un lado la eficacia, tanto con estrategias de ventana como con heurísticas de ponderación sería necesario restringir la búsqueda y/o reducir el tiempo de proceso por ejemplo mediante técnicas de aproximación [76] para poder clasificar *streams* de forma eficiente.

El segundo gran inconveniente es el ruido que presentan los *data streams* debido al tráfico entre las diversas fuentes. Al no poder ser preprocesados como lo son los ejemplos de las bases de datos residentes en disco, la detección de *outliers* y el filtrado del ruido queda en manos del algoritmo de aprendizaje. Puesto que la idea es retener aquellos ejemplos muy próximos a otros de distinta etiqueta, cuando el ruido aparece en el atributo de clase el número de ejemplos considerados erróneamente cercanos a las fronteras de decisión no sólo dañará la precisión en la clasificación sino que además elevará el coste computacional.

Por último, la naturaleza incremental del problema da lugar además a errores de subajuste, de forma que lo que a priori puede ser considerado una anomalía, tras recibir nuevos ejemplos puede revelarse como regularidad. Si la frecuencia de llegada de los ejemplos que describen cierta región del espacio es menor

que el tamaño de la ventana o la mayor *edad* permitida, dichos ejemplos serán descartados por *no esperar* o *no poder esperar lo suficiente*, no llegando nunca a modelar dicha región.

Si a los inconvenientes anteriores añadimos el problema de la dependencia del contexto, los cambios en la función objetivo se traducirán en variaciones continuas en las fronteras de decisión y, consecuentemente, la selección de puntos tanto cercanos a las mismas - puntos externos - como lejanos a ellas - internos - convierten al vecino más cercano, como método de clasificación por sí mismo, en una estrategia poco fiable y altamente ineficiente en este contexto.

Ahora bien, delegando la clasificación de ejemplos fronterizos al vecino más cercano y el modelado de regiones internas a las reglas de decisión, los problemas de ambos métodos son atenuados.

Hasta la fecha, el reducido número de propuestas aparecidas en la literatura que integran reglas de decisión y vecinos más cercanos se enmarcan dentro del aprendizaje por lotes, necesitando almacenar todos los ejemplos en memoria y no pudiendo ser aplicadas a problemas de carácter incremental ni en entornos *data streams*. Hasta donde sabemos, FACIL supone la primera propuesta incremental dentro de esta reducida familia dirigida específicamente a la clasificación de secuencias de alta velocidad con atributos numéricos. Induciendo el modelo mediante un esquema incremental de cubrimiento progresivo y clasificando a ejemplos no cubiertos o pertenecientes a regiones para las que no se tiene certeza mediante el vecino más cercano, las ventajas de este enfoque con respecto a los árboles de decisión son más que notables, destacando:

1. la reducción de la sensibilidad del modelo al sobreajuste, limitando éste a las regiones para las cuales existe una fuerte evidencia.
2. la reducción del coste computacional que conllevan revisiones innecesarias causadas por falsas alarmas o cambios virtuales, ya que gracias a que el modelo no exige una estructura jerárquica, distintas partes del mismo pueden ser modificadas sin involucrar al resto.
3. la reducción de la complejidad computacional en la simplificación del modelo, gracias a lo anterior y al hecho de no ser necesarias fases de poda intermedias.
4. la ganancia en la simplicidad y sencillez del modelo, pudiendo además ser presentado en función de las medidas de interés definidas por el usuario y de la certeza sobre cada región modelada.
5. la ganancia en la precisión obtenida al clasificar ejemplos cercanos a las fronteras de decisión.
6. la reducción en la memoria necesaria para los ejemplos a utilizar en cada episodio de aprendizaje.

## 10.2 Múltiples ventanas deslizantes e independientes

Ajenas a las exigencias impuestas por la velocidad a la que los datos deben ser procesados en entornos *data streams*, las técnicas incrementales basadas en reglas atienden a un esquema común para maximizar tres aspectos críticos (sección 2):

**Tabla 1.** FACIL frente a las principales técnicas relacionadas.

Técnica	NN	Reglas	Incremental	C. Drift	Virt. Drift	Data Streams
IB2	•			○	○	
AQ		•				
CN2		•				
C4.5Rules		○				
RISE	•	•		○		
SUNRISE	•	•				
RIONA	•	•				
AQ11		•		•		•
GEM		○		•		○
STAGGER		•		•		•
FLORA		•		•		•
AQ-PM	○	•		•		•
GEM-PM	○	○		•		○
FACIL	•	•	•	•	•	•

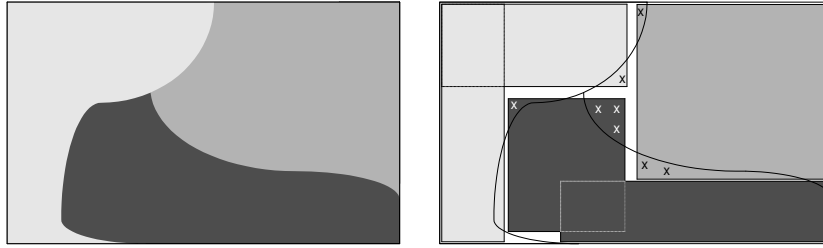
- La pendiente de la curva de aprendizaje, que determina la cantidad de ejemplos necesarios para poder realizar predicciones con un nivel de confianza satisfactorio.
- La independencia al orden de llegada de los ejemplos, que influye fuertemente sobre la curva de aprendizaje y el coste computacional de cada ciclo de aprendizaje.
- La adaptación al cambio en la función objetivo y en la distribución de los datos de entrada.

Con objeto de evitar tales problemas, la estrategia común consiste en aumentar o disminuir dinámicamente el tamaño de una única ventana conforme disminuye o aumenta respectivamente la consistencia del modelo con respecto a los nuevos ejemplos recibidos, por lo que el tamaño de la misma en cada ciclo de aprendizaje supone un factor crítico.

Al igual que las técnicas basadas en reglas relacionadas, el esquema de aprendizaje de FACIL puede enmarcarse dentro del de ventana deslizante. Sin embargo, FACIL no utiliza una ventana común para todas las reglas del modelo sino una ventana distinta por cada una ellas. Además, el tamaño de éstas no es modificado constantemente cada vez que se produce una variación en la consistencia, con lo que se consigue reducir al mínimo la complejidad computacional.

Las principales ventajas de este esquema con respecto a las estrategias de única ventana son dos:

1. Permite que cada región del espacio sea modelada con un número y conjunto de ejemplos independientes del resto. Estos ejemplos serán a su vez calculados en función de los cambios que afectan exclusivamente a la región a la que pertenecen.



**Figura 6.** Modelado en FACIL mediante reglas inconsistentes.

2. Elimina la dependencia de la periodicidad global, puesto que no todas las regiones tienen por qué cambiar simultáneamente ni los ejemplos de cada una de ellas tienen por qué llegar con la misma frecuencia.

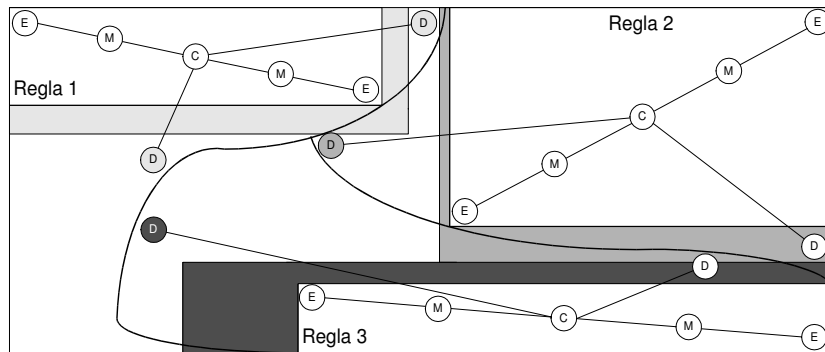
### 10.3 Reglas con ejemplos

La figura 6 muestra un ejemplo del conjunto de hiperrectángulos paralelos a los ejes que obtendría FACIL como modelo de conocimiento (imagen derecha) para una base de datos con dos atributos continuos y tres etiquetas de clase representadas mediante tres niveles de gris (imagen izquierda). A diferencia del esquema de particionado seguido por las técnicas basadas en árboles, el espacio de atributos no es necesariamente modelado en su totalidad, pudiendo existir:

- regiones para las cuales no hay ninguna regla asociada directamente (*huecos*).
- regiones para las cuales exista más de una regla asociada de igual etiqueta (*solapes*).
- regiones descritas por reglas asociadas a una etiqueta distinta a la cual pertenecen los ejemplos de las mismas.

Si bien la primera característica es común a las técnicas relacionadas, no todas ellas permiten la inconsistencia en las reglas obtenidas ni la intersección entre las mismas. Tal estrategia beneficia la sencillez del modelo pero daña su exactitud cuando las fronteras de decisión corresponden a superficies complejas o cuando aparecen regiones con elevada concentración y un alto grado de entropía. Esta pérdida es debida principalmente a dos motivos.

El primero es que numerosos ejemplos de entrenamiento localizados en regiones muy distantes entre sí no participarán en la construcción del modelo, dando lugar a un gran número de *huecos*. Mediante una lista de decisión o un conjunto de reglas estándar, la clasificación de ejemplos de test no cubiertos dará lugar a un gran número de errores, tanto con una única etiqueta o regla por defecto como con una métrica para la distancia entre un ejemplo y una regla. En la figura 5 se puede observar un claro ejemplo de este problema. Los puntos señalados con una A, B, y C representan ejemplos de test a clasificar pertenecientes a diferentes regiones (motivo de error en el uso de una regla por defecto) y más próximos a reglas de distinta etiqueta (motivo de error en el uso de una métrica).



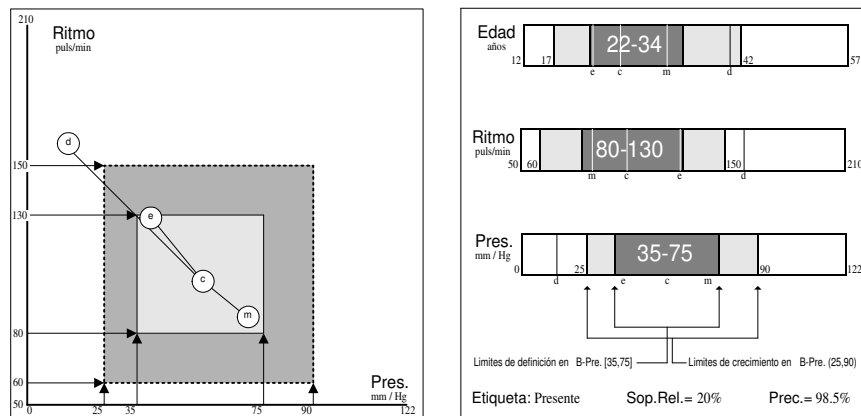
**Figura 7.** Reglas según SCARP v2.

El segundo motivo es debido al problema de las pequeñas disyunciones ya descrito anteriormente. Si durante la construcción del modelo van siendo eliminadas reglas de soporte reducido y éstas no son tenidas en cuenta al generalizar las reglas finales, aparecerá la inconsistencia en las mismas al modelar regiones para las que se procesaron ejemplos de distinta etiqueta.

Como solución a ambos problemas, en una primera aproximación utilizamos reglas consistentes formadas por seis elementos según ilustra la figura 7:

Mediante este modelo, la certeza de cada regla queda ampliada con información extra sobre las subregiones internas a la región a la que pertenecen y para las que existe probabilidad de inconsistencia, evitando así la pérdida de precisión en la fase de clasificación. La figura 7 muestra un ejemplo de las posibles reglas que compondrían un modelo intermedio a partir de un conjunto de entrada correspondiente al ejemplo dado en las figuras 5 y 6. Las zonas sombreadas indican los límites de crecimiento de cada regla en las que el punto central, los extremos, los delimitadores y las marcas son representados respectivamente como C, E, D y M. En fases tempranas del aprendizaje, las reglas no tendrían límites de crecimiento y podrían ser ampliadas en cualquier dirección - en ambos sentidos de toda dimensión. Conforme nuevos ejemplos van siendo procesados, el modelado de cada región va perfeccionándose, de forma que el punto central y los delimitadores de cada regla van desplazándose respectivamente hacia el centro de masa geométrico y hacia los vértices dados por los límites de definición, corrigiendo automáticamente la posición de las marcas. Paralelamente los límites de crecimiento van reduciéndose mientras los límites de definición van ajustándose a las fronteras de decisión.

En su versión final, FACIL induce reglas más simplificadas de forma que los ejemplos asociados son todos internos a las mismas y próximos a otros asociados a distintas etiquetas - ejemplos enemigos. Tanto las reglas como los ejemplos son actualizados dinámicamente a medida que van llegando otros nuevos, de forma que:



(a) Proyección en el plano sobre los atributos Presión y Ritmo.

(b) Extensión de los intervalos para los atributos Edad, Ritmo y Presión%.

**Figura 8.** Ampliación del valor semántico de una regla.

- las reglas son actualizadas en función de las demás, mientras que
- los ejemplos son actualizados en función de únicamente aquéllos asociados a la regla a la que pertenecen.

Con este nuevo modelo, estudios experimentales demuestran que no sólo se consigue una adaptación al cambio más rápida sino que además se aumenta la pendiente en la curva de aprendizaje. Con este último objetivo, para poder realizar predicciones tempranas las reglas pueden tener asociadas opcionalmente, junto a los ejemplos internos, un ejemplo externo de cada etiqueta distinta. Esta opción se lleva a cabo cuando se fija el tamaño máximo del modelo mediante un parámetro de usuario. Sin embargo, al contrario que en el modelo anterior, los nuevos extremos no intentan ser lo más lejanos posible a cada regla sino que corresponden a los últimos ejemplos recibidos más cercanos a las mismas. De esta forma se aproximan también los límites de la región hasta los cuales las reglas pueden ser ampliadas, pero ahora el *hueco* entre las regiones modeladas es mucho mayor.

Paralelamente, FACIL separa el modelo en dos conjuntos de modo similar a FLORA2: uno principal y otro secundario. El conjunto principal contiene las reglas actuales y el secundario aquellas que son potencialmente obsoletas. La construcción del modelo se lleva a cabo actualizando las reglas del conjunto principal, las cuales, cuando dejan de ser válidas, pasan al conjunto secundario y desaparecen de éste cuando existe una fuerte evidencia de su invalidez. Así mismo, reglas obsoletas - pertenecientes al conjunto secundario - pueden volver

a ser actuales si recuperan la consistencia, reduciendo la influencia en el modelo del orden de llegada de los ejemplos.

Además, cada regla tiene asociada cinco medidas estadísticas que son actualizadas conforme nuevos ejemplos son procesados y que permiten, a partir de las medidas de interés dadas por el usuario, corregir reglas erróneas y descartar reglas *obsoletas*. Estas medidas son:

- **Soporte Positivo** ( $e_p$ ): número de ejemplos cubiertos por una regla asociados a la misma etiqueta que la regla. Es utilizado para modelar únicamente aquellas regiones de mayor influencia.
- **Soporte Negativo** ( $e_n$ ): número de ejemplos cubiertos por una regla asociados a una etiqueta distinta a la de la regla. Es utilizado para eliminar patrones con poca influencia o pertenecientes a regiones complejas de elevada entropía (utilizando conjuntamente el soporte positivo).
- **Novedad** ( $i_n$ ): índice o posición en la secuencia de lectura del último ejemplo positivo cubierto por una regla. Es utilizada para detectar reglas que se han quedado *obsoletas* al cambiar la función objetivo.
- **Persistencia** ( $i_p$ ): frecuencia con la una regla es actualizada, bien por recibir nuevos ejemplos de su misma etiqueta, bien por ser generalizada para cubrirlos. Es utilizada para evitar el problema de la fragmentación y equilibrar el soporte en las reglas, dando prioridad a la hora de cubrir nuevos ejemplos a aquellas que son actualizadas con menor frecuencia.
- **Debilidad** ( $i_d$ ): número de veces que los límites de definición de una regla han sido reducidos. Es utilizada junto con la novedad para detectar reglas erróneas o que han dejado de estar *vigentes* al cambiar la función objetivo.

En resumen, el modelo de conocimiento que FACIL obtiene amplía la semántica y proporciona más información que la dada mediante las listas de decisión o los conjuntos de reglas estándar utilizados por todas las técnicas relacionadas.

## Referencias

1. KDD-95: International Conference on Knowledge Discovery in Databases, Montreal, Canada (1995)
2. Provost, F., Kolluri, V.: A survey of methods for scaling up inductive algorithms. *Data Mining and Knowledge Discovery* **3** (1999) 131–169
3. Giraud-Carrier, C.: A note on the utility of incremental learning. *AI Communications* **13** (2000) 215–223
4. Maloof, M., Michalski, R.: Selecting examples for partial memory learning. *Machine Learning* **41** (2000) 27–52
5. Vijayakumar, S., Schaal, S.: Fast and efficient incremental learning for high-dimensional movement systems. In: *Proc. of The Int. Conf. on Robotics and Automation - ICRA'00*. (2000) 1894–1899
6. Maybury, M., W. Wahlster, E., eds.: *Readings in intelligent user interfaces*. San Francisco, CA: Morgan Kaufmann (1998)
7. Srinivasan, K., Fisher, D.: Machine learning approaches to estimating software development effort. *IEEE Transactions on Software Engineering* **21** (1995) 126–137

8. Gupta, G.: Modeling customer dynamics using motion estimation in a value-based cluster space for large retail data-sets. Master's thesis, Department of Electrical and Computer Engineering, University of Texas, Austin (2000)
9. Cohen, W.: Learning rules that classify e-mail. In: Machine learning in information access: Papers from the 1996 AAAI Spring Symposium. Technical Report SS-96-05, Menlo Park, CA (1996) 18–25
10. Maloof, M.: An initial study of an adaptive hierarchical vision system. In: Proc. of The 17<sup>th</sup> Int. Conf. on Machine Learning – ICML'00. (2000) 567–573
11. Blum, A.: Empirical support for winnow and weighted-majority algorithms: Results on a calendar scheduling domain. *Machine Learning* **26** (1997) 5–23
12. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and issues in data stream systems. In Kolaitis, P.G., ed.: Proc. of The 21<sup>th</sup> Symposium on Principles of Database Systems – PODS'02, ACM Press (2002) 1–16
13. Golab, L., Ozsu, M.: Issues in data stream management. *SIGMOD Record* **32** (2003) 5–14
14. Muthukrishnan, S.: Data streams: algorithms and applications. In: Proc. of The 14<sup>th</sup> annual ACM-SIAM Symposium on Discrete Algorithms. (2003)
15. Muthukrishnan, S.: Seminar on processing massive data sets. Available Online: <http://athos.rutgers.edu/>
16. Maloof, M.: Progressive partial memory learning. PhD thesis, School of Information Technology and Engineering, George Mason University, Fairfax, VA (1996)
17. Utgoff, P.: ID5: An incremental ID3. In: Proc. of The 5<sup>th</sup> Int. Conf. on Machine Learning – ICML'88. (1988) 107–120
18. Utgoff, P.: Incremental induction of decision trees. *Machine Learning* **4** (1989) 161–186
19. Utgoff, P., Berkman, N., Clouse, J.: Decision tree induction based on efficient tree restructuring. *Machine Learning* **29** (1997) 5–44
20. Klinkenberg, R.: Learning drifting concepts: example selection vs. example weighting. *Intelligent Data Analysis, Special Issue on Incremental Learning Systems Capable of Dealing with Concept Drift* **8** (2004)
21. Maloof, M.: Incremental rule learning with partial instance memory for changing concepts. In: Proc. of The Int. Joint Conf. on Neural Networks, IEEE Press (2003) 2764–2769
22. Maloof, M., Michalski, R.: Incremental learning with partial instance memory. *Artificial Intelligence* **154** (2004) 95–126
23. Kelly, M., Hand, D., Adams, N.: The impact of changing populations on classifier performance. In: Proc. of The 5<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD'99, ACM Press (1999) 367–371
24. Widmer, G., Kubat, M.: Learning in the presence of concept drift and hidden contexts. *Machine Learning* **23** (1996) 69–101
25. Harries, M., Sammut, C., Horn, K.: Extracting hidden context. *Machine Learning* **32** (1998) 101–126
26. Widmer, G., Kubat, M.: Effective learning in dynamic environments by explicit context tracking. In: Proc. of The 6<sup>th</sup> European Conf. on Machine Learning – ECML'93, Springer-Verlag, LNCS 667 (1993) 227–243
27. Salganicoff, M.: Tolerating concept and sampling shift in lazy learning using prediction error context switching. *AI Review, Special Issue on Lazy Learning* **11** (1997) 133–155
28. Stanley, K.: Learning concept drift with a committee of decision trees. Technical Report UTAI-TR-03-302, Department of Computer Sciences, University of Texas at Austin, USA (2003)

29. Kubat, M., Widmer, G.: Adapting to drift in continuous domains. Technical Report ÖFAI-TR-94-27, Austrian Research Institute for Artificial Intelligence, Vienna (1994)
30. Lazarescu, M., Venkatesh, S., Bui, H.: Using multiple windows to track concept drift. *Intelligent Data Analysis Journal* **8** (2004) 29–59
31. Kuh, A., Petsche, T., Rivest, R.: Learning time-varying concepts. *Advances in Neural Information Processing Systems (NIPS)* **3** (1991) 183–189
32. Helmbold, D., Long, P.: Tracking drifting concepts by minimizing disagreements. *Machine Learning* **14** (1994) 27–45
33. Gaber, M., Krishnaswamy, S., Zaslavsky, A.: (Adaptive mining techniques for data streams using algorithm output granularity)
34. Babcock, B., Datar, M., Motwani, R.: Load shedding techniques for data stream systems (short paper). In: *Proc. of The 2003 Workshop on Management and Processing of Data Streams – MPDS’03*. (2003)
35. Tatbul, N., Cetintemel, U., Zdonik, S., Cherniack, M., Stonebraker, M.: Load shedding in a data stream manager. In: *Proc. of The 29<sup>th</sup> Int. Conf. on Very Large Data Bases – VLDB’03*. (2003)
36. Tatbul, N., Cetintemel, U., Zdonik, S., Cherniack, M., Stonebraker, M.: Load shedding on data streams. In: *Proc. of The 2003 Workshop on Management and Processing of Data Streams – MPDS’03*. (2003)
37. Viglas, S., Naughton, J.: Rate based query optimization for streaming information sources. In: *Proc. of SIGMOD*. (2002)
38. Kargupta, H.: **VEhicle DA**tA Stream mining (vedas) project. Available Online: <http://www.cs.umbc.edu/>
39. Tanner, S., Alshayeb, M., Criswell, E., Iyer, M., McDowell, A., McEniry, M., Renger, K.: EVE: On-board process planning and execution. In: *Earth Science Technology Conference*, Pasadena, CA (2002)
40. Burl, M., Fowlkes, C., Roden, J., Stechert, A., Mukhtar, S.: Diamond eye: A distributed architecture for image data mining. In: *SPIE DMKD*, Orlando (1999)
41. Domingos, P., Hulten, G.: Mining high-speed data streams. In: *Proc. of The 6<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’00*, ACM Press (2000) 71–80
42. Gama, J., Medas, P., Rocha, R.: Forest trees for on-line data. In: *ACM Symposium on Applied Computing – SAC’04*, ACM Press (2004) 632–636
43. Gama, J., Rocha, R., Medas, P.: Accurate decision trees for mining high-speed data streams. In Domingos, P., Faloutsos, C., eds.: *Proc. of The 9<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’03*, ACM Press (2003) 523–528
44. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: *Proc. of The 7<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’01*, ACM Press (2001) 97–106
45. Jin, R., Agrawal, G.: Efficient decision tree construction on streaming data. In Domingos, P., Faloutsos, C., eds.: *Proc. of The 9<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’03*, ACM Press (2003)
46. Wang, H., Fan, W., Yu, P., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In Domingos, P., Faloutsos, C., eds.: *Proc. of The 9<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’03*, ACM Press (2003) 226–235
47. Michalski, R., Larson, J.: Incremental generation of VL1 hypotheses: The underlying methodology and the description of the program AQ11. Technical Re-

- port UIUCDCS-F-83-905, Department of Computer Science, University of Illinois, Urbana-Champaign (1983)
48. Michalski, R.: Discovering classification rules using variable-valued logic system. In: Proc. of The 3<sup>th</sup> Int. Joint Conf. on Artificial Intelligence, Stanford, CA (1973) 162–172
  49. Reinke, R., Michalski, R.: Incremental learning of concept descriptions: A method and experimental results. *Machine Intelligence* **11** (1986) 263–288
  50. Schlimmer, J., Granger, R.: Beyond incremental processing: Tracking concept drift. In: Proc. of The 5<sup>th</sup> National Conf. on Artificial Intelligence, AAAI Press, Menlo Park, CA (1986) 502–507
  51. Schlimmer, J.: Incremental adjustment of representations for learning. In: Proc. of The 4<sup>th</sup> Int. Workshop on Machine Learning. (1987) 79–90
  52. Widmer, G., Kubat, M.: Learning flexible concepts from streams of examples: FLOORA2. In: Proc. of The 10<sup>th</sup> European Conf. on Artificial Intelligence – ECAI’92. (1992) 463–467
  53. Widmer, G., Kubat, M.: Combining robustness and flexibility in learning drifting concepts. In: Proc. of The 11<sup>th</sup> European Conf. on Artificial Intelligence – ECAI’94. (1994) 468–472
  54. Aha, D., Kibler, D., Albert, M.: Instance-based learning algorithms. *Machine Learning* **6** (1991) 37–66
  55. Winston, P.: Learning structural descriptions from examples. In: *Psychology of Computer Vision*. MIT Press, Cambridge, MA (1975)
  56. Littlestone, N.: Redundant noisy attributes, attribute errors, and linear-threshold learning using winnow. In: Proc. of The 4<sup>th</sup> Annual Workshop on Computational Learning Theory. (1991) 147–156
  57. Littlestone, N., Warmuth, M.: The weighted majority algorithm. *Information and Computation* **108** (1994) 212–261
  58. Widmer, G.: Tracking context changes through meta-learning. *Machine Learning* **23** (1997) 259–286
  59. Syed, N., Liu, H., Sung, K.: Handling concept drifts in incremental learning with support vector machines. In: Proc. of The 5<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’99, ACM Press (1999) 272–276
  60. Blake, C., Merz, E.: UCI repository of machine learning databases (1998)
  61. Elia, R., Watanabe, L.: An incremental deductive strategy for controlling constructive induction in learning from examples. *Machine Learning* **7** (1991) 7–44
  62. Iba, W., Woogulis, J., Langley, P.: Trading simplicity and coverage in incremental concept learning. In: Proc. of The 5<sup>th</sup> Int. Conf. on Machine Learning. (1988) 73–79
  63. Salganicoff, M.: Density-adaptive learning and forgetting. In: Proc. of The 10<sup>th</sup> Int. Conf. on Machine Learning. (1993) 276–283
  64. Kubat, M., Krizakova, I.: Forgetting and aging of knowledge in concept formation. *Appl. Artificial Intelligence* **6** (1992) 195–206
  65. Hulten, G., Spencer, L., Domingos, P.: Mining complex models from arbitrarily large databases in constant time. In: Proc. of The 8<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’02, ACM Press (2002)
  66. Cattlet, J.: Megainduction: machine learning on very large databases. PhD thesis, Basser Department of Computer Science, University of Sydney, Australia (1991)
  67. Hoeffding, W.: Probabilities inequalities for sums of bounded random variables. *Journal of American Statistical Association* **58** (1963) 13–30
  68. Breiman, L.: Bagging predictors. *Machine Learning* **24** (1996) 123–140

69. Freund, Y.: Boosting a weak learning algorithm by majority. In: Proc. of The 3<sup>th</sup> Annual Workshop on Computational Learning Theory. (1990) 202–216
70. Schapire, R.: The strength of weak learnability. *Machine Learning* **5** (1990) 197–227
71. Freund, Y., Schapire, R.: Experiments with a new boosting algorithm. In L. Saitta, e., ed.: *Machine Learning: Proc. 13<sup>th</sup> National Conference*, Morgan Kaufmann (1996) 148–156
72. Street, W., Kim, Y.: A streaming ensemble algorithm SEA for large-scale classification. In: Proc. of the 7<sup>th</sup> ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining – KDD’01, ACM Press (2001) 377–382
73. Kolter, J.Z., Maloof, M.: Dynamic weighted majority: A new ensemble method for tracking concept drift. In: Proc. of The 3<sup>th</sup> IEEE Int. Conf. on Data Mining ICDM-2003, IEEE CS Press (2003) 123–130
74. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is nearest neighbor meaningful? Technical report, CS Dept., University of Wisconsin-Madison, 1210 W. Dayton St. Madison, WI 53706 (1998)
75. Salzberg, S.L.: On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery* **1** (1997) 317–328
76. Micó, M., Oncina, J., Vidal, E.: A new version of the nearest-neighbour approximating and eliminating search algorithm (aesa) with linear preprocessing time and memory requirements. *Pattern Recognition Letters* **15** (1994) 9–17