Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

Aprendizaje automático para flujos de datos

Trabajo Fin de Máster

Máster Universitario en Inteligencia Artificial

AUTOR: Javier Ramos Fernández
TUTOR/ES: María Concepción Bielza Lozoya y
Pedro María Larrañaga Múgica

2019

Agradecimientos

Gracias a mis padres José Luis y Conchi por apoyarme día a día, por el cariño que me han dado, proporcionarme el sustento necesario, ser una fuente de inspiración y soportar mis frustraciones durante el desarrollo de este trabajo.

A mi hermano mayor Eduardo por ser siempre un ejemplo a seguir y por sus consejos de incalculable valor.

A mi hermano mellizo Pablo por ser un compañero leal, honesto, trabajador y hacer que el recorrido de mi vida sea un camino lleno de alegrías.

Al resto de mi familia por darme su apoyo incondicional durante la realización del máster.

A mis compañeros de clase por proporcionarme ayuda académica constantemente siempre que la he necesitado.

A mis tutores Pedro y Concha por sus valiosos consejos y por la comprensión que han tenido conmigo en todo momento para llevar a cabo este proyecto.

A mis amigos de toda la vida y de Madrid por ayudarme a evadirme de mis obligaciones y hacerme pasar muy buenos ratos.

A la gente que ha dedicado su tiempo a escuchar mis problemas y me ha animado a seguir adelante.

A todos gracias de corazón. Con paciencia y dedicación se puede conseguir todo lo que te propongas.

Javier

Resumen

En la actualidad existen numerosas aplicaciones que generan constantemente datos tales como transacciones financieras, consumo de electricidad, datos de monitorización de tráfico, registros telefónicos, búsquedas en Internet, información que se sube a las redes sociales, etcétera. Debido a la existencia de una gran variedad de aplicaciones que generan estos tipos de datos, en los últimos años ha surgido un gran interés por crear modelos que representen estos flujos continuos de datos. Éstos imponen una serie de restricciones a la hora de crear modelos de aprendizaje automático que representen la distribución subyacente a los mismos, de tal forma que las técnicas de aprendizaje automático convencionales no son adecuadas para llevar a cabo esta tarea. La principal característica de los flujos de datos es que el concepto que los describe puede evolucionar en el transcurso del tiempo, y las técnicas tradicionales construyen modelos a partir de conjuntos de datos estáticos, de manera que un modelo que se cree en un instante de tiempo puede que se quede obsoleto en un instante de tiempo posterior. De esta manera, es necesario adaptar dichas técnicas a la naturaleza dinámica de los flujos de datos o crear nuevas con el objetivo de tener un modelo consistente que permita representar de la mejor forma posible el concepto de los datos en cualquier instante de tiempo.

A partir de este hecho, en este trabajo el objetivo principal es realizar un estado del arte sobre las diferentes técnicas de aprendizaje automático propuestas para tratar con la modelización de flujos de datos. Concretamente, se van a abordar tanto algoritmos de aprendizaje supervisado como no supervisado, así como redes bayesianas para el descubrimiento de conocimiento. Con respecto a los algoritmos de aprendizaje supervisado, vamos a hacer hincapié en aquellos que gozan de mayor popularidad, que son los clasificadores bayesianos, los árboles de decisión, la inducción de reglas, las redes neuronales, los k-Vecinos más cercanos, las máquinas de vectores soporte, la regresión logística y la combinación de métodos de aprendizaje. En cuanto a los algoritmos de aprendizaje no supervisado, nos centraremos en aquellos que abordan un agrupamiento de los datos debido a su amplia utilización en la aplicación de aprendizaje automático sobre datos; concretamente, nos enfocaremos en abordar métodos de agrupamiento particional y jerárquico debido a la amplia gama de propuestas que tratan estos tipos de agrupamiento. Por último, atentiendo a las redes bayesianas para el descubrimiento del conocimiento, nos enfocaremos en aquellas utilizadas para manejar incertidumbre en entornos donde el estado de las variables evoluciona con el tiempo, que son principalmente las redes bayesianas dinámicas, las redes bayesianas en tiempo continuo y las redes bayesianas de nodos

temporales. Para cada uno de los tipos de algoritmos presentes en este trabajo se muestra una tabla comparativa de las diferentes propuestas abordadas.

Abstract

Today there are numerous applications that constantly generate data such as financial transactions, electricity consumption, traffic monitoring data, telephone records, Internet searches, information that is uploaded to social networks, and so on. Due to the existence of a great variety of applications that generate these types of data, in the last years a great interest has arisen to create models that represent these continuous data streams. These impose a number of constraints on creating machine learning models that represent the underlying distribution, so that conventional machine learning techniques are not suitable for carrying out this task. The main feature of data streams is that the concept that describes them can evolve over time, and traditional techniques construct models from static datasets, so that a model that is created at an instant of time may become obsolete at a later instant of time. In this way, it is necessary to adapt these techniques to the dynamic nature of the data streams or to create new ones with the objective of having a consistent model that allows the best possible representation of the concept of the data at any instant of time.

From this fact, the main objective of this work is to carry out a state of the art on the different machine learning techniques proposed to deal with the modelling of data streams. Specifically, both supervised and unsupervised learning algorithms will be addressed, as well as Bayesian networks for knowledge discovery. As regards supervised learning algorithms, we will focus on those that enjoy more popularity, which are Bayesian classifiers, decision trees, rule induction, neural networks, k-nearest Neighbors, support vector machines, logistic regression and ensemble methods. As unsupervised learning algorithms are concerned, we will deal with those that address data clustering due to their wide use in the application of machine learning on data; specifically, we will focus on addressing partitional and hierarchical clustering methods because of the wide range of proposals that address these types of clustering for data streams. Finally, looking at Bayesian networks for the knowledge discovery, we will describe those used to manage uncertainty in environments where the state of variables evolves over time, which are mainly dynamic Bayesian networks, continuous time Bayesian networks and temporal nodes Bayesian networks. For each type of algorithms present in this work we show a comparative table of the different proposals addressed.

Índice general

Índice de figuras

Índice de tablas

Índice de algoritmos

Capítulo 1

Introducción

1.1. Motivación

Actualmente vivimos en la era de la información, una época en la que se está generando una cantidad ingente de información. Esta abundancia de datos se debe principalmente a la aparición de ordenadores y de otros dispositivos como los teléfonos móviles que son capaces de recoger información de lo que nos rodea, procesarla y transmitirla. Además, su capacidad de conexión a Internet hace que haya una generación de tal cantidad de información que es imposible tener un control absoluto de todo lo que circula por esta red informática de nivel mundial.

Este aumento exponencial del volumen y variedad de información ha generado la necesidad de llevar a cabo un almacenamiento masivo de los datos, y con ello el interés por analizar, interpretar y extraer información útil de los mismos con el objetivo de obtener conocimiento. Para manejar toda esta información, los sistemas tradicionales de almacenamiento de datos no son convenientes puesto que no tienen la capacidad necesaria para su correcto procesamiento. El volumen, la variedad y la velocidad de los grandes datos causan inconvenientes de rendimiento cuando se utilizan técnicas tradicionales de procesamiento de datos (Mohanty et al. (2015)). Por ello surge lo que se denomina **Big Data**, un concepto relativamente nuevo que se refiere a conjuntos de datos cuyo tamaño va más allá de la capacidad de las herramientas típicas de software de bases de datos para almacenar, gestionar y analizar datos (Oguntimilehin y Ademola (2014)).

Los datos son la materia prima para conseguir información provechosa, que se puede utilizar para llevar a cabo una toma de decisiones y la realización de conclusiones, por lo que se han desarrollado nuevas herramientas que sobrepasan las disponibles con anterioridad para analizar grandes volúmenes de datos. De esta manera, surge el concepto de **minería de datos**, que se define como el proceso de extraer conocimiento útil y comprensible, previamente desconocido, a partir de grandes cantidades de datos almacenados en distintos formatos. Es decir, la tarea primordial de la minería de datos es encontrar modelos inteligibles a partir de los datos que posibiliten el hallazgo de aspectos previamente desconocidos de los mismos.

Para ejecutar el proceso de extracción del conocimiento, una de las posibilidades

más populares es la aplicación de una rama de la inteligencia artificial denominada **aprendizaje automático**, la ciencia (y el arte) de programar computadoras para que puedan aprender de los datos (Gron (2017)). En este caso, el objetivo es que los ordenadores aprendan automáticamente sin intervención humana. Este proceso de aprendizaje se realiza proporcionándoles a los algoritmos pertinentes una serie de datos sobre los que se entrenan con el objetivo de buscar patrones en los mismos y llevar a cabo mejores decisiones en el futuro.

En general, los algoritmos de aprendizaje automático asumen que los datos están disponibles a la hora de llevar a cabo el entrenamiento y que se generan a partir de una distribución estática. No obstante, la información presente hoy en día circula en un entorno que cambia de manera *continua* y *rápida*, de manera que la distribución que genera los datos puede sufrir transformaciones. Todo esto ha propiciado la aparición de lo que se denominan **flujos de datos** (*data streams*), que son secuencias continuas y ordenadas de datos en tiempo real.

A la hora de tratar con flujos de datos, los algoritmos de aprendizaje automático tradicionales no son capaces de funcionar correctamente puesto que el volumen de los mismos puede llegar a ser *infinito*. Para que puedan realizar el entrenamiento, el conjunto de datos debe estar almacenado en memoria, y solo pueden llevar a cabo tareas de predicción cuando la fase de entrenamiento haya finalizado. Sin embargo, el almacenamiento de datos generados continuamente se hace inviable. Además, para que los algoritmos puedan manejar flujos de datos cuya distribución subyacente puede cambiar es necesario un *procesamiento en tiempo real*, característica que los algoritmos tradicionales no poseen. Es decir, con la llegada de nuevos datos, en lugar de llevar a cabo un aprendizaje incremental con dichos datos sin utilizar de nuevo las instancias ya utilizadas para entrenar el modelo, los algoritmos tradicionales deben volver a realizar un entrenamiento del modelo de clasificación desde el principio teniendo en cuenta las nuevas instancias, lo que conlleva mucho tiempo de cómputo, algo impensable a la hora de tratar flujos de datos.

En este trabajo pretendemos realizar una revisión de la literatura sobre métodos propuestos para aplicar aprendizaje automático en flujos de datos con el objetivo de tener una visión global de las diferentes posibilidades existentes para resolver el problema planteado anteriormente.

1.2. Objetivos

El objetivo principal de este trabajo es realizar una **comparativa amplia de las diferentes aproximaciones propuestas en la literatura para resolver el problema de aprendizaje automático para flujos de datos, tanto clasificación supervisada como no supervisada**, así como la inclusión de propuestas relacionadas con redes bayesianas para el descubrimiento de conocimiento adaptadas a la temporalidad de los datos. Para abordar esta meta, nos centraremos en los algoritmos de aprendizaje automático más populares. En el caso del aprendizaje supervisado, nos enfocaremos en propuestas relacionadas con clasificadores bayesianos, árboles de decisión, inducción de reglas, redes neuronales, k-vecinos más cerca-

nos, máquinas de vectores soporte, regresión logística y combinación de métodos de aprendizaje (*ensemble*). Con respecto al aprendizaje no supervisado, consideraremos primordialmente los algoritmos de agrupamiento (*clustering*), uno de los principales de este tipo de aprendizaje; concretamente, nos enfocaremos en abordar métodos de agrupamiento particional y jerárquico debido a la amplia gama de propuestas que tratan estos tipos de agrupamiento. También nos centraremos en las redes bayesianas para el descubrimiento del conocimiento que tienen en cuenta la dimensión temporal. Otros algoritmos de aprendizaje automático que existen y que no se van a tratar en este trabajo son la regresión lineal, la asignación latente de Dirichlet y el análisis de componentes principales.

Para alcanzar este objetivo, se han contemplado una serie de metas específicas dentro del proyecto:

- *Búsqueda de artículos relacionados con cada uno de los algoritmos de aprendizaje automático contemplados.* Con el objetivo de llevar a cabo una comparativa de los diferentes métodos propuestos de aprendizaje automático para flujos de datos, es de vital importancia realizar una búsqueda de diferentes artículos propuestos para cada uno de los algoritmos de aprendizaje automático mencionados anteriormente. Para ello hemos consultado diferentes revistas que abordan esta rama de la inteligencia artificial, así como otros recursos como *Google Académico*.

- *Anotación de aspectos claves de los diferentes artículos encontrados.* Con ello se pretende tener disponible información resumida de las diferentes propuestas que se encuentran a nuestra disposición y utilizarla para compararlas con otros artículos con la finalidad de dar una perspectiva general de la utilidad de las diferentes aproximaciones existentes.

- *Exploración de las diferentes revisiones halladas sobre algoritmos de aprendizaje automático para flujos de datos.* El objetivo de esto es conocer qué propuestas de las que hemos encontrado se encuentran referenciadas en esas revisiones para tener en cuenta qué artículos son novedosos con respecto a dichas revisiones. Además, se lleva a cabo esta exploración para tener conocimiento de qué algoritmos de aprendizaje automático tratan, con el fin de aportar nuevos algoritmos. Todo esto se realiza con la finalidad de establecer qué características nos diferencian de las revisiones sobre el estado del arte encontradas.

- *Estructuración de la revisión de la literatura de algoritmos de aprendizaje automático para flujos de datos en función de los algoritmos en los que se centren las diferentes propuestas encontradas.* Se persigue comparar los diferentes artículos hallados según el algoritmo de aprendizaje automático que aborden. Para ello hemos realizado una división de los mismos en diferentes apartados para cada uno de los algoritmos de aprendizaje automático.

1.3. Metodología

Para desarrollar este proyecto se ha seguido una metodología que permitiera sobrellevar las dificultades del mismo de la mejor forma posible. A continuación se enumeran los pasos ejecutados durante este proceso:

- En primer lugar, hemos procedido a realizar una **búsqueda de artículos relacionados con aprendizaje automático para flujos de datos**. Antes de realizar una comparación entre las diferentes propuestas, es necesario recabar la mayor cantidad de aproximaciones desarrolladas con el fin de ampliar nuestra visión genérica del estado del arte del tema abordado. Para lograr esto, hemos indagado en numerosas revistas que incluyen dentro de su temática el aprendizaje automático para flujos de datos. Algunas que se han consultado y que son relevantes en el mundo académico son *Journal of Machine Learning Research*, *IEEE Transactions on Knowledge and Data Engineering* y *Machine Learning Journal*. También se han explorado diferentes editoriales como *Elsevier* y *Springer* en las que, aparte de artículos publicados en revistas, también aparecen publicaciones de conferencias. De forma complementaria, hemos buscado artículos en *Google Académico*, una herramienta de búsqueda de Google que permite hallar literatura del mundo científico de diferentes recursos (bibliotecas, editoriales, etcétera).

- Tras aglomerar una cantidad aceptable de artículos, hemos iniciado la **lectura de los mismos**. Durante este proceso, hemos ido apuntando características relevantes de las propuestas con el fin de poder tener la información fundamental para realizar la comparativa entre diferentes artículos. Para guardar dicha información, hemos creado un documento en el que, por cada propuesta, se apunta el *título*, la *fecha de publicación* y *contenido de interés* de las mismas, así como *comparaciones con otras aproximaciones* que se encontraban dentro de los artículos para utilizarlas en el estado el arte. Asimismo, hemos añadido información adicional con respecto a la *presencia o no de los artículos en las diferentes revisiones* encontradas que abordan el tema de aprendizaje automático para flujos de datos. Esto se ha llevado a cabo con el objetivo de tener conocimiento sobre qué artículos de los que hemos encontrado aportan nueva información con respecto a revisiones previas para establecer características que nos diferencian de las mismas. Estas propuestas se han ordenado por *fecha de publicación* para contemplar la evolución cronológica de las mismas.

- De forma paralela a la lectura de artículos, hemos procedido a **buscar más propuestas**, enfocándonos en encontrar aquellas que son más recientes para aportar más información que nos diferencie de las revisiones encontradas.

- Utilizando la información proveniente de la lectura de las diferentes propuestas, hemos llevado a cabo **el desarrollo del estado del arte de cada uno de los algoritmos de aprendizaje automático para flujos de datos**. La redacción del mismo se ha realizado de tal forma que el lector tenga una

idea general y clara del trabajo desarrollado en cada una de las propuestas abordadas.

1.4. Organización de la memoria

La disposición de la información que se va a seguir para abarcar todo lo relacionado con el desarrollo del proyecto es la siguiente:

- En el capítulo 2 se introduce la notación que se utiliza en este trabajo y se explican conceptos teóricos relacionados con cada uno de los algoritmos de aprendizaje automático que se van abordar en el proyecto.
- En el capítulo 3 se exponen una serie de conceptos que son frecuentes en la literatura relacionada con algoritmos de aprendizaje automático para flujos de datos y se abordan propuestas de los diferentes algoritmos de aprendizaje automático mencionados en el capítulo 2 para manejar flujos de datos. También se mencionan diferentes conjuntos de datos cuya utilización es frecuente para comprobar el desempeño de las distintas propuestas realizadas.
- En el capítulo 4 se exponen las conclusiones y las líneas futuras planteadas para seguir desarrollando el proyecto llevado a cabo.

Capítulo 2

Aprendizaje automático

Arthur Samuel, uno de los pioneros del aprendizaje automático, estableció en 1959 una definición general de esta rama de la inteligencia artificial:

> *El aprendizaje automático es el campo de estudio que da a las computadoras la capacidad de aprender sin estar programadas explícitamente.*

Tom Mitchell, otro investigador reputado del campo del aprendizaje automático, propuso en 1997 una definición más precisa y más orientado a la ingeniería:

> *Se dice que un programa de ordenador aprende de la experiencia E con respecto a alguna tarea T y alguna medida de rendimiento P, si su rendimiento en T, medido por P, mejora con la experiencia E.*

Por lo tanto, el aprendizaje automático se centra en aplicar sistemáticamente algoritmos para sintetizar de forma automática las relaciones subyacentes en un conjunto de datos proporcionados en forma de ejemplos a través de una fase de entrenamiento, de tal forma que en el futuro se utilice esta información para la ejecución de predicciones de eventos desconocidos y una mejor toma de decisiones. Los campos de aplicación que existen de esta rama de la inteligencia artificial son muy variados. Algunos de ellos son la predicción bursátil, predicción meteorológica, detección de correos spam, construcción de sistemas de recomendación y detección de fraude en el uso de tarjetas de crédito.

Según el propósito que persigan los algoritmos de aprendizaje automático, éstos se clasifican en dos categorías principales: **aprendizaje supervisado** y **aprendizaje no supervisado**. En el aprendizaje supervisado se engloban aquellos algoritmos que buscan aprender una *función de mapeo* entre una serie de características de entrada (variables predictoras) y una variable de salida (variable clase) mediante la ejecución de una fase de entrenamiento en la que se usan *datos de entrenamiento etiquetados* (valor de la variable clase conocida), de tal forma que se utiliza esta función para predecir el valor de la variable de salida a partir de los valores de las variables predictoras. En cambio, en el aprendizaje no supervisado la finalidad es aprender una *función que modele la estructura o distribución subyacente en los datos*

a partir de datos de entrenamiento no etiquetados, de manera que se lleva a cabo una exploración de los datos sólo conociendo los valores de las variables predictoras.

Existe además otra categoría en la que se pueden incluir las técnicas de aprendizaje automático denominada **aprendizaje semisupervisado**. Los algoritmos que llevan a cabo este tipo de aprendizaje entrenan sobre un conjunto de datos parcialmente etiquetados, generalmente muchos datos sin etiquetar y una pequeña parte de datos etiquetados. Debido a que para la fase de entrenamiento se utilizan tanto datos etiquetados como no etiquetados, se considera que esta categoría se sitúa entre el aprendizaje supervisado y el no supervisado. Concretamente, para entrenar un modelo de forma semisupervisada, se debe realizar una suposición sobre la estructura de la distribución subyacente a los datos, y una de las más frecuentes es el *cluster assumption*. Esta suposición establece que las instancias de cada una de las clases tienden a formar un *cluster*, y una clase puede estar constituida por varios *clusters*. De esta forma, se aplica un algoritmo de agrupamiento sobre los datos para encontrar los límites de cada uno de los *clusters* (obtener información sobre la estructura de los datos) y se usan las instancias etiquetadas para asignar una clase a cada *cluster*. En este trabajo se van a abordar algunas propuestas relacionadas con el aprendizaje semisupervisado para flujos de datos; no obstante, la contribución predominante de algoritmos de aprendizaje automático para flujos de datos que se va a realizar en esta tesis fin de máster proviene de las categorías de aprendizaje supervisado y no supervisado.

2.1. Notación

A continuación se expone la notación a utilizar en este trabajo:

Símbolo	Explicación
X_i	Variable predictora i
$\mathbf{X}$	Vector de variables predictoras
x_i	Valor de la variable predictora i
Ω_{X_i}	Dominio de valores de la variable i
Ω_C	Dominio de valores de la variable clase (finito)
x_{ij}	Valor j de la variable predictora i
$p(x_{ij})$	Probabilidad del valor j de una variable i
$\mathbf{x}$	Instancia de las variables predictoras
$x_i^{(j)}$	Valor de la variable predictora i de la instancia j
$\mathbf{x}^{(j)}$	Valores de las variables predictoras de la instancia j
C	Variable clase
c_j	Valor j de la variable clase
$c^{(j)}$	Valor de la variable clase en la instancia j
$(\mathbf{x}^{(j)}, c^{(j)})$	Estructura de una instancia j de clasificación supervisada
N	Número de instancias de un conjunto de datos
n	Número de variables predictoras
$\|\Omega_{X_i}\|$	Número de valores que puede tomar una variable predictora
$\|\Omega_C\|$	Número de valores que puede tomar la variable clase
D	Conjunto de casos
S	Flujo de datos
$*$	No es seguro (tablas comparativas)

Tabla 2.1: Notación utilizada en el trabajo

2.2. Algoritmos de aprendizaje supervisado

La mayor parte de las propuestas que se van a abordar en este trabajo relacionadas con el aprendizaje automático para flujos de datos se engloban dentro de la categoría de **aprendizaje supervisado**. Los algoritmos de clasificación supervisada son aquellos en los que, a partir de un conjunto de ejemplos etiquetados (conjunto de entrenamiento) $D = \{(\mathbf{x}^{(1)},\mathrm{c}^{(1)}), ..., (\mathbf{x}^{(N)}, c^{(N)})\}$, se intentan clasificar nuevas instancias. Formalmente, en el aprendizaje supervisado el objetivo es encontrar una función f que permita mapear una instancia $\mathbf{x} = (x_1, ..., x_n)$ a una determinada clase c:

$$\begin{array}{rcl} f : \Omega_{X_1} \times \ldots \times \Omega_{X_n} & \longrightarrow & \Omega_C \\ \mathbf{x} = (x_1, ..., x_m) & \mapsto & c \end{array} \tag{2.1}$$

A continuación se expone la estructura típica que presenta el conjunto de datos para un problema de aprendizaje supervisado:

	X_1	...	X_n	C
$(\mathbf{x}^{(1)}, c^{(1)})$	$x_1^{(1)}$	...	$x_n^{(1)}$	$c^{(1)}$
$(\mathbf{x}^{(2)}, c^{(2)})$	$x_1^{(2)}$	...	$x_n^{(2)}$	$c^{(2)}$
...		...		...
$(\mathbf{x}^{(N)}, c^{(N)})$	$x_1^{(N)}$	...	$x_n^{(N)}$	$c^{(N)}$
$\mathbf{x}^{(N+1)}$	$x_1^{(N+1)}$	...	$x_n^{(N+1)}$	???

Tabla 2.2: Problema de clasificación supervisada

Los algoritmos de aprendizaje automático que se van a tratar en este documento gozan de una gran popularidad. Concretamente, nos enfocaremos en **clasificadores bayesianos**, **árboles de decisión**, **inducción de reglas**, **redes neuronales**, **k-vecinos más cercanos**, **máquinas de vector soporte**, **regresión logística** y **combinación de métodos de aprendizaje**.

2.2.1. Clasificadores bayesianos

Un tipo de modelo que se utiliza ampliamente para la clasificación supervisada son las **redes bayesianas**. Las *redes bayesianas* son modelos gráficos probabilísticos que permiten representar de manera sencilla, compacta, precisa y comprensible la distribución de probabilidad conjunta de un conjunto de variables aleatorias. Este modelo gráfico está compuesto por *nodos*, que representan a las variables aleatorias; *arcos*, que representan las relaciones de dependencia entre nodos y *tablas de probabilidad condicional*, que representan la distribución de probabilidad condicional de cada uno de los nodos.

Formalmente, la estructura de una red bayesiana sobre un conjunto de variables aleatorias $X_1, ..., X_n, C$ es un grafo acíclico dirigido cuyos vértices corresponden a las variables, cuyos arcos codifican las dependencias e independencias probabilísticas entre tripletas de variables y, en cada uno de los vértices, se representa una distribución categórica local $p(x_i|\mathbf{pa}(x_i))$ o $p(c|\mathbf{pa}(c))$, donde $\mathbf{pa}(x_i)$ es un conjunto de valores para el conjunto de variables $\mathbf{Pa}(X_i)$, que son los padres de la variable X_i en el modelo gráfico. Lo mismo se aplica para $\mathbf{pa}(c)$ (Bielza y Larrañaga (2014)). Por lo tanto, la factorización que permite llevar a cabo la red bayesiana de la probabilidad conjunta de todas las variables aleatorias y que trata de evitar estimar un número exponencial de parámetros es la siguiente:

$$p(\mathbf{x}, c) = p(c|\mathbf{pa}(c)) \prod_{i=1}^{n} p(x_i|\mathbf{pa}(x_i)) \tag{2.2}$$

Las redes bayesianas, cuando se utilizan con propósitos de realizar tareas de clasificación, reciben el nombre de **clasificadores bayesianos**. En los clasificadores bayesianos, el objetivo es asignar la clase más probable a una instancia determinada, definida por un conjunto de valores de las variables predictoras. En términos

probabilísticos, se asigna a una instancia de prueba la etiqueta de clase con la *mayor probabilidad a posteriori* (*MAP*). Es decir:

$$\underset{c}{\operatorname{argmax}}\, p(c|\mathbf{x}) \tag{2.3}$$

Utilizando la regla de Bayes, podemos relacionar los términos de las ecuaciones (2.2) y (2.3) y además, puesto que el objetivo es calcular el valor de C con mayor probabilidad a posteriori, no es necesario tener en cuenta el denominador en la regla de Bayes (el factor de normalización). De esta manera, obtenemos la siguiente expresión (Bielza y Larrañaga (2014)):

$$\underset{c}{\operatorname{argmax}}\, p(c|\mathbf{x}) = \underset{c}{\operatorname{argmax}}\, p(\mathbf{x}, c) \tag{2.4}$$

Así, podemos utilizar la ecuación (2.4) para hallar la clase con la mayor probabilidad a posteriori. Esta ecuación establece el caso general de los clasificadores bayesianos, en el que $p(\mathbf{x}, c)$ se puede factorizar de diferentes maneras. Para llevar a cabo esta factorización tenemos que buscar lo que se denomina el **manto de Markov** (*Markov blanket*) de la variable C para encontrar la solución de la ecuación (2.4). El manto de Markov se define como el conjunto de variables MB_c que hacen que, dado dichas variables, la variable C sea condicionalmente independiente de las demás variables de la red bayesiana. El manto de Markov está formado, cogiendo a la variable C de referencia, por los **padres**, los **hijos** y los **padres de los hijos** de dicha variable. De esta forma (Bielza y Larrañaga (2014)):

$$p(c|\mathbf{x}) = p(c|\mathbf{x}_{MB_c}) \tag{2.5}$$

La Figura 2.1 muestra un ejemplo de la estructura del manto de Markov de una variable C, formado por las variables X_1 (hijo), X_2 (padre), X_3 (padre de un hijo) y X_4 (hijo).

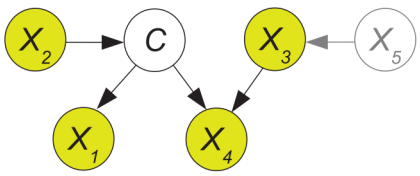

Figura 2.1: Estructura del manto de Markov. Fuente: Bielza y Larrañaga (2014)

Para el caso específico en el que la variable C no tenga padres y, utilizando la regla de la cadena, la probabilidad conjunta de las variables predictoras y de la variable clase se puede expresar de la siguiente manera (Bielza y Larrañaga (2014)):

$$p(\mathbf{x}, c) = p(c)p(\mathbf{x}|c) \tag{2.6}$$

de tal forma que el objetivo es maximizar en c.

Los distintos clasificadores bayesianos que existen se diferencian en la forma en la que factorizan $p(\mathbf{x}|c)$. Los clasificadores bayesianos más conocidos son **naive Bayes**, **tree agumented naive Bayes** (TAN), **k-dependence Bayesian classifier** (k-DB), **semi-naive Bayes** y **Bayesian multinet**. Las Figuras 2.2-2.6 muestran ejemplos de estructuras de cada uno de ellos.

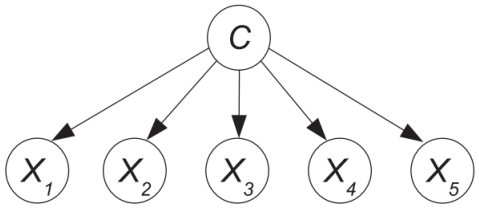

Figura 2.2: Naive Bayes. Fuente: Bielza y Larrañaga (2014)

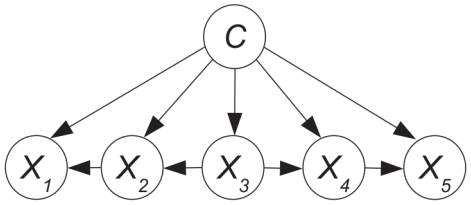

Figura 2.3: TAN. Fuente: Bielza y Larrañaga (2014)

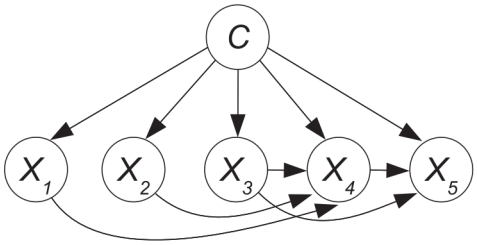

Figura 2.4: k-dependence Bayesian classifier. Fuente: Bielza y Larrañaga (2014)

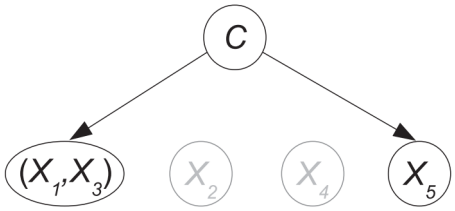

Figura 2.5: Semi-naive Bayes. Fuente: Bielza y Larrañaga (2014)

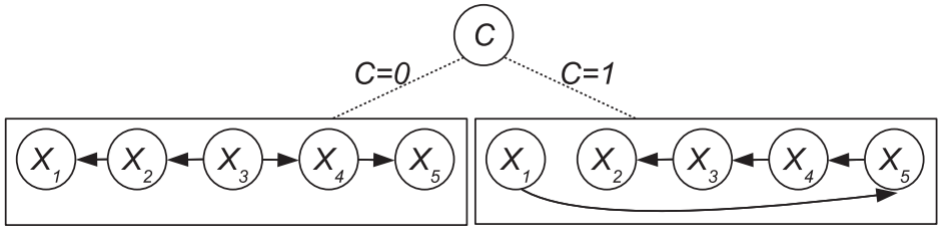

Figura 2.6: Bayesian multinet. Fuente: Bielza y Larrañaga (2014)

El clasificador *naive Bayes* es uno de los clasificadores bayesianos más simples. Su estructura se basa en establecer la variable clase sin padre, las variables predictoras como vértices hijo de la variable clase y sin ningún tipo de dependencias entre las variables predictoras; asume que, dada la variable clase, las variables predictoras son condicionalmente independientes. Se ha demostrado que este tipo de

clasificador bayesiano presenta un desempeño bastante aceptable, incluso realizando suposiciones de independencia condicional tan firmes y que en la realidad no se suelen cumplir. No obstante, se ha intentado mejorar este clasificador relajando estas suposiciones. El clasificador *TAN* se basa en realizar un aumento de la estructura de la red bayesiana **añadiendo arcos entre los diferentes atributos**, de tal forma que cada variable predictora tenga como padres la variable clase y una variable predictora como máximo. De manera análoga, el clasificador *k-DB* permite a las variables predictoras tener como padres a la variable clase y como máximo un número k de variables predictoras, de tal manera que se pueden representar más posibles dependencias entre atributos. El clasificador *k-DB* generaliza tanto al *naive Bayes* como al *TAN* puesto que el *naive Bayes* se puede ver como un *k-DB* con $k = 0$ y el *TAN* como un *k-DB* con $k = 1$.

Otro clasificador bayesiano utilizado con frecuencia es el *semi-naive Bayes*, que particiona el conjunto de atributos; es decir, considera **nuevas variables teniendo en cuenta productos cartesianos de las variables originales**, de tal forma que modela dependencias entre las variables predictoras originales. Por otra parte, existe el clasificador *Bayesian multinet*, cuya estructura está compuesta por varias redes bayesianas locales, donde cada una de ellas representa la **probabilidad conjunta de las variables predictoras condicionada a un subconjunto de valores de la variable clase**. En función de los posibles valores de la variable clase, las dependencias entre las variables predictoras pueden ser distintas, dando lugar a la noción de *asimetría en las declaraciones de independencia* o de *independencias específicas del contexto*.

A la hora de construir un clasificador Bayesiano, existen dos formas de aprender tanto su estructura como sus parámetros: utilizando una aproximación **generativa** o una aproximación **discriminativa**. En el aprendizaje generativo se construye un clasificador modelando la probabilidad conjunta de las características y la correspondiente etiqueta de clase y se realizan predicciones utilizando la regla de Bayes para determinar la probabilidad *a posteriori* de la clase. Los clasificadores aprendidos de forma discriminativa modelan directamente la probabilidad *a posteriori* de la clase, que es más importante para la precisión de la clasificación. En Bielza y Larrañaga (2014) se abordan diferentes métodos que existen en la literatura para resolver estas cuestiones.

2.2.2. Árboles de decisión

El aprendizaje mediante **árboles de decisión** es un método de aproximación de una función objetivo la cual está representada mediante un *árbol de decisión*. Se trata de un clasificador expresado como una partición recursiva del espacio de instancias que consta de *nodos*, *ramas* y *hojas*.

- Los *nodos* representan atributos utilizados para particionar un conjunto de datos en subconjuntos de los mismos de acuerdo con una determinada función discreta de los valores de los atributos de entrada. Es decir, representan *tests* de los valores de los atributos.

- Las *ramas* son los distintos valores de los atributos (nodos), que dan lugar a los diferentes hijos de los nodos (diferentes particiones). Las particiones que se realizan en los distintos nodos del árbol de decisión varían en función de si los atributos son *discretos* o *numéricos*. En el caso de los atributos *discretos*, se realizan las particiones en función de cada uno de los posibles valores de ese atributo. En el caso de los atributos *numéricos*, se particiona teniendo en cuenta diferentes rangos o intervalos de valores.

- Las *hojas* representan las posibles etiquetas de clase. Los árboles de decisión también se conocen como *árboles de clasificación*, de tal forma que este paradigma clasifica una instancia en una determinada clase. No obstante, también existen los *árboles de regresión*, donde la variable de destino puede tomar valores continuos.

De esta manera, a la hora de clasificar una instancia, se comienza desde el nodo *raíz* y, en función de los valores de las variables predictoras de esa instancia, el ejemplo va recorriendo las ramas pertinentes asociadas a esos valores hasta que llega a una hoja, que tiene asignada una clase que se va a utilizar para clasificar la instancia. Por lo tanto, las *ramas* representan las conjunciones de características que conducen a esas etiquetas de clase.

A la hora de llevar a cabo la construcción de ese paradigma clasificatorio, es necesario que las particiones se realicen de tal forma que los subconjuntos resultantes sean lo más *puros* posible, es decir, que en cada subconjunto de instancias de las hojas todos (o casi todos) los ejemplos pertenezcan a la misma clase, lo que conlleva a una correcta partición de los datos proporcionados. La construcción del árbol se basa principalmente en escoger, en cada caso, el atributo que mejor particiona en cada nodo, y para tomar esta decisión se utiliza una *función de medida* del grado de impureza de la partición realizada por cada uno de los atributos considerados. Algunas de las funciones más populares son la **información mutua** (o *ganancia de información*) y el **índice Gini**. La *información mutua* se define como la cantidad que mide una relación entre dos variables aleatorias; concretamente mide cuánto se reduce la incertidumbre (*entropía*, medida de impureza) de una variable al conocer el valor de la otra variable. En el caso de la elección del mejor atributo en los árboles de decisión, interesa la cantidad de información mutua entre una variable predictora y la variable clase, de tal forma que se elige aquel atributo que más reduzca la incertidumbre de la variable clase. La fórmula matemática es la siguiente:

$$I(X_i, C) = H(C) - H(C|X_i) \tag{2.7}$$

donde $H(C)$ representa la entropía de una variable, en este caso de la variable clase, que se define de la siguiente manera:

$$H(C) = -\sum_{j=1}^{|\Omega_C|} p(c_j) log_2 p(c_j) \tag{2.8}$$

y $H(C|X_i)$ representa la entropía de una variable sabiendo el valor de otra variable, en este caso la entropía de la variable clase sabiendo el valor de una variable

predictora. Se define de la siguiente manera:

$$H(C|X_i) = -\sum_{j=1}^{|\Omega_C|}\sum_{l=1}^{|\Omega_{X_i}|} p(c_j, x_{il}) log_2 p(c_j|x_{il}) \tag{2.9}$$

Por otra parte, el *índice de Gini* es un criterio que mide el grado de pureza de un nodo con respecto a los valores de la variable clase. Es una alternativa a la *ganancia de información*. Se define con la siguiente fórmula:

$$Gini(X_i) = \sum_{l=1}^{|\Omega_{X_i}|} p(x_{il})(1 - \sum_{j=1}^{|\Omega_C|} p(c_j|x_{il})^2) \tag{2.10}$$

donde $p(x_{il})$ es la probabilidad del valor l de la variable i y $p(c_j|x_{il})$ es la probabilidad del valor j de la variable clase condicionada al valor l de la variable i. Cuanto mayor es el valor de este criterio, menor es el grado de pureza del nodo. Por tanto se trata de minimizar este índice.

Existen dos maneras de llevar a cabo el proceso de construcción del árbol de decisión: desde **arriba hacia abajo** (*top-down*) y desde **abajo hacia arriba** (*bottom-up*). No obstante, la aproximación *top-down* es la que goza de mayor popularidad. Algunos de los algoritmos más conocidos que llevan a la práctica esta aproximación son **ID3** (Quinlan (1986)), **C4.5** (Quinlan (1993)) y **CART** (Breiman et al. (1984)). Estos algoritmos pertenecen a la familia de los *Top Down Induction of Decision Trees*, que inducen el modelo del árbol de decisión a partir de datos preclasificados. El algoritmo de construcción en el que se basan los árboles de esta familia es el *método de Hunt* (Hunt et al. (1966)).

El algoritmo ID3 construye el árbol de decisión mediante la aproximación *top-down* sin realizar *backtracking*, es decir, lleva a cabo una estrategia de búsqueda voraz a través del espacio de todos los árboles de clasificación posibles. Para tomar la decisión de elegir la variable que aporta mayor información a la hora de realizar las diferentes particiones el algoritmo utiliza la **ganancia de información**. El *ID3* tiene algunos inconvenientes: el árbol se sobreajusta a los datos de entrenamiento, no es capaz de manejar atributos numéricos ni valores faltantes y no realiza un podado del árbol.

El algoritmo C4.5 se desarrolló como una mejora del algoritmo *ID3*. En primer lugar, en lugar de utilizar la *ganancia de información* para elegir la variable predictora más informativa en cada momento, emplea lo que se denomina la **proporción de ganancia** (*gain ratio*), que se calcula diviendo la información mutua entre una variable predictora y la variable clase por la entropía de la variable predictora ($I(X_i, C)/H(X_i)$), que hace justicia con aquellas variables predictoras que tengan un mayor rango de valores y un reparto de frecuencias de instancias similar entre estos valores, de tal forma que no tengan tanta probabilidad de ser elegidas, como ocurre en el *ID3*. Además, permite trabajar con **atributos continuos** definiendo una serie de intervalos que dividen el dominio de valores de los mismos, de tal forma que particionan las instancias en función de los valores de los atributos. Por

otra parte, con respecto al manejo de **datos faltantes**, se estiman los mismos por imputación. Asimismo, trata de lidiar con el sobreajuste del modelo realizando una **poda**, que puede hacerse parando la construcción del árbol en algún punto antes de que clasifique perfectamente los datos (*pre-prunning*) o permitiendo que el modelo se sobreajuste y luego sustituyendo subárboles por hojas (*post-prunning*) etiquetando la hoja con la clase mayoritaria. Además, es capaz de **manejar el ruido**.

Con respecto a CART (*Classification and Regression Trees*), una característica importante de este algoritmo es que es capaz de generar, además de árboles de clasificación, *árboles de regresión*; la predicción en cada hoja se realiza mediante una *media ponderada*. Por otra parte, para elegir el atributo que va a particionar en cada nodo del árbol *CART*, utiliza el *índice de Gini* y, al igual que el *C4.5*, es capaz de manejar atributos categóricos y numéricos y el ruido. También tiene la habilidad de tratar con valores atípicos. No obstante, este algoritmo puede producir árboles inestables.

2.2.3. Inducción de reglas

Otro de los paradigmas utilizados frecuentemente para tareas de clasificación es la **inducción de reglas**. El objetivo de este modelo es encontrar asociaciones o correlaciones entre las variables que describen las instancias de un conjunto de datos mediante la inducción de **reglas de asociación**, que tienen la siguiente forma (Morales y Escalante (2009)):

$$Y \implies Z \tag{2.11}$$

donde Y y Z son conjuntos de literales (o atributos) que tienen asociado un determinado valor y $Y \cap Z = \emptyset$. El significado de esta representación es que las instancias del conjunto de datos que contienen a Y tienden a contener a Z. Los conjuntos Y y Z se denominan **antecedente** y **consecuente** de la regla, respectivamente. Estas reglas también se pueden en expresar en el formato *IF antecedente THEN consecuente*.

En los métodos de clasificación basados en este tipo de estructuras, las reglas se utilizan con la finalidad de llevar a cabo una tarea de clasificación, recibiendo el nombre de **reglas de clasificación**. En las reglas de asociación, tanto en la parte del antecedente como la del consecuente puede aparecer cualquier variable del conjunto de datos y más de un par variable-valor (Morales y Escalante (2009)); no obstante, en las de clasificación la parte del antecedente contiene pares *variable predictora-valor* que se combinan para definir la parte del consecuente, que va a ser **la clase en la que se va a clasificar la instancia con dichos valores en los atributos**. En el caso de los atributos cuyo rango de valores sea continuo, se utilizan particiones de ese rango para discretizarlos. De esta manera, las reglas de clasificación se expresan de la forma *IF combinación_ valores_ variables_ predictoras THEN valor_ clase*.

En la construcción de modelos basados en inducción de reglas, existe una terminología propia de los mismos. A continuación se exponen algunos de los conceptos más importantes:

- **Cobertura de un ejemplo**. Un ejemplo x es *cubierto* por una regla r si pertenece al espacio definido por los límites de r; es decir, si los valores de

los atributos del ejemplo x satisfacen cada una de las condiciones de la regla (pares atributo-valor).

- **Ejemplo positivo**. Un ejemplo x que es cubierto por una regla r es *positivo* si la clase a la que pertenece x coincide con la clase a la que clasifica r.
- **Ejemplo negativo**. Un ejemplo x que es cubierto por una regla r es *negativo* si la clase a la que pertenece x no coincide con la clase a la que clasifica r.
- **Soporte positivo de una regla**. El soporte positivo de una regla que clasifica a la clase c se define como el *número de ejemplos* pertenecientes a la clase c que son cubiertos por dicha regla.
- **Soporte negativo de una regla**. El soporte negativo de una regla que clasifica a la clase c se define como el *número de ejemplos* pertenecientes a una clase distinta de c que son cubiertos por dicha regla.
- **Consistencia de una regla**. Una regla se dice que es *consistente* si no cubre ningún ejemplo negativo.

Un algoritmo de inducción de reglas popular para clasificación supersivada es el denominado **IREP** (*Incremental Reduced Error Pruning*, Fürnkranz y Widmer (1994)). El conjunto de reglas que construye este algoritmo está en *forma normal disyuntiva*, es decir, está formado por una disyunción de reglas, donde cada una de las reglas está constituida por una conjunción de literales (Moujahid et al. (2015)). Suponiendo que la variable clase C toma dos valores, este algoritmo particiona el conjunto de datos D en dos subconjuntos, uno con instancias etiquetadas con un valor de C (**instancias positivas**) denominado D_{pos}, y otro con instancias etiquetadas con el otro valor de C (**instancias negativas**) denominado D_{neg}. Asimismo, cada uno de estos subconjuntos de datos se subdivide en dos subconjuntos, de tal forma que D_{pos} se subdivide en $D_{grow-pos}$ y $D_{prune-pos}$, y D_{neg} se subdivide en $D_{grow-neg}$ y $D_{prune-neg}$. Los subconjuntos $D_{grow-pos}$ y $D_{grow-neg}$ son utilizados por el algoritmo para construir las reglas de manera voraz, escogiendo en cada paso el mejor literal a añadir a la regla de construcción (**regla parcial**, R^{par}). Concretamente, se añade de forma repetida a la regla parcial el literal que da origen a la regla parcial R'^{par} con el mayor valor del criterio siguiente (Moujahid et al. (2015)):

$$v(R^{par}, R'^{par}, D_{grow-pos}, D_{grow-neg}) = cu\left[-\log_2\left(\frac{pos}{pos+neg}\right)+\log_2\left(\frac{pos'}{pos'+neg'}\right)\right] \tag{2.12}$$

donde cu es el porcentaje de ejemplo positivos en $D_{grow-pos}$ que siendo cubiertos por R^{par} están también cubiertos por R'^{par}, pos el número de ejemplos positivos cubiertos por R^{par} en $D_{grow-pos}$, neg el número de ejemplos negativos cubiertos por R^{par} en $D_{grow-neg}$, pos' el número de ejemplos positivos cubiertos por R'^{par} en $D_{grow-pos}$ y neg' el número de ejemplos negativos cubiertos por R'^{par} en $D_{grow-neg}$.

El proceso de crecimiento de una regla se para cuando no se encuentra ningún literal cuya inclusión mejore el criterio de la ecuación (2.12). Tras este proceso, se comienza el podado de la regla, que consiste en un proceso de borrado de literales de manera secuencial, empezando por el último literal introducido en la regla en la fase de crecimiento. El proceso de poda utiliza los subconjuntos de datos $D_{prune-pos}$ y $D_{prune-neg}$ y se realiza mientras se mejore el siguiente criterio (Moujahid et al. (2015)):

$$v(Rule, D_{prune-pos}, D_{prune-neg}) = \frac{pos + (Neg - neg)}{Pos + Neg} \tag{2.13}$$

donde Pos es el número de ejemplos en $D_{prune-pos}$, Neg el número de ejemplos en $D_{prune-neg}$, pos el número de ejemplos positivos en $D_{prune-pos}$ cubiertos por la regla y neg el número de ejemplos negativos en $D_{prune-neg}$ cubiertos por la regla. Tras terminar el proceso de construcción de una regla, se añade al conjunto de reglas, se borran los ejemplos cubiertos por la misma de D y se prosigue con la creación de una nueva regla. El proceso de construcción del conjunto de reglas se para cuando no quedan más instancias en el conjunto de datos D_{pos}.

Otro algoritmo que induce reglas de clasificación es el **RIPPER** (*Repeated Incremental Pruning Produce Error Reduction*, Cohen (1995)), que constituye una mejora del algoritmo IREP. En primer lugar, propone una métrica alternativa para mejorar la fase de poda, que es la siguiente (Moujahid et al. (2015)):

$$v(Rule, D_{prune-pos}, D_{prune-neg}) = \frac{pos - neg}{pos + neg} \tag{2.14}$$

Además, el algoritmo RIPPER incorpora un heurístico para determinar cuándo parar el proceso de adición de reglas. También, tras el proceso visto para IREP, el algoritmo RIPPER efectúa una búsqueda local para optimizar el conjunto de reglas de dos maneras distintas: reemplazando una regla R_i perteneciente al conjunto de reglas por otra regla $R_i^{'}$ siempre y cuando se obtenga un menor error de clasificación en $D_{prune-pos} \cup D_{prune-neg}$ o revisando una determinada regla R_i añadiendo literales para que así se consiga un menor error en $D_{prune-pos} \cup D_{prune-neg}$ (Moujahid et al. (2015)).

La inducción de reglas, aparte de ser un modelo **transparente** y fácilmente **interpretable**, es parecida al paradigma de árboles de decisión puesto que un árbol de decisión se puede descomponer en un conjunto de reglas de clasificación (la inducción de reglas es más **genérica**). Además, la estructura que presentan las reglas es más **flexible** que la forma jerárquica que tiene el árbol de decisión puesto que las reglas son componentes separados que pueden evaluarse de forma aislada y ser eliminados del modelo sin dificultades (Kosina y Gama (2012b)), al contrario que los árboles de decisión que habría que reestructuralos al realizar alguna eliminación de nodos en el modelo. No obstante, las reglas **no garantizan que puedan cubrir toda la región del espacio de entrada** (Gama y Kosina (2011)), de manera que puede ocurrir que llegue una nueva instancia a clasificar y las reglas no cubran dicha instancia. A diferencia de la inducción de reglas, las aproximaciones utilizadas para

clasificación que se basan en árboles de decisión cubren todo el espacio de valores de los atributos de entrada.

Con respecto a la región de entrada de datos que cubren los árboles de decisión, éstos lo particionan en regiones mutuamente exclusivas y en algunas ocasiones no es conveniente este tipo de divisiones del espacio de entrada; en su lugar, sería adecuado que esas regiones de decisión se solapen, y con el modelo de inducción de reglas se puede conseguir.

A la hora de llevar a cabo la predicción de una nueva instancia mediante el paradigma de inducción de reglas, se comprueban los antecedentes de las reglas para ver si los valores de las variables predictoras de la instancia coinciden con la parte izquierda de las reglas. Una vez realizado esto, se comprueba el valor de la variable clase del consecuente del conjunto de reglas obtenido del paso anterior. Si todas ellas tienen asignada la misma clase, el nuevo ejemplo se clasifica en dicha clase; en caso contrario, es necesario resolver el conflicto mediante la utilización de alguna métrica.

2.2.4. Redes neuronales

Las **redes neuronales** son un modelo computacional de interconexión de neuronas en una red que colabora para producir un estímulo de salida (Wikipedia (2019c)). El bloque de construcción de este sistema son las **neuronas artificiales** o **perceptrones**, que son unidades computacionales simples que ponderan las señales de entrada y producen una señal de salida usando una función de activación (Figura 2.7):

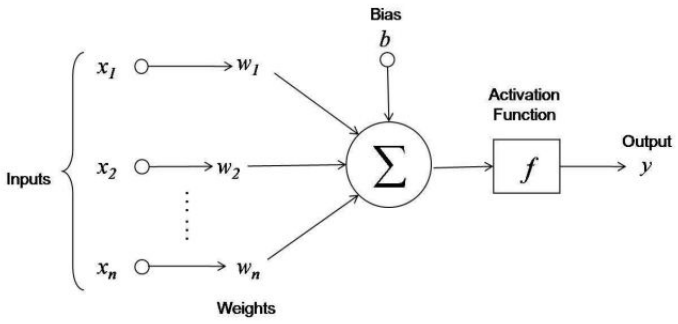

Figura 2.7: Estructura de una red neuronal. Fuente: phuong (2013)

En primer lugar en el perceptrón se recibe una serie de **entradas**, que pueden ser características de un conjunto de entrenamiento o salidas de otras neuronas. A continuación, se aplican unos **pesos** a las entradas, que se suelen inicializar a valores aleatorios pequeños, como valores en el rango de 0 a 0.3. Después, las entradas ponderadas se suman junto con un *sesgo* o *bias* que tiene la neurona (se interpreta como una entrada que permite desplazar la función de activación a la izquierda o a la derecha, que siempre tiene el valor 1.0 y que también debe ser ponderada) que, a su vez, pasan a través de una **función de activación**, obteniendo así las **salidas**.

Esta función de activación es un simple mapeo de la entrada ponderada sumada a la salida de la neurona; es decir, se utiliza para determinar la salida de la red neuronal: mapea los valores resultantes de 0 a 1 o de -1 a 1, etcétera (dependiendo de la función). Las distintas funciones de activación se engloban en dos tipos, *funciones de activación lineales* y *funciones de activación no lineales*. Entre las no lineales existen una gran variedad de ellas: *función sigmoide*, *tangente hiperbólica*, *unidad lineal rectificada*, etc.

La estructura de una red neuronal se compone de las siguientes partes:

- **Capa de entrada**: La capa que toma la entrada del conjunto de datos se denomina *capa de entrada*. Éstas no son neuronas como se describió anteriormente, sino que simplemente pasan los valores de las entradas a la siguiente capa.

- **Capas ocultas**: Las capas posteriores a la capa de entrada se denominan *capas ocultas* y están formadas por aquellas neuronas cuyas entradas provienen de capas anteriores y cuyas salidas pasan a neuronas de capas posteriores.

- **Capa de salida**: La última capa de la red neuronal se denomina *capa de salida* y es la responsable de producir un valor o un vector de valores que dependerán del problema a resolver. La elección de la función de activación en la capa de salida está fuertemente limitada por el tipo problema que se esté modelando.

Entrenamiento de una red neuronal

El primer paso para entrenar una red neuronal es *preparar los datos*, de tal forma que éstos deben ser numéricos y tienen que estar escalados de manera consistente. Con la *normalización* (reescalar al rango entre 0 y 1) y la *estandarización* (para que la distribución de cada columna tenga media cero y desviación estándar uno, de modo que todas las entradas se expresen en rangos similares), el proceso de entrenamiento de la red neuronal se realiza con mucha mayor velocidad.

Uno de los algoritmos de entrenamiento para redes neuronales más populares se denomina **descenso por gradiente** (*gradient descent*). En este algoritmo la red procesa la entrada hacia delante activando las neuronas a medida que se va avanzando a través de las capas ocultas hasta que finalmente se obtiene un valor de salida; esto se denomina *propagación hacia delante* en la red neuronal. La salida del grafo se compara con la salida esperada y se calcula el error. Este error es entonces propagado de nuevo hacia atrás a través de la red neuronal, una capa a la vez, y los pesos son actualizados de acuerdo a su grado de contribución al error calculado (*algoritmo de retropropagación*).

A la hora de llevar a cabo el entrenamiento de la red neuronal, existen diferentes posibilidades de realizarlo según el número de instancias que se procesen antes de actualizar los pesos de la red neuronal, que está definido por el hiperparámetro denominado ***tamaño del lote*** (*batch size*). Una de ellas es procesar todo el conjunto de datos (el tamaño del lote es el número de instancias del conjunto de datos), guardar los errores de todos los ejemplos de entrenamiento y actualizar los parámetros de

la red (*batch gradient descent*). Otra posibilidad es definir el tamaño del lote a una instancia, de tal manera que los pesos en la red neuronal se pueden actualizar al procesar un solo ejemplo de entrenamiento (*stochastic gradient descent*). También existe la opción de definir un tamaño de lote que se encuentre entre una sola instancia y todo el conjunto de entrenamiento, de tal forma que se actualizan los pesos tras un número de instancias establecido (*mini-batch gradient descent*).

Por otra parte, existe un hiperparámetro denominado ***época*** (*epoch*), que establece el número de iteraciones que va a realizar el algoritmo de aprendizaje a través de todo el conjunto de datos de entrenamiento. Una época comprende uno o más lotes. En el caso del *batch gradient descent*, una época tiene un solo lote puesto que se actualizan los pesos tras procesar todo el conjunto de datos, y en el caso del *stochastic gradient descent* una época contiene tantos lotes como ejemplos haya en el conjunto de datos, puesto que se actualizan los parámetros de la red al procesar una instancia.

Por otra parte, el grado en el que se actualizan los pesos es controlado por un parámetro de configuración denominado ***velocidad de aprendizaje*** (*learning rate*). Este parámetro controla el cambio realizado en el peso de la red neuronal para un error determinado. A menudo se utilizan tamaños de peso pequeños tales como 0,1, 0,01 o más pequeños.

Una vez que una red neuronal ha sido entrenada puede ser usada para realizar predicciones. La topología de la red neuronal y el conjunto final de pesos es todo lo que se necesita para implantar el modelo. Las predicciones se realizan proporcionando la entrada a la red y ejecutando una propagación hacia delante que genera una salida que se utiliza como predicción.

2.2.5. k-vecinos más cercanos

El paradigma clasificatorio denominado **k-vecinos más cercanos** (*k-Nearest Neighbours*, kNN) se fundamenta en la idea de identificar el grupo de k objetos en el conjunto de datos de entrenamiento que más cerca está de un nuevo objeto a clasificar, de manera que a este nuevo caso se le asigna la etiqueta de clase más frecuente en ese grupo de k objetos. Este método de clasificación supervisada se basa en tres componentes principales: un *conjunto de datos de entrenamiento etiquetados*, una *métrica de distancia* para calcular las distancias entre distintos objetos y el número k de vecinos más cercanos. De esta forma, utilizando la métrica de distancia correspondiente, se calcula la distancia entre el nuevo caso a clasificar y los casos etiquetados para averiguar cuáles son las k instancias que más próximas están al nuevo objeto; una vez ejecutado este paso, se calcula la clase más repetida en el grupo de k casos más cercanos y se le asigna al nuevo objeto.

El Algoritmo 1 presenta un pseudocódigo para el clasificador kNN básico:

Se está usando la notación:

- $D_{\mathbf{x}}^{K}$ es el conjunto de casos de D de tamaño K más cercano a la instancia $\mathbf{x}$.
- $d^{(i)}$ es la distancia d entre el caso clasificado i y la nueva instancia a clasificar $\mathbf{x}$.

Algoritmo 1 Pseudocódigo del algoritmo kNN. Fuente: Larrañaga et al. (2007)

COMIENZO
  Entrada: $D = \{(\mathbf{x}^{(1)}, c^{(1)}), ..., (\mathbf{x}^{(N)}, c^{(N)})\}$
    $\mathbf{x} = (x_1, ..., x_n)$ nuevo caso a clasificar
  PARA todo objeto ya clasificado $(\mathbf{x}^{(i)}, c^{(i)})$
  Calcular $d^{(i)} = d(\mathbf{x}^{(i)}, \mathbf{x})$
  Ordenar $d^{(i)} (i = 1, ..., N)$ en orden ascendente
  Observar los K casos $D_{\mathbf{x}}^{K}$ ya clasificados más cercanos a $\mathbf{x}$
  Asignar a $\mathbf{x}$ la clase más frecuente en $D_{\mathbf{x}}^{K}$
FIN

En este algoritmo puede ocurrir que, a la hora de averiguar la clase en la que se va a clasificar una nueva instancia, dos o más clases obtengan el mismo número de votos, por lo que hay que establecer *reglas de desempate*. Algunas de ellas pueden ser elegir aquella clase cuyas instancias tengan menor distancia media a $\mathbf{x}$, aquella a la que pertenezca el vecino más cercano, etcétera (Mohanty et al. (2015)). Además, todos los datos deben estar *normalizados* para evitar que las características en el conjunto de entrada con valores más altos dominen el cálculo de la distancia.

El algoritmo kNN lleva a cabo un **aprendizaje perezoso** (*lazy learning*) puesto que no aprende un modelo discriminativo o generativo de los datos de entrenamiento (*eager learning*), sino que memoriza el conjunto de datos de entrenamiento y la labor de predicción se realiza cuando llega un nuevo caso a clasificar. Por lo tanto, en contraste con otros paradigmas clasificatorios, donde hay un proceso de construcción del modelo de predicción y posteriormente su aplicación sobre nuevas instancias, el algoritmo kNN engloba estos dos pasos en uno. Además, es un algoritmo **no paramétrico** puesto que no hace ninguna suposición sobre la distribución de datos subyacente en el conjunto de datos de entrenamiento.

Un aspecto importante de este paradigma clasificatorio es el número K de vecinos que se va a utilizar para decidir la clase a la que pertenece una nueva instancia. Una de las opciones para establecer este número es hacerlo de forma *fija*. Se ha constatado empíricamente que la proporción de casos clasificados correctamente es no monótono con respecto a K, de manera que el rendimiento del clasificador no aumenta siempre al incrementar K; un valor adecuado sería entre 3 y 7 vecinos (Larrañaga et al. (2007)). No obstante, no es práctico asignar un valor fijo a este número para todos los nuevos casos a clasificar, sino que podría modificarse en función de las características de cada uno de ellos (Wang et al. (2006), Cheng et al. (2014), Zhang et al. (2018)).

Con respecto al algoritmo básico del kNN, se han propuesto diferentes variantes con el objetivo de mejorar su rendimiento. Uno de ellos es el **kNN con rechazo**, en el que se tienen que cumplir una serie de condiciones que garanticen la clasificación del nuevo caso, como puede ser una mayoría absoluta de una determinada clase, o que supere un determinado umbral de votos. Otras variantes del kNN básico son

kNN con pesado de casos seleccionados, en el que se le da más importancia a unas instancias que a otras a la hora de realizar la clasificación según la cercanía que tengan con el nuevo caso a predecir; **kNN con pesado de variables**, en el que se le da más relevancia a ciertas variables predictoras que a otras a la hora de calcular las distancias entre los casos clasificados y la nueva instancia a clasificar; etcétera.

2.2.6. Máquinas de vectores soporte

Las **máquinas de vectores soporte** (*support vector machines*, SVMs) son un algoritmo de aprendizaje supervisado que puede utilizarse tanto para desafíos de clasificación como de regresión; no obstante, se utiliza principalmente en problemas de clasificación. En este algoritmo se representa cada instancia (de la muestra de entrenamiento) como un punto en el espacio n-dimensional, con el valor de cada característica mapeándose al valor de una determinada coordenada. A continuación, dado este conjunto de ejemplos de entrenamiento, cada uno perteneciente a una clase, entrenamos una SVM para construir un modelo que prediga la clase de una nueva muestra mediante la construcción de un **hiperplano** que separe las clases de los datos de entrenamiento y maximice el margen entre esas clases (maximice la distancia entre los puntos más cercanos de cada clase al hiperplano de separación óptimo, llamados **vectores soporte**) en el espacio n-dimensional (Figura 2.8):

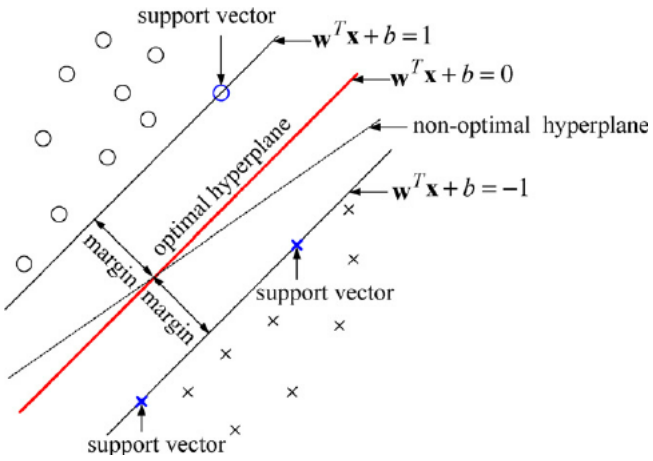

Figura 2.8: SVM. Fuente: Unagar y Unagar (2017)

Las máquinas de vectores soporte, cuando los datos de entrada no son *linealmente separables*, convierten esos datos a un **espacio de mayor dimensión**, de tal forma que en ese espacio el algoritmo puede que encuentre un hiperplano que sea capaz de separar los datos (linealmente). A la hora de realizar este proceso de mapeo de los datos de entrada a un espacio de mayor dimensión, no es necesario calcular la transformación de cada uno de los puntos originales a dicho espacio, sino que solo es necesario calcular **el producto escalar de los puntos en el espacio de mayor dimensión** puesto que es este cálculo lo que se necesita para encontrar el hiperplano que maximice el margen entre las distintas clases. El cálculo de este producto escalar

de los puntos en una dimensión mayor es mucho más sencillo que convertir los puntos originales a dicho espacio a través de lo que se denomina el **truco del núcleo** (*kernel trick*). El *kernel trick* consiste en utilizar una función kernel, que permite obtener el producto escalar entre puntos en un espacio de mayor dimensión sin necesitar la función de mapeo de un punto original a un punto de dimensión mayor.

2.2.7. Regresión logística

La **regresión logística** es un modelo estadístico que se utiliza en aprendizaje automático para describir las relaciones que existen entre un conjunto de variables predictoras y una variable clase con el objetivo de estimar la *probabilidad* de que una instancia pertenezca a una determinada clase. En su formulación original se trata de un *clasificador binario* (predicción dicotómica), pero puede generalizarse para clasificación multiclase.

El nombre del modelo procede de la función sobre la que se fundamenta, que recibe el nombre de **función logística** o **función sigmoide** (Figura 2.9):

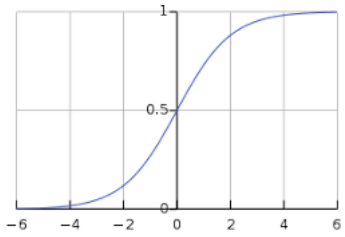

Figura 2.9: Función logística o sigmoide. Fuente: Wikipedia (2019b)

Esta función tiene la característica de que mapea cualquier valor real a un número **comprendido entre 0 y 1** (sin llegar a dar como salida estos dos valores puesto que hay asíntotas horizontales en esos puntos), lo que permite obtener valores de probabilidad. De esta manera, si la probabilidad que estima el modelo es mayor que un determinado umbral (por ejemplo 0,5), entonces se predice que la instancia si pertenece a esa clase (se le asigna un 1, refiríendose a la **clase positiva**) o, en caso contrario, que no pertence (se le asigna un 0, refiriéndose a la **clase negativa**). La fórmula matemática de esta función es la siguiente:

$$f_\theta(\mathbf{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^T \mathbf{x}}} \tag{2.15}$$

donde $\boldsymbol{\theta}^T \mathbf{x}$ es una función lineal de una instancia de entrada $\mathbf{x}$:

$$\boldsymbol{\theta}^T \mathbf{x} = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \tag{2.16}$$

donde $\boldsymbol{\theta}^T = (\theta_0, \theta_1, \ldots, \theta_n)$ es el vector de parámetros que define el modelo de regresión logística. La regresión logística se fundamenta en la idea de que el modelo de regresión lineal no se puede utilizar para tareas de clasificación puesto que produce

como salida un rango infinito de valores. Esto es adecuado para resolver un problema de regresión, pero en este caso el interés reside en solventar un problema de clasificación, donde la variable clase a predecir debe tomar **valores discretos**, por lo que surge el modelo de regresión logística con el objetivo de mapear la salida de un modelo de regresión lineal a una probabilidad de pertenencia a una determinada clase.

Cuando la clase a predecir puede tomar más de dos valores discretos, se utiliza un modelo denominado **regresión logística multinomial**, que es una extensión de la regresión logística binaria. La fórmula matemática es la que se detalla a continuación:

$$f_{\theta^{(i)}}(\mathbf{x}) = \frac{e^{-\boldsymbol{\theta}^{(i)T}\mathbf{x}}}{\sum_{j=1}^{|\Omega_C|} e^{-\boldsymbol{\theta}^{(j)T}\mathbf{x}}} \tag{2.17}$$

donde $f_{\theta^{(i)}}(\mathbf{x})$ en este caso es la probabilidad de que la instancia pertenezca a la clase i, de manera que $f_{\theta^{(1)}}(\mathbf{x}) + \ldots f_{\theta^{(|\Omega_C|)}}(\mathbf{x}) = 1$ debido a que el denominador normaliza cada uno de los términos exponenciales (es la suma de todos los términos exponenciales). Por otra parte, $\boldsymbol{\theta}^{(i)T} = (\theta_1^{(i)}, \ldots, \theta_n^{(i)})$ son los parámetros para predecir la clase i. Esta fórmula recibe el nombre de **función *softmax***, por lo que la regresión logística multinomial también recibe el nombre de **regresión softmax**. Este clasificador predice la clase que reciba el mayor valor de probabilidad.

La clasificación multinomial con el modelo de regresión logística también se puede abordar mediante la aproximación *one vs all*. En este método, si hay $|\Omega_C|$ clases posibles para predecir, se crean $|\Omega_C| - 1$ modelos de regresión logística binaria, de tal forma que, a la hora de aprender cada uno de ellos, se elige una determinada clase (*clase positiva*) y las instancias que no pertenezcan a esa clase se agrupan en una segunda clase (*clase negativa*) con el objetivo de construir el modelo de regresión logística binaria pertinente. La predicción se realiza de la misma forma que en la regresión logística multinomial y, con respecto a la utilización de la aproximación *one vs all* o de la regresión logística multinomial, no existe una superioridad notoria de uno de ellos sobre el otro.

2.2.8. Combinación de métodos de aprendizaje

Con el objetivo de obtener un mejor rendimiento a la hora de realizar labores de predicción, existe la **combinación de métodos de aprendizaje** (*ensemble methods*). La combinación de métodos de aprendizaje es una técnica de aprendizaje automático que está constituida por un conjunto de métodos clasificatorios (o de regresión), de tal forma que las predicciones que lleve a cabo cada uno de ellos se combinan para clasificar una nueva instancia (se agregan para forman un solo clasificador); por ello reciben el nombre de **meta-clasificadores**. De esta forma, no se lleva a cabo un aprendizaje de un solo clasificador, sino de un conjunto de ellos. En general, esta técnica tiene un mejor desempeño que los modelos de clasificación por separado y, para que esto suceda, los clasificadores de los que se compone este meta-clasificador tienen que ser precisos (que tengan una mejor tasa de error que la

realización de una predicción aleatoria) y variados (que cometan diferentes errores al clasificar nuevas instancias) (Dietterich (2000)).

Algunas de las ventajas que presenta la utilización de la combinación de métodos de aprendizaje es que reducen la probabilidad de que haya un **sobreajuste a los datos durante la fase de entrenamiento** y disminuyen tanto el error de **varianza** (los resultados que proporcione el meta-clasificador dependerán menos de las peculiaridades de los datos de entrenamiento) como de **sesgo** (al combinar clasificadores, se aprenden mejor las particularidades del conjunto de datos de entrenamiento). Todo esto se debe a que, si se constituye una combinación de clasificadores variados a partir de un conjunto de instancias de entrenamiento, éstos pueden proporcionar información complementaria con respecto a los patrones que subyacen a los datos y, por tanto, una mayor precisión a la hora de clasificar nuevos ejemplos. No obstante, debido a la complejidad de construcción de esta técnica de aprendizaje, la combinación de métodos de aprendizaje tienen el inconveniente de que **aumenta su tiempo de procesamiento** puesto que no entrenan un solo clasificador, sino muchos.

De los diferentes métodos de *ensemble* que existen, los más conocidos son la **agregación bootstrap** (***bagging***, Breiman (1996)), ***boosting*** (Freund y Schapire (1995)) y ***stacking*** (Wolpert (1992)). El método *bagging* se basa en entrenar cada uno de los clasificadores que componen el meta-clasificador sobre un conjunto de datos del **mismo tamaño que el conjunto de datos original de entrenamiento**, que se obtiene en cada caso escogiendo N instancias del conjunto de instancias original D mediante un **muestreo uniforme y con reemplazamiento de** D. De esta manera, cada clasificador individual se entrena sobre una muestra de datos distinta y habrá instancias del conjunto original que estarán repetidas en dichas muestras. La efectividad de este método se fundamenta en clasificadores individuales que sean inestables, es decir, aquellos cuyo aprendizaje sobre conjuntos de datos de entrenamiento ligeramente distintos realicen predicciones con grandes diferencias, como son los árboles de decisión.

Por otra parte, el método *boosting* se basa en construir un meta-clasificador de forma **incremental**. En este sentido, en este método de *ensemble* se crea una sucesión de clasificadores individuales, donde cada uno de ellos se va a entrenar sobre un conjunto de datos de entrenamiento que va a estar influido por **los ejemplos mal clasificados por los clasificadores individuales previos**. Es decir, al utilizar un clasificador individual para clasificar nuevas instancias, aquellas cuya clase se prediga erróneamente, a la hora de construir el conjunto de ejemplos para entrenar un nuevo clasificador individual, serán elegidas más frecuentemente con el objetivo de centrarse en clasificarlas bien en el siguiente paso. El algoritmo de *boosting* más conocido y exitoso es el denominado **AdaBoost** (*Adaptive Boosting*) (Freund y Schapire (1995)).

En comparación con el método *bagging*, se asemejan en que combinan clasificadores del mismo tipo (métodos de *ensemble* homogéneos), utilizan un sistema de votos para realizar las predicciones y utilizan el mismo método para muestrear los datos de entrenamiento de los clasificadores individuales. No obstante, una de las

diferencias del método *boosting* con respecto al *bagging* es que el primero muestrea del conjunto de datos original teniendo en cuenta el **rendimiento del clasificador individual previo** (las instancias tienen distinta probabilidad de ser elegidas del conjunto de datos original), mientras que en el segundo no (las instancias tienen la misma probabilidad de ser elegidas), por lo que en el primero los clasificadores individuales son dependientes entre ellos y en el segundo no.

Por otra parte, en el *bagging* la clasificación se realiza obteniendo las predicciones individuales y eligiendo la clase más votada, mientras que en el *boosting* el sistema de votos es **ponderado**, por lo que la predicción de cada uno de los clasificadores individuales no pesa igual en la decisión final. También se distinguen en que el método *boosting* se centra en reducir el **sesgo**, es decir, en intentar incrementar la complejidad de los modelos que no son capaces de adaptarse a los datos (*underfitting*), mientras que el método *bagging* se enfoca en reducir la **varianza**, es decir, en reducir la complejidad de los modelos que se ajustan demasiado a los datos de entrenamiento (*overfitting*).

Otro método de *ensemble* popular es el denominado *stacking*. Esta técnica de *ensemble* combina diferentes tipos de clasificadores base en un primer nivel (es un método heterogéneo, a diferencia del *bagging* y del *boosting*), de tal forma que las predicciones que realizan cada uno de ellos se utilizan como **atributos de entrada para entrenar un meta-clasificador** en un segundo nivel con el objetivo de que éste lleve a cabo la decisión final. Al llevar a cabo el entrenamiento sobre las predicciones de varios tipos de clasificadores, el meta-clasificador puede descubrir en cuáles de ellos puede **confiar mayoritariamente**, de manera que aprende cuáles son los patrones que subyacen en los valores de sus predicciones con el objetivo de mejorar el desempeño del meta-modelo. Así, una característica importante que distingue este método del *bagging* y del *boosting* es que estos dos últimos utilizan un sistema de **votación** para saber qué clase ha sido la más predicha por los clasificadores base y no hay un aprendizaje en el meta-nivel, mientras que en el *stacking* sí se realiza ese meta-aprendizaje.

2.3. Algoritmos de aprendizaje no supervisado

Los algoritmos de aprendizaje automático vistos anteriormente asumen que, para cada una de las instancias de entrada que utilizan en la fase de entrenamiento, tienen a su disposición la etiqueta de clase a la que pertenecen. Sin embargo, en muchas aplicaciones del mundo real sucede que no tenemos conocimiento de la categoría a la que pertenecen cada una de las instancias (conjunto de datos no etiquetados), ya sea porque la obtención de las etiquetas resulte caro, sea propenso a errores o directamente sea imposible su adquisición. En este caso, con el objetivo de aplicar aprendizaje automático en esos datos, es necesario recurrir a algoritmos pertenecientes a la categoría de **aprendizaje no supervisado**. La finalidad de este tipo de aprendizaje es descubrir las diferentes categorías que describen las características de los datos no etiquetados. La Tabla 2.3 muestra la estructura de datos con la que trabajan comúnmente los algoritmos de aprendizaje no supervisado.

	X_1	...	X_i	...	X_n
$\mathbf{x}^{(1)}$	$x_1^{(1)}$	...	$x_i^{(1)}$	...	$x_n^{(1)}$
...		...		...	
$\mathbf{x}^{(j)}$	$x_1^{(j)}$	...	$x_i^{(j)}$	...	$x_n^{(j)}$
...		...		...	
$\mathbf{x}^{(N)}$	$x_1^{(N)}$	...	$x_i^{(N)}$	...	$x_n^{(N)}$

Tabla 2.3: Estructura de los datos en un problema de clasificación no supervisado

Uno de los algoritmos de clasificación no supervisada que más se utilizan es el denominado **agrupamiento** (*clustering*). Debido a la gran popularidad que presenta, en este trabajo nos centraremos en abordar propuestas relacionadas con este algoritmo.

2.3.1. Agrupamiento

El **agrupamiento** (*clustering*) es una técnica que consiste en particionar un conjunto de objetos de entrada en una serie de grupos o *clusters*, de tal forma que los objetos que se sitúan dentro de un grupo sean **muy similares** y que haya una **heterogeneidad alta** entre objetos de distintos grupos. Un ejemplo del resultado de esta técnica se muestra en la Figura 2.10.

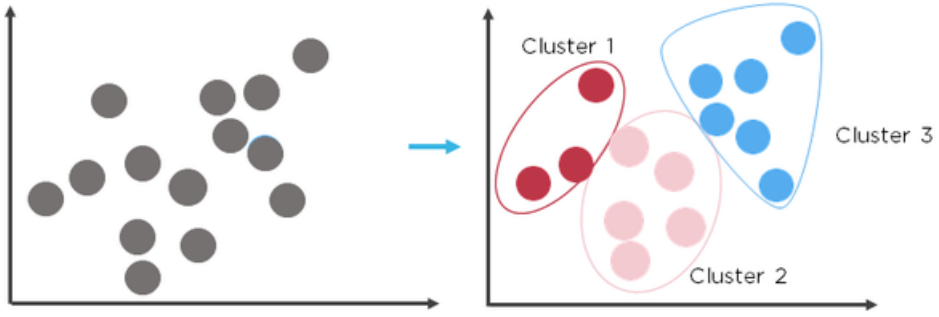

Figura 2.10: Problema de agrupamiento. Fuente: Sharma (2018)

Existen diferentes tipos de métodos para abordar el problema del agrupamiento de objetos. Entre ellos, los métodos más utilizados son el ***clustering* particional**, el ***clustering* jerárquico** y el ***clustering* probabilístico**. En el *clustering* particional el objetivo es dividir el conjunto de instancias de entrada en un número k de *clusters*. El algoritmo de *clustering* particional más conocido es el denominado ***k*-medias** (*k-means*). La versión estándar de este algoritmo, propuesto por Stuart Lloyd en 1957 (Lloyd (1957)), aunque no publicado en una revista hasta 1982, se muestra en el Algoritmo 2 (Lloyd (1982)).

Algoritmo 2 Pseudocódigo del algoritmo k-medias estándar

Paso 1: Crear un agrupamiento inicial de los objetos en k grupos, cada uno representado mediante un *centroide*.
Paso 2: Calcular la distancia de cada uno de los objetos a los centroides para asignarlos al centroide más cercano.
Paso 3: Calcular los k nuevos centroides de los nuevos grupos construidos tras la asignación.
Paso 4: Repetir desde el Paso 2 hasta que se cumpla una condición de parada.

En primer lugar, este algoritmo elige un conjunto de k objetos inicial del conjunto de datos; esta elección puede realizarse de forma **aleatoria**, o bien escogiendo los **primeros k objetos** del fichero, o mediante una **heurística** que permita que los k objetos estén lo más alejados posibles, etcétera. A continuación, cada uno de los objetos se asigna al *cluster* cuyo representante (centroide) se encuentre más cerca de esos objetos; para ello, es necesario utilizar una medida de distancia que, en la versión original del algoritmo, es la distancia Euclídea. Tras esto, se recalculan los centroides de los nuevos grupos construidos computando la media de los objetos incluidos en cada uno de los *clusters*. Todos estos pasos se vuelven a repetir hasta que se alcanza un criterio de convergencia que, comúnmente, suele ser cuando las asignaciones de los objetos a los distintos grupos no cambia de una iteración a otra. De esta manera, el objetivo de este proceso de agrupamiento es minimizar las distancias de los objetos de cada *cluster* al centroide del mismo (Smola y Vishwanathan (2008)):

$$J(r,\mu) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{k} r_{ij}\left\|\mathbf{x}^{(i)} - \boldsymbol{\mu}_j\right\|^2 \tag{2.18}$$

donde $\boldsymbol{\mu}_j$ representa el centroide del grupo j, r_{ij} es una variable que indica si el objeto i está asignado al *cluster* j (se le asigna el valor 1) o no (se le asigna el valor 0), $r = \{r_{ij}\}$, $\mu = \{\boldsymbol{\mu}_j\}$ y $||\cdot||^2$ representa la distancia Euclídea. El centroide $\boldsymbol{\mu}_j$, que corresponde a la media de todos los objetos del grupo j, se calcula de la siguiente forma:

$$\boldsymbol{\mu}_j = \frac{\sum_i r_{ij}\boldsymbol{x}^{(i)}}{\sum_i r_{ij}} \tag{2.19}$$

donde el denominador corresponde al número de objetos asignados al *cluster* j.

Esta versión del algoritmo *k-medias* también fue publicado en 1965 por E.W. Forgy (Forgy (1965)), por lo que en ocasiones se denomina algoritmo de *Lloyd-Forgy* y una característica importante de esta versión es que los centroides se actualizan tras realizar **todas las asignaciones de los objetos a los diferentes grupos**. En este sentido, J. MacQueen propuso en 1967 (MacQueen (1967)) un algoritmo *k-means* en el que considera los primeros k objetos del fichero como los k grupos iniciales y la actualización de los centroides no se realiza tras llevar a cabo todas las asignaciones de los objetos a los distintos grupos, sino que, **cada vez que se**

asigna un objeto a un *cluster*, se recalcula el centroide de ese grupo. El algoritmo de MacQueen es el método de *clustering* particional que más se utiliza.

Otro tipo de método utilizado en *clustering* es el ***clustering* jerárquico**, cuyo objetivo es agrupar los objetos de entrada en una estructura de árbol jerárquico denominada dendrograma (Figura 2.11), de tal forma que los nuevos *clusters* que se construyan dependen de los creados previamente.

En función de cómo se genere esta estructura, el *clustering* jerárquico puede ser **aglomerativo** (*bottom-up*) o **divisivo** (*top-down*). En el aglomerativo se parte de tantos *clusters* como instancias haya en el conjunto de datos y se van agrupando por pares aquellos grupos que más cerca se encuentren. De forma contraria, el divisivo parte de un solo grupo con todos los objetos y se va dividiendo en grupos más pequeños hasta tener tantos *clusters* como instancias haya en el fichero de datos. De esta manera, el aglomerativo tiene un buen desempeño a la hora de identificar pequeños *clusters* y el divisivo en identificar grandes *clusters*.

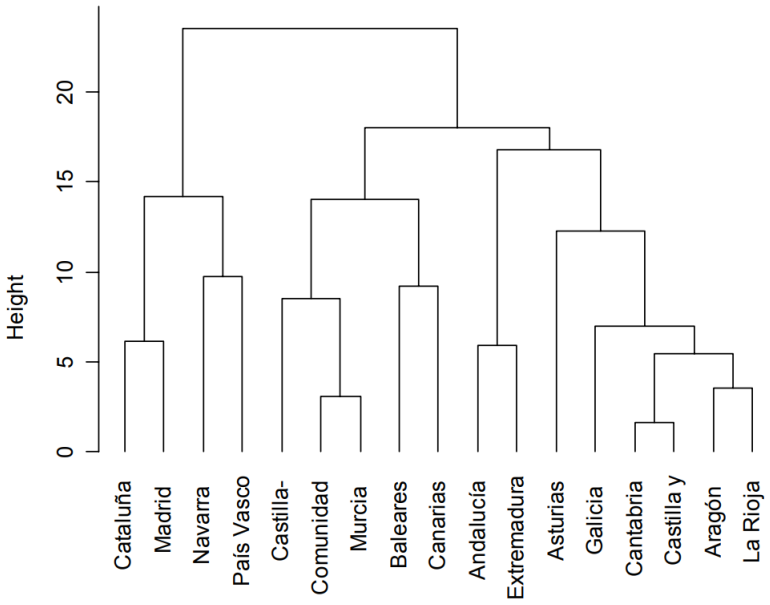

Figura 2.11: Representación de un dendrograma. Agrupamiento de comunidades autónomas en función de la situación laboral de las mismas. Fuente: Jiménez (2019)

La estructura que genera ese tipo de *clustering* aporta más información que los grupos obtenidos por el *clustering* particional puesto que el dendrograma permite obtener diferentes números de grupos según a qué altura se desee cortar la estructura, donde la altura representa la **distancia entre los grupos formados**. De esta forma, con el *clustering* jerárquico **no es necesario especificar de antemano un número k de grupos**, a diferencia del *clustering* particional, aunque es **computacionalmente más costoso que éste**. Además, en el *clustering* jerárquico **no se permiten reasignaciones de los objetos** a otros clusters, característica que sí posee el *clustering* particional.

Otro método de *clustering* conocido es el ***clustering* probabilístico**. En este

tipo de *clustering* se asume que las instancias del conjunto de datos son generadas por una *mixtura de distribuciones de probabilidad*, donde cada una de las componentes que la forman representa un *cluster*, a diferencia de los otros tipos de *clustering* comentados anteriormente, que se basan en construir *clusters* a través de la optimización de criterios que se fundamentan en la distancia entre los objetos de entrada. De esta forma, las instancias no pertenecen a un solo *cluster* (*hard assignment*), como ocurre en el *clustering* particional y jerárquico, sino que considera la incertidumbre presente a la hora de asignar los objetos a los diferentes grupos mediante "**asignaciones suaves**"(*soft assignments*), de tal manera que estas asignaciones no se realizan de forma determinística, sino **probabilística**. Es decir, los objetos pertenecen a cada *cluster* con una determinada probabilidad, que viene establecida por la distribución de probabilidad que define a cada uno de los *clusters*.

Se suele suponer que las distribuciones que representan a cada uno de los *clusters* son distribuciones **Gaussianas**. Para hallar los parámetros de cada una de las distribuciones, se utiliza el método de **estimación de máxima verosimilitud**, cuyo objetivo es maximizar la probabilidad de ajuste de los datos a la mixtura. A la hora de hallar las estimaciones de máxima verosimilitud, surge el problema de que este proceso de optimización no tiene una solución analítica cerrada. De esta manera, uno de los métodos más conocidos para hallar las estimaciones de máxima verosimilitud de los parámetros desconocidos de las distribuciones que representan a los distintos *clusters* es el **algoritmo EM** (Dempster et al. (1977)). Este método comienza estableciendo unos valores iniciales de los parámetros de las distribuciones (en el caso de las distribuciones Gaussianas las medias, las matrices de covarianzas y los pesos de cada componente en la mixtura) e iterativamente alterna entre un paso de **Esperanza** (E) y otro paso de **Maximización** (M). El paso E consiste en, para cada uno de los objetos de entrada, utilizar los parámetros actuales para averiguar sus probabilidades de pertenencia a cada uno de los *clusters* (denominadas **responsabilidades**) y, una vez realizado esto, en el paso M se reestiman los parámetros de las distribuciones que maximizan la verosimilitud de los datos a partir de las probabilidades calculadas. La Figura 2.12 ilustra los pasos del algoritmo en una iteración:

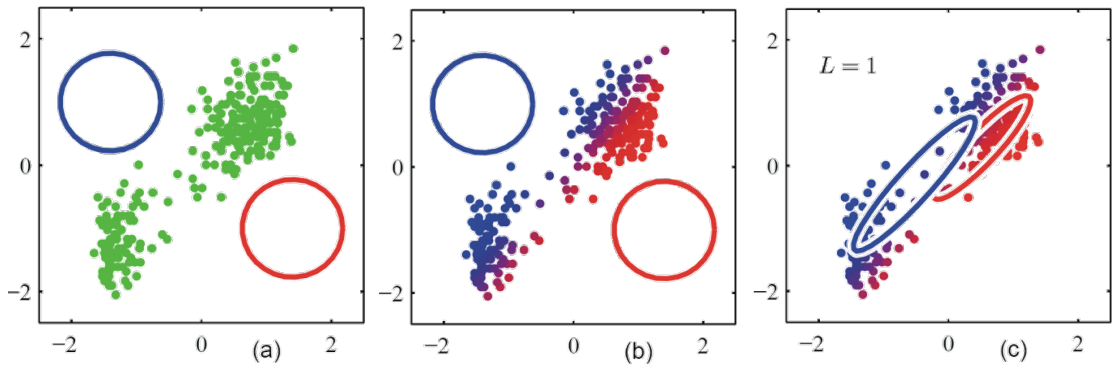

Figura 2.12: Iteración del algoritmo EM. Fuente: Bishop (2006)

En la Figura 2.12, la imagen (a) muestra los puntos del conjunto de datos en

verde, junto con la configuración inicial del modelo de mixturas, compuesto por las componentes Gaussianas cuyos contornos se representan con los colores azul y rojo (sus matrices de covarianzas iniciales son proporcionales a la matriz identidad). En la imagen (b) se muestra el resultado del paso E inicial en el que cada instancia de datos se representa utilizando una proporción de color azul igual a la probabilidad posterior de haber sido generada a partir de la componente azul, y una proporción correspondiente de color rojo dada por la probabilidad posterior de haber sido generada a partir de la componente roja. Por lo tanto, las instancias que tienen una probabilidad significativa de pertenecer a cualquiera de las dos componentes aparecen en color púrpura. En la imagen (c) se expone el paso M, en el que la media de cada componente se desplaza a la media del conjunto de datos, ponderada por la probabilidad de que cada instancia pertenezca a cada componente. La covarianza de cada una de las componentes es igual a la covarianza del color correspondiente.

Con respecto al algoritmo *k-means* visto con anterioridad, se puede considerar que es un tipo particular de *clustering* probabilístico en el que las distribuciones de los *clusters* se suponen que siguen una distribución Gaussiana pero cuyas varianzas son iguales a cero. Entre las desventajas del *clustering* probabilístico, la principal es que **depende mucho de la distribución de probabilidad escogida**, por lo que la elección de un tipo u otro influye de forma notoria en el rendimiento del mismo.

2.4. Redes bayesianas para el descubrimiento de conocimiento

Los clasificadores bayesianos vistos con anterioridad son un caso especial de las redes bayesianas en cuya estructura están representadas las relaciones entre un conjunto de variables predictoras y una variable clase, de tal forma que el objetivo es obtener, a partir de los valores de las variables predictoras, la clase más probable asociada a dichos valores. Las redes bayesianas que generalizan a los clasificadores bayesianos se denominan **redes bayesianas para el descubrimiento del conocimiento**. De esta forma, en general, las redes bayesianas modelan la distribución de probabilidad conjunta de una serie de variables $\mathbf{X} = (X_1, ..., X_n)$ de la siguiente manera (Larrañaga y Bielza (2019a)):

$$p(\mathbf{X}) = p(X_1, ..., X_n) = \prod_{i=1}^{n} p(X_i | Pa(X_i)) \tag{2.20}$$

La modelización de un conjunto de datos mediante una red bayesiana permite principalmente llevar a cabo un **proceso de inferencia** sobre su estructura (conocido como *belief updating* o *probability propagation*). El objetivo de este proceso de inferencia es calcular la distribución de probabilidad de un conjunto de variables de interés dadas observaciones de otras variables, que corresponden a la **evidencia** (Larrañaga y Bielza (2019b)).

Los problemas que plantean los flujos de datos, aparte de poder ser resueltos por los algoritmos vistos con anterioridad, pueden abordarse con modelos que sean

capaces de representar *procesos dinámicos*. Con respecto a las redes bayesianas, existen tres tipos principales para tratar con esta clase de procesos: **redes bayesianas dinámicas** (*dynamic Bayesian networks*), **redes bayesianas de nodos temporales** (*temporal nodes Bayesian networks*) y **redes bayesianas de tiempo continuo** (*continuous time Bayesian networks*).

Las redes bayesianas dinámicas (Dean y Kanazawa (1989)) modelan el estado de cada variable en **intervalos de tiempo discretos**, donde cada uno de estos intervalos representa el valor de cada una de las variables en un instante de tiempo determinado, de manera que la red bayesiana **se repite por cada uno de los intervalos de tiempo**. En esta red bayesiana existen dos tipos de arcos: arcos instantáneos y de transición. Los arcos correspondientes al primer tipo conectan nodos **dentro del mismo intervalo de tiempo**, mientras que aquellos correspondientes al segundo tipo conectan nodos de **diferentes intervalos de tiempo** y especifican cómo cambian las variables de un intervalo de tiempo a otro. Los arcos de transición solo pueden ir hacia delante en el tiempo puesto que el estado de una variable en un instante de tiempo depende de los estados de otras variables (o de ella misma) en **instantes de tiempo anteriores** (Larrañaga et al. (2018)).

Formalmente, el estado de las variables en un instante de tiempo t se representa por un conjunto de variables $\mathbf{X}[t] = (X_1[t], ..., X_n[t])$. Normalmente, en este modelo, debido a que la complejidad de la estructura puede ser alta, se asume que cada uno de los estados de las variables solo dependen del estado de las variables del instante de tiempo **inmediatamente anterior** (modelo de transición Markoviano de primer orden):

$$P(\mathbf{X}[1], ..., \mathbf{X}[T]) = P(\mathbf{X}[1]) \prod_{t=2}^{T} P(\mathbf{X}[t]|\mathbf{X}[t-1]), \tag{2.21}$$

donde $P(\mathbf{X}[1])$ corresponde a la distribución inicial de las variables, $P(\mathbf{X}[t]|\mathbf{X}[t-1]) = \prod_{i=1}^{n} p(X_i[t]|\mathbf{Pa}[t](X_i))$, donde los padres de la variable X_i pueden estar en el mismo intervalo de tiempo o en el anterior que X_i, y T denota el horizonte temporal.

La Figura 2.13 muestra un ejemplo de una red bayesiana dinámica.

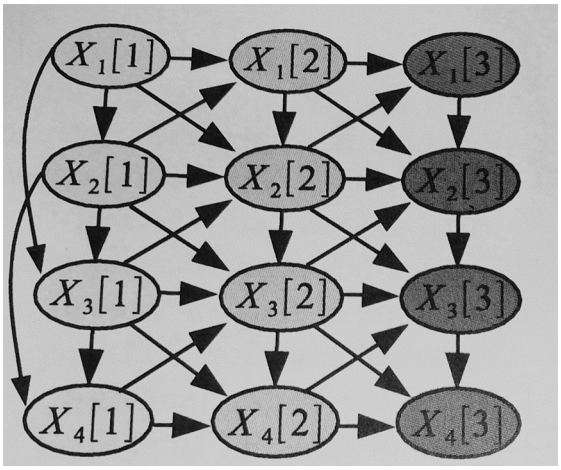

Figura 2.13: Estructura de una red bayesiana dinámica. Fuente: Larrañaga et al. (2018)

Por otra parte, las redes bayesianas de nodos temporales (Galán et al. (2007)) son otro tipo de red bayesiana para manejar procesos dinámicos. Su estructura se fundamenta en la utilización de dos tipos de nodos: **nodos temporales** y **nodos instantáneos**. Un nodo temporal está definido mediante un conjunto de estados, donde cada uno de los estados está determinado por un par ordenado $S = (\lambda, \tau)$, siendo λ el valor de una variable aleatoria y $\tau = [a, b]$ el intervalo de tiempo en el que ocurre el estado de la variable (Larrañaga et al. (2018)), de tal forma que la red modela en los nodos temporales **distintos estados que pueden darse en distintos intervalos de tiempo**. Además, cada uno de los nodos temporales tiene un estado de más que no tiene un intervalo asociado y corresponde al *estado inicial*. Con respecto a los nodos instantáneos, son aquellos que no tienen intervalos de tiempo asociados a los distintos estados de los mismos. Los arcos que conectan los nodos corresponden a relaciones causal-temporal entre ellos. La diferencia principal entre las redes bayesianas dinámicas y las redes bayesianas de nodos temporales es que las primeras se utilizan para modelar una serie de procesos donde se dan muchos cambios de estado, mientras que las segundas se emplean para modelar aquellos procesos donde los estados sufren pocos cambios. Además. en las primeras los intervalos de tiempo son uniformes, mientras que en las segundas se permiten muchas granularidades.

En la Figura 2.14 se muestra un ejemplo de red bayesiana de nodos temporales, donde las variables F (*Failure*), C_1 (*first Component*) y C_2 (*second Component*) son nodos instantáneos, mientras que las variables W (*Water leaks*) y O (*Oil leaks*) son nodos temporales. El tiempo que tarda en producirse alguna fuga de agua o de petróleo depende de los nodos instantáneos.

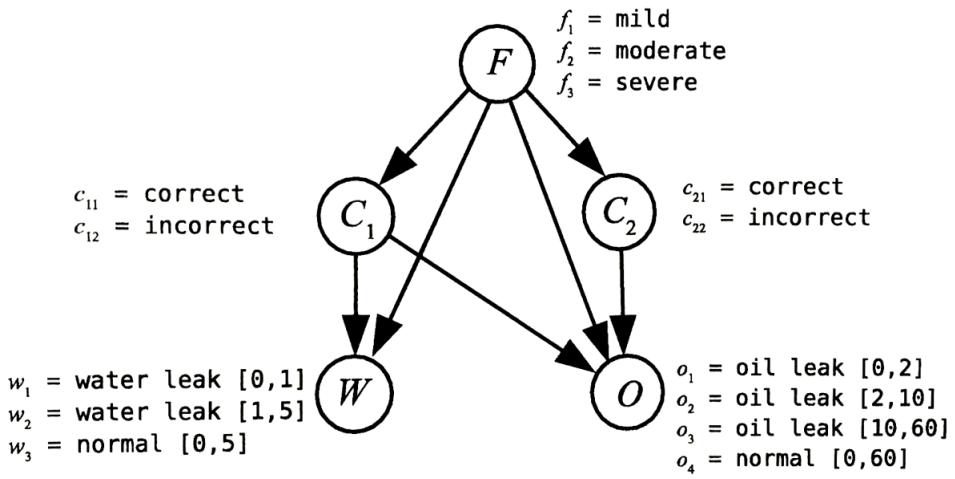

Figura 2.14: Red bayesiana de nodos temporales. Fuente: Larrañaga et al. (2018)

A diferencia de los dos tipos de redes bayesianas anteriores, las redes bayesianas de tiempo continuo (Nodelman et al. (2002)) **modelan directamente el tiempo**. Las redes bayesianas dinámicas representan los estados del modelo en instantes de tiempo discretos, por lo que es difícil consultar a la red bayesiana la distribución de probabilidad del modelo durante el tiempo en el que ocurre un evento; lo mismo ocurre con las redes bayesianas de nodos temporales puesto que su estructura es invariante durante el transcurso del tiempo. Las redes bayesianas de tiempo continuo, para resolver esto, modelan la dinámica del proceso utilizando nodos que representan variables aleatorias cuyo estado **cambia constantemente con el tiempo**, de manera que la evolución de cada una de las variables depende del estado de los padres en la estructura de la red bayesiana (Larrañaga et al. (2018)).

Una red bayesiana de tiempo continuo sobre un conjunto de variables $\mathbf{X} = (X_1, X_2, ..., X_n)$ está compuesta por una **distribución de probabilidad inicial sobre las variables** representada por una red bayesiana y un **modelo de transición continua** representado como un grafo dirigido (posiblemente cíclico), donde cada variable X_i, con posibles valores $x_{i1}, ..., x_{iR_i}$, tiene asociada un conjunto de matrices de intensidades condicionales, una por cada una de las posibles configuraciones de los padres (Larrañaga et al. (2018)). La estructura de una matriz de intensidades condicionales para una instanciación $\mathbf{pa}(X_i)$ de los padres $\mathbf{Pa}(X_i)$ de una variable X_i se muestra a continuación (Larrañaga et al. (2018)):

$$\mathbf{Q}_{X_i}^{\mathbf{pa}(X_i)} = \begin{pmatrix} -q_{x_{i1}}^{\mathbf{pa}(X_i)} & q_{x_{i1},x_{i2}}^{\mathbf{pa}(X_i)} & \cdots & q_{x_{i1},x_{iR_i}}^{\mathbf{pa}(X_i)} \\ q_{x_{i2},x_{i1}}^{\mathbf{pa}(X_i)} & -q_{x_{i2}}^{\mathbf{pa}(X_i)} & \cdots & q_{x_{i2},x_{iR_i}}^{\mathbf{pa}(X_i)} \\ \vdots & \vdots & \ddots & \vdots \\ q_{x_{iR_i},x_{i1}}^{\mathbf{pa}(X_i)} & q_{x_{iR_i},x_{i2}}^{\mathbf{pa}(X_i)} & \cdots & -q_{x_{iR_i}}^{\mathbf{pa}(X_i)} \end{pmatrix} \tag{2.22}$$

donde

$$q_{x_{ik}}^{\mathbf{pa}(X_i)} = \sum_{j \neq k} q_{x_{ik},x_{ij}}^{\mathbf{pa}(X_i)} \tag{2.23}$$

Los elementos fuera de la diagonal representan la probabilidad instantánea de realizar una transición de un valor de la variable X_i a otro para una específica instanciación $\mathbf{pa}(X_i)$ de los padres $\mathbf{Pa}(X_i)$, mientras que los elementos de la diagonal representan la probabilidad instantánea de que la variable X_i deje de tomar el correspondiente estado (de la fila/columna); estas probabilidades corresponden a instanciaciones concretas de los padres de la variable X_i. Con respecto a las evidencias que pueden estar presentes en las redes bayesianas de tiempo continuo, manejan tanto discretas (observación de un valor de una variable en un instante de tiempo) como continuas (valor de una variable en un intervalo de tiempo).

Capítulo 3

Aprendizaje automático para flujos de datos

3.1. Introducción

En la actualidad, existen numerosas aplicaciones que constantemente están generando una cantidad inmensa de información. Entre los dominios donde se encuentran implantadas esas aplicaciones, están los sitemas de monitorización de tráfico de red, redes de sensores para el control de los proceso de fabricación, gestión de redes de telecomunicaciones, modelado de usuarios en una red social, minería web, transacciones bancarias, etcétera. Las técnicas tradicionales de minería de datos realizan un **aprendizaje por lotes** (*batch learning*), de manera que se enfocan en encontrar conocimiento en datos alojados en **repositorios estáticos**; no obstante, debido a las propiedades inherentes en los datos orginados por las aplicaciones mencionadas anteriormente, este tipo de técnicas no se pueden aplicar en dichos datos.

En primer lugar, **no es factible ni tampoco práctico guardar tanta información en bases de datos** puesto que estos almacenes de datos utilizados por las técnicas comunes de aprendizaje automático suelen tener un tamaño fijo, pero la naturaleza de los datos generados continuamente implica que la cantidad de información originada puede llegar a ser infinita. Esta característica es inabordable por los repositorios de información tradicionales, sobre todo a la hora de entrenar los modelos cuyos datos de entrenamiento deben estar en la **memoria principal**, la cual posee poca capacidad de almacenamiento. Por otra parte, las aplicaciones comentadas previamente generan información a una **gran velocidad** y, a diferencia de los algoritmos de aprendizaje automático habituales que construyen modelos estáticos a partir de datos fijos, los patrones que subyacen a dicha información pueden **cambiar dinámicamente** durante el transcurso del tiempo debido al **entorno no estacionario** en el que se originan. De esta manera, es necesario que las técnicas de aprendizaje automático sean capaces de construir modelos que de forma continua se adapten a dichos cambios con el objetivo de que mantengan un buen rendimiento.

Además de lo comentado previamente, los algoritmos de aprendizaje automático tradicionales tienen a su disposición la posibilidad de analizar múltiples veces el

conjunto de datos estático. No obstante, debido a los problemas de almacenamiento de los datos generados actualmente por las aplicaciones del mundo real y a las propiedades que demandan las aplicaciones, **no es abordable realizar múltiples escaneos del conjunto de datos**. Los modelos generados por los algoritmos de aprendizaje automático deben estar actualizados a medida que se van originando nuevos datos para que ofrezcan un buen desempeño. También deben tener en cuenta una serie de restricciones temporales, y esto no se puede llevar a cabo si, a la hora de entrenarlos, realizamos varios ciclos de lectura de los datos.

Por tanto, los sistemas modernos de aprendizaje automático deben tener en cuenta la rapidez y la continuidad con la que se generan los datos hoy en día. Dadas las propiedades de estos datos, éstos reciben el nombre de **flujos de datos** y dada la importancia de extraer conocimiento a partir este tipo de datos, en los últimos años se han realizado una gran cantidad de investigaciones en el campo del **aprendizaje automático aplicado a los flujos de datos**. A la hora de desarrollar algoritmos de aprendizaje automático para manejar flujos de datos, teniendo en cuenta los problemas que presentan los algoritmos tradicionales, deben asumir una serie de desafíos y restricciones:

- Las instancias de entrada del flujo de datos **deben ser procesadas una sola vez** (son descartadas después de ser procesadas), aunque el algoritmo puede recordar algunas instancias.
- No hay un control sobre el **orden en el que los objetos de datos deben ser procesados**.
- El tamaño de un flujo de datos se debe suponer que es **ilimitado**.
- El proceso responsable de generar el flujo de datos puede ser **no estacionario**, es decir, la distribución de probabilidad que subyace a los datos puede cambiar durante el transcurso del tiempo.
- La memoria utilizada por los algoritmos es **limitada**.
- El trabajo realizado por los algoritmos debe cumplir unas **restricciones estrictas de tiempo**.
- El modelo inducido por los algoritmos debe poder llevar a cabo tareas de predicción **en cualquier momento**.
- Durante el transcurso del tiempo, pueden aparecer nuevas clases que requieren ser modeladas para el buen desempeño del modelo.
- Pueden ocurrir, al igual que en las tareas de clasificación, problemas relacionados con valores faltantes, sobreajuste del modelo, variables irrelevantes y redundantes y desbalanceo de las clases.

El aprendizaje que llevan a cabo los algoritmos de aprendizaje automático que tienen en cuenta estas restricciones impuestas por los flujos de datos se denomina **aprendizaje en línea** (*online learning*). Este tipo de aprendizaje supone una versión más restrictiva de otro tipo de aprendizaje denominado **aprendizaje incremental**, que consiste en ir integrando nuevas instancias sin tener que volver a realizar un entrenamiento por completo del modelo (Wikipedia (2019a)). En el aprendizaje incremental se establecen las restricciones de solo procesar una vez cada una de las instancias y construir un modelo similar al que se construiría llevando a cabo un aprendizaje por lotes; no obstante, el aprendizaje en línea, aparte de esas restricciones tiene otras adicionales, que son las mencionadas con anterioridad (Lemaire et al. (2015)).

En el campo de investigación relacionado con el aprendizaje para flujos de datos, al abordarse el problema de extracción del conocimiento desde una perspectiva diferente a las técnicas de aprendizaje automático tradicionales, surge una terminología característica del mismo. Para comprender las propuestas que se van abordar en este trabajo relacionadas con este campo, en el siguiente apartado se va a proceder a la descripción de diferentes conceptos englobados dentro de la terminología vinculada al aprendizaje automático para flujos de datos.

3.1.1. Conceptos

A la hora de tratar flujos de datos para realizar tareas de clasificación, existen una serie de desafíos, que han sido comentados previamente. Para hacer frente a estos problemas que pueden surgir de la aplicación de aprendizaje automático en flujos de datos, existen tres aproximaciones principales (Krawczyk y Wozniak (2015)):

- **Entrenar un clasificador cada vez que se disponga de nuevos datos**. Esta opción suele ser poco adoptada debido a que tiene altos costes computacionales.
- **Detectar cambios en los patrones de los datos** (*concept drifts*), de manera que si son relevantes, se vuelve a entrenar el modelo sobre los nuevos datos tras la ocurrencia del *concept drift*.
- **Llevar a cabo un aprendizaje incremental** con el objetivo de adaptar el modelo a los cambios en el concepto subyacente de los datos de forma gradual

Para entender cómo funcionan estos algoritmos, es imprescindible tener una idea general de las nociones sobre las que se basan.

Concept drift

Uno de los desafíos comentados anteriormente al que deben hacer frente los paradigmas de aprendizaje automático para lidiar con flujos de datos es que la distribución que subyace a los datos puede cambiar durante el transcurso del tiempo (la distribución de los datos es no estacionaria). Este fenómeno se denomina **concept**

drift, de tal forma que la palabra *concept* se refiere al concepto que describe y está inherente en los datos.

Formalmente, el fenómeno de *concept drift* se presenta cuando se producen cambios en la **probabilidad conjunta de las variables predictoras y de la clase que se quiere predecir**, es decir, $p(\mathbf{X}, C)$. Para estimar esta probabilidad, se utiliza la probabilidad a priori de la clase, $p(C)$, y la probabilidad de las variables predictoras condicionada a la variable clase, $p(\mathbf{X}|C)$, de tal forma que $p(\mathbf{X}, C) = p(C)p(\mathbf{X}|C)$. A partir de esta estimación de la probabilidad conjunta y utilizando la regla de Bayes, podemos obtener la probabilidad de la clase condicionada a las variables predictoras, $p(C|\mathbf{X})$.

Teniendo en cuenta los términos probabilísticos mencionados anteriormente, existen dos tipos de *concept drifts*. Según en cual de los términos probabilísticos se produzca un cambio: ***real concept drift***, ***virtual concept drift*** y ***class prior concept drift*** (Khamassi et al. (2016)). El primer tipo de *concept drift* se refiere a cambios que tienen lugar en la **probabilidad** $p(C|\mathbf{X})$, de manera que los límites de decisión para clasificar una instancia a una determinada clase cambian. Con respecto al segundo tipo, sucede cuando se produce un cambio en la probabilidad conjunta de las variables predictoras $p(\mathbf{X})$ y, por lo tanto, en la **probabilidad** $p(\mathbf{X}|C)$, pero no en la probabilidad a posteriori de la clase $p(C|\mathbf{X})$, de manera que esto implica que la región de decisión para llevar a cabo la clasificación de una instancia a una clase en concreto no se ven afectada. En cuanto al tercer tipo de *concept drift*, se refiere a cambios que afectan a la **probabilidad a priori de la clase** $p(C)$, y comúnmente, según el comportamiento del cambio que se produce en dicha probabilidad, se ha clasificado este tipo como *real concept drift* o *virtual concept drift*, aunque es de gran relevancia tenerlo en cuenta como un tipo de *concept drift* aparte. Además, estos tipos de *concept drifts* se pueden dar simultáneamente. En la Figura 3.1 se ilustran los dos tipos principales de *concept drifts*.

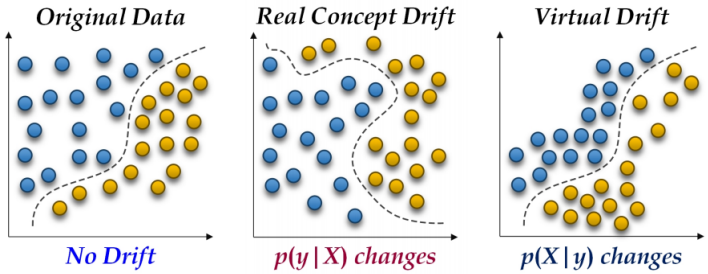

Figura 3.1: Real vs virtual concept drift. Fuente: Pesaranghader et al. (2018)

Otra categorización que se aplica con respecto a los *concept drifts* es en función del **ritmo** con el que ocurren. De esta forma, los *concept drifts* se pueden producir principalmente de manera **abrupta**, **gradual** o **recurrente**. Un *concept drift abrupto* se produce cuando, en cualquier momento, *ocurre* un *concept drift* de forma repentina, de manera que degrada el desempeño del modelo ya que el concepto subyacente de los datos cambia completamente. En cambio, un *concept drift* gradual

tiene lugar cuando el fenómeno de *concept drift* va apareciendo de forma paulatina. El *concept drift* gradual se puede presentar de dos maneras distintas; puede ocurrir que tanto el concepto antiguo como el nuevo estén activos, cada uno con una probabilidad de aparición asociada (los conceptos se alternan), predominando inicialmente el primero y, con el tiempo, desapareciendo con la presencia total del nuevo concepto (*gradual concept drift*, también denominado *gradual probabilistic drift*); por otro lado, el concepto antiguo puede ir sufriendo pequeñas modificaciones hasta la presencia completa del nuevo concepto, de tal manera que esos cambios son sutiles y solo se detectan en un intervalo de tiempo extenso (*incremental concept drift*, también denominado *gradual continuous drift*). En cuanto al *concept drift* recurrente, ocurre cuando conceptos que estuvieron presentes en el pasado vuelven a reaparecer, pudiendo ser cíclico si tienen lugar con cierta periodicidad, o acíclico si no posee la propiedad de periodicidad. La recurrencia del *concept drift* se puede dar de forma gradual o abrupta. En la Figura 3.2 se exponen dichos tipos de *concept drifts*.

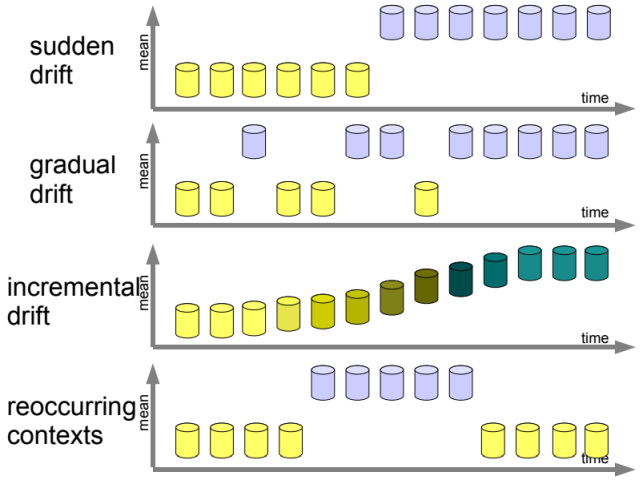

Figura 3.2: Tipos de *concept drift* según el ritmo de cambio. Fuente: Zliobaite (2010)

Existen otras categorizaciones de *concept drifts*, siendo una de ellas si éstos ocurren de forma **local** (si los cambios del concepto de los datos ocurre en algunas regiones del espacio de entrada) o **global** (si los cambios del concepto de los datos ocurre en todo el espacio de entrada). Por otra parte, teniendo en cuenta la predictibilidad de los *concept drifts*, éstos se pueden clasificar en **predecibles** (si siguen un patrón) o **impredecibles** (si son totalmente aleatorios).

Modelos de ventanas

Uno de los conceptos más recurrentes en la terminología de los algoritmos de aprendizaje automático aplicados a flujos de datos son las **ventanas deslizantes**. El objetivo de los algoritmos que se basan en ventanas deslizantes es **manejar los *concept drifts***, y se fundamenta en la idea de que las instancias más recientes

del flujo de datos tienen mayor relevancia a la hora de describir la distribución de probabilidad actual que subyace a los datos. Con respecto a este método, existen tres modelos utilizados frecuentemente: el modelo ***landmark window***, el modelo ***sliding window*** y el modelo ***damping window*** (Zhu y Shasha (2002)).

El modelo *landmark window* (Figura 3.3) se basa en utilizar **toda la historia del flujo de datos desde un punto de inicio en el pasado denominado *landmark* hasta el instante de tiempo actual**, de tal forma que los datos que se encuentren antes del *landmark* no se tienen en cuenta. De esta manera, el *landmark* se mantiene fijo, pero el punto que representa el instante de tiempo actual se va desplazando, por lo que el tamaño de la ventana va aumentando y se van teniendo en cuenta más datos. Un caso particular de este modelo es cuando el *landmark* se establece en el instante de tiempo del origen del flujo de datos, por lo que el modelo tiene en cuenta todo el flujo de datos generado hasta el momento actual. El problema que tiene el modelo *landmark window* es que es **difícil establecer el *landmark* idóneo** y todos los instantes de tiempo posteriores al punto inicial tienen la **misma importancia** a la hora de construir el modelo.

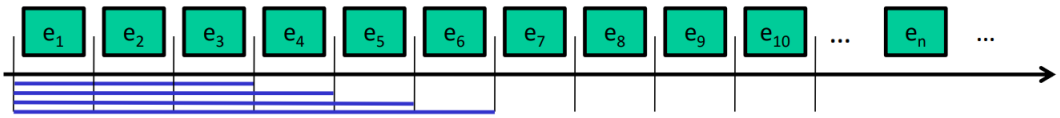

Figura 3.3: Modelo *landmark window*. Los e_i corresponden a cada una de las instancias que llegan en cada uno de los instantes de tiempo i, y las líneas azules representan las instancias que se tienen en cuenta para la construcción del modelo cada vez que llega una nueva instancia. Fuente: Ntoutsi et al. (2015)

Por otra parte, en el modelo *sliding window* (Figura 3.4) solo se tiene en cuenta la información **más reciente del flujo de datos** (desde el instante de tiempo actual hasta un instante de tiempo en el pasado). Esta información está definida por una ventana temporal cuyo tamaño define la cantidad de datos que van a ser relevantes para la construcción del modelo. De esta manera, la primera ventana del flujo de datos cubre el primer conjunto de datos que se van a utilizar para el entrenamiento del modelo y, cuando llega el siguiente instante de tiempo, la ventana se desplaza un cierto número de unidades en el tiempo y se elimina de la misma la instancia de datos más antigua para mantener el tamaño de la ventana. Así, este proceso se repite a medida que va avanzando el tiempo. Esta ventana puede ser de tamaño fijo o variable, y la ventaja principal de ese modelo de ventana es que evita que datos obsoletos influyan en el proceso de generación del modelo de aprendizaje automático.

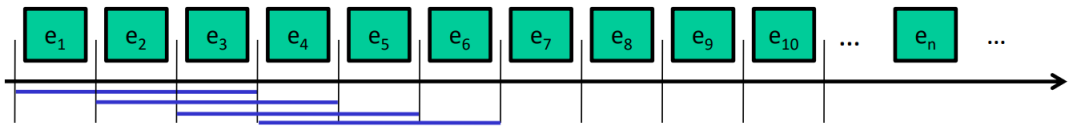

Figura 3.4: Modelo *sliding window*. Fuente: Ntoutsi et al. (2015)

Con respecto al modelo *damping window* (Figura 3.5), al igual que el modelo *sliding window*, considera la información más reciente como relevante para la construcción del modelo de aprendizaje automático, pero en este caso se asignan una **serie de pesos a los datos en función del instante de tiempo en el que se han generado**. De esta forma, aquellas instancias que son más recientes van a recibir un peso mayor que aquellas que provienen de instantes de tiempo anteriores, por lo que van a influir más en la construcción del modelo inducido por el algoritmo de aprendizaje automático. Este modelo de ventana no descarta instancias completamente, sino que asigna pesos pequeños a los objetos antiguos, y para controlar el decrecimiento de los pesos a medida que se vuelve hacia atrás en el tiempo existe lo que se denomina un **factor de desvanecimiento** (λ), de tal manera que, cuanto mayor es su valor, menor importancia tienen los datos del pasado.

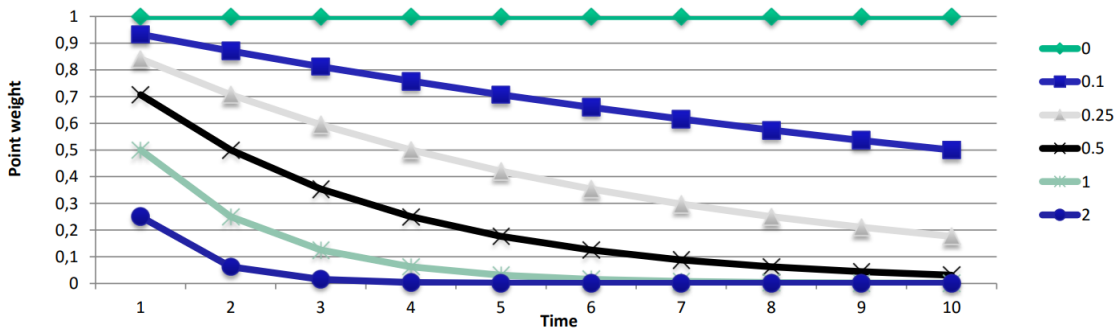

Figura 3.5: Modelo *damping window*. Efecto del valor del factor de desvanecimiento λ. A medida que se retrocede al pasado (hacia la derecha en la gráfica), el peso asignado a cada instancia que ha llegado en instantes de tiempo anteriores disminuye. Fuente: Ntoutsi et al. (2015)

Con respecto a los tres modelos de ventana comentados anteriormente, el modelo *landmark window* se puede transformar en el modelo *damping window* añadiendo pesos de influencia de los datos sobre la construcción del modelo de aprendizaje automático, y el modelo *landmark window* se puede convertir al modelo *sliding window* realizando todo el proceso de construcción dentro de una ventana temporal. Por otra parte, existe también otro modelo de ventana denominado ***tilted window***, que también se utiliza bastante y que consiste en guardar una síntesis del flujo de datos con memoria limitada en **diferentes niveles de granularidad temporal**. De esta forma, la información resumida de aquellos datos que son más recientes se almacena en un nivel de granularidad temporal alto (cada cuarto de hora, por

ejemplo) y la de objetos más antiguos en un nivel menos alto de granularidad (cada día, por ejemplo). Al igual que el modelo *damping window*, se concentra en datos que son más recientes y no descarta plenamente objetos del pasado.

Métodos de evaluación de tareas de clasificación para flujos de datos

Para comprobar el rendimiento de los modelos construidos para tratar con flujos de datos también se utilizan las métricas que se emplean para estimar **el desempeño de algoritmos de aprendizaje automático tradicionales** debido a su posible aplicabilidad; por lo tanto, se usan una gran variedad de métricas presentes en la literatura de métodos de aprendizaje automático convencionales. No obstante, para llevar a cabo la evaluación del desempeño de un clasificador de flujos de datos, existe un método popular denominado ***prequential error*** o ***interleaved test-train***. Este método consiste primero en utilizar una nueva instancia cuya etiqueta no se tiene para comprobar la eficacia del modelo actual, de tal forma que se lleva a cabo la predicción de la etiqueta. Una vez realizado esto, se obtiene la etiqueta de dicha instancia y se calcula el error cometido por el modelo. Tras esto, se actualiza el modelo con dicha instancia y se utiliza este modelo para predecir la siguiente instancia. La fórmula que calcula el *prequential error* cometido hasta un determinado instante de tiempo t en un flujo de datos es la siguiente:

$$E(S) = \frac{1}{t}\sum_{i=1}^{t} \mathbb{1}(h_{i-1}(\mathbf{x}^{(i)}), c^{(i)}) \tag{3.1}$$

donde h_{i-1} es el modelo actual utilizado para predecir la instancia del instante i y $\mathbb{1}(\cdot)$ es la función indicatriz, cuya salida es 1 si la clase predicha para una instancia $\mathbf{x}^{(i)}$ por el modelo actual h_{i-1} es igual a la clase $c^{(i)}$ de la misma; en caso contrario la salida es 0.

Con respecto al *prequential error*, en Gama et al. (2009) afirman que este método es pesimista, es decir, informa de errores más altos de los que en realidad son. El *prequential error* definido sobre todo el flujo de datos está notablemente influenciado por la primera parte de la secuencia de errores, es decir, cuando se han utilizado pocas instancias para realizar el entrenamiento del clasificador. Por ello, en Gama et al. (2009) proponen utilizar el *prequential error* con una **ventana deslizante** o **mecanismos de olvido**, que permiten que dicho error converja al error de Bayes (el error de predicción más bajo posible). El modelo que se utiliza para clasificar el primer ejemplo en un flujo de datos es diferente del que se utiliza para clasificar una instancia en un instante de tiempo posterior, es decir, el modelo evoluciona con el tiempo; de esta manera, su rendimiento mejora y esto se refleja mejor utilizando una ventana deslizante o mecanismos de olvido, evitando que los errores del pasado influyan en el presente igual que los recientes.

Para llevar a cabo la evaluación de clasificadores en flujos de datos, también esta presente la opción de utilizar el ***holdout error***, que consiste, en el caso de que se suponga que no van a aparecer *concept drifts* en el flujo de datos, en utilizar un conjunto estático de ejemplos. En el caso de que se suponga la aparición de *concept*

drifts, se aplica el modelo actual en intervalos de tiempo regulares posteriores en el tiempo a los intervalos de tiempo de entrenamiento sobre un conjunto de ejemplos no utilizados en el entrenamiento del modelo (*look ahead* en el flujo de datos), siendo éste un estimador insesgado. No obstante, este método necesita bastantes instancias, y en el entorno de los flujos de datos puede ocurrir que no se disponga de las verdaderas etiquetas de las instancias. En el cálculo del *prequential error* no es necesario saber las etiquetas de todas las instancias del flujo de datos puesto que el valor del error se obtiene utilizando aquellas instancias de las que sí se disponga de etiqueta. Por ello, se utiliza el *prequential error* como una aproximación del *holdout error*.

3.2. Algoritmos de aprendizaje supervisado

La mayor parte del esfuerzo dedicado para desarrollar algoritmos de aprendizaje automático para flujos de datos se ha enfocado en la realización de propuestas relacionadas con **aprendizaje supervisado**. Existen diversas revisiones dedicadas al aprendizaje automático para tareas de clasificación (Aggarwal (2014), Nguyen et al. (2015), Lemaire et al. (2015)); no obstante, en estas revisiones, por cada uno de los paradigmas clasificatorios, no se mencionan muchas propuestas, por lo que en este trabajo pretendemos **aportar más artículos que tratan el aprendizaje automático para flujos de datos**, enfocándonos en aquellas propuestas más recientes. Además, por cada uno de los algoritmos de aprendizaje automático, vamos a añadir una **tabla comparativa entre las diferentes propuestas**, con el objetivo de que el lector adquiera una idea global de las características que cada propuesta tiene.

3.2.1. Clasificadores bayesianos

Uno de los clasificadores Bayesianos más utilizados para realizar tareas de clasificación de flujos de datos es el **naive Bayes**. Esto se debe principalmente a su gran facilidad para adaptarlo para realizar un aprendizaje en línea, debido a que su estructura es simple (puesto que la complejidad solo depende del número de variables predictoras) y su consumo de memoria es bajo (debido a que únicamente se requiere una distribución de probabilidad condicional por cada una de las variable predictoras). Para llevar a cabo un aprendizaje en línea del *naive Bayes*, es suficiente con actualizar los contadores utilizados para hallar las diferentes probabilidades representadas por el modelo, permitiendo llevar a cabo de esta manera una **estimación incremental** de las mismas.

Una propuesta que se basa en el modelo *naive Bayes* para tratar con tareas de clasificación de flujos de datos es la planteada en Salperwyck et al. (2015). En este trabajo desarrollan un algoritmo denominado **weighted naive Bayes** (*WNB*), que se fundamenta en asignar pesos a las variables predictoras del clasificador *naive Bayes* para lidiar con flujos de datos y averiguar dicha ponderación realizando una estimación incremental de los pesos. Para hallar los pesos óptimos de las diferentes variables predictoras en línea, utilizan un modelo gráfico similar a una red neuronal

(Figura 3.6), donde los valores de entrada son las probabilidades asociadas a cada uno de los valores de las variables explicativas condicionadas a cada uno de los valores de la variable clase. Los pesos que se aplican a estos valores se optimizan utilizando el algoritmo de retropropagación del gradiente, que los va actualizando utilizando el método de **descenso de gradiente estocástico**, que se basa en utilizar una única instancia en cada iteración que se modifican los pesos; los resultados de la red son las probabilidades *a posteriori* de la clase. Para calcular las probabilidades de entrada a la red, utilizan tres métodos: dos de discretización incremental de dos capas basados en estadísticos de orden, en los que en el primer nivel se utiliza el método **cPid** o **GkClass** y en el segundo la discretización **MODL**, y un tercer método que es la **aproximación Gaussiana**. El método cPid es una modificación del método PiD (Gama y Pinto (2006)) que utiliza memoria constante; el método PiD (*Partition Incremental Discretization*) consta de dos capas, en las que la primera simplifica y compacta los datos y se va actualizando de forma incremental, mientras que la segunda realiza otra discretización uniendo intervalos de la primera capa. El método GkClass (Greenwald y Khanna (2001)) calcula información resumida de los cuantiles de un flujo de datos utilizando memoria limitada; dicha información consiste en un pequeño número de instancias del flujo de datos que permite dar respuestas aproximadas a cuestiones relacionadas con los cuantiles de los datos. El método MODL (Boullé (2006)) se basa en el número de clases y evalúa todos los posibles intervalos para las variables numéricas y grupos para las variables categóricas; la evaluación de la calidad del modelo se basa en un enfoque bayesiano.

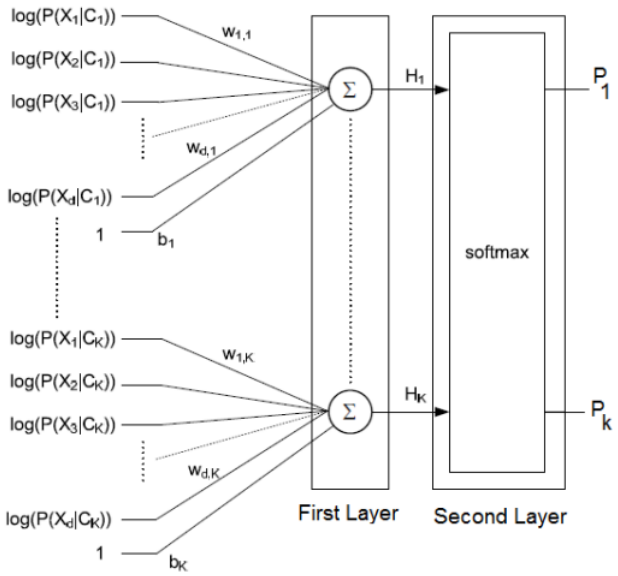

Figura 3.6: Modelo gráfico utilizado para la optimización de los pesos. k es el número de clases existentes en los datos. Fuente: Salperwyck et al. (2015)

De la misma manera que en la propuesta anterior, Krawczyk y Wozniak (2015)

también proponen un *weighted naive Bayes*, pero en este caso, en lugar de ponderar las variables explicativas, asignan pesos a las **instancias del flujo de datos**. Estos pesos indican el nivel de importancia que tienen a la hora de utilizarlas para llevar a cabo el entrenamiento del clasificador, concretamente para calcular las probabilidades *a posteriori* de cada una de las clases. Para establecer los pesos de cada una de las instancias, utiliza un módulo de ponderación, cuya función es relevante a la hora de adaptar rápidamente el clasificador *naive Bayes* a la presencia de *concept drifts* de forma **automática** (no utiliza un detector de *concept drift*); a medida que pasa el tiempo, se encarga de degradar la influencia de las instancias, de tal forma que se descartan al transcurrir una cantidad de tiempo determinada. A las instancias más recientes se les asigna un peso igual a 1, mientras que el peso de instancias menos recientes se obtiene utilizando una **función sigmoide**, en la que interviene un factor β, que define la rapidez con la que se degrada la importancia de las instancias. Para descartar los ejemplos, emplean un umbral ϵ, de manera que las instancias que tengan un peso menor que ese umbral se descartan. Para discretizar variables continuas, a diferencia de la propuesta anterior, utilizan el **esquema supervisado de Fayyad e Irani basado en MDL** (Fayyad y Irani (1993)). El algoritmo que proponen lo denominan **weighted naïve Bayes for concept drift** (*WNB-CD*).

Por otra parte, en la propuesta planteada en Bifet y Gavaldà (2007), para manejar el *concept drift*, en lugar de asignar un peso a las instancias para determinar su influencia en el entrenamiento del clasificador como se propone en Krawczyk y Wozniak (2015), utilizan un algoritmo denominado **ADWIN**, cuya función es mantener una ventana de instancias de longitud variable en línea según el ritmo del cambio del concepto de los datos producidos dentro de la ventana, favoreciendo que el usuario no tenga que preocuparse de elegir un tamaño de ventana. Además, proponen otra versión del algoritmo *ADWIN* que reduce los costes computacionales manteniendo las mismas garantías de rendimiento, denominado **ADWIN2**. Para comprobar la eficacia de este último algoritmo, lo combinan con el clasificador **naive Bayes** debido a la facilidad de observar en el mismo los *concept drifts* que puedan ocurrir. La composición del algoritmo *ADWIN2* con el clasificador *naive Bayes* lo llevan a cabo de dos formas distintas: utilizando *ADWIN2* para **monitorizar los errores del modelo** y llevar a cabo una comprobación de la corrección del mismo, e **integrando dicho algoritmo dentro del clasificador *naive Bayes*** para mantener actualizadas las diferentes probabilidades condicionadas.

Cuando adquirimos datos para entrenar un algoritmo de aprendizaje automático, en muchas ocasiones ocurre que no tenemos el *ground truth* de algunas instancias del flujo de datos; uno de los motivos puede ser que las etiquetas de las instancias no lleguen en el momento en el que se obtienen las instancias, sino que tienen un determinado retardo. Por ello, en Borchani et al. (2011), a diferencia de las propuestas anteriores, plantean un **algoritmo semi-supervisado** para manejar el *concept drift* en los flujos de datos; concretamente, controlan la ocurrencia de un *real concept drift*, un *virtual concept drift* o de los dos a la vez. Para comprobar si se han producido cambios en la distribución subyacente a los datos utilizan la **divergencia de Kullback-Leibler** (*KL*), que mide la divergencia entre dos funciones de distribu-

ción de probabilidad, en este caso aquellas correspondientes a dos bloques de datos consecutivos del flujo de datos, una de las cuales (la correspondiente a los datos del bloque antiguo) se toma como referencia. Para determinar si se ha producido un *concept drift*, se establece la hipótesis nula de que los datos de dos bloques consecutivos proceden de la misma función de distribución de probabilidad y, utilizando el **método bootstrap** (realiza un muestreo repetido con reemplazamiento a partir de los datos con la misma probabilidad de elegir cada instancia), se acepta o rechaza la hipótesis nula de igualdad de distribuciones. En el caso de que se detecte un *concept drift* (se rechaza la hipótesis nula), se aplica el **algoritmo EM** sobre las nuevas instancias para construir el nuevo clasificador. Uno de los clasificadores que utilizan es el **naive Bayes**.

Otra propuesta que utiliza el clasificador *naive Bayes* es la planteada en Kishore Babu et al. (2016), donde desarrollan el algoritmo denominado **rough Gaussian naive Bayes classifier** (*RGNBC*). Este algoritmo consiste en utilizar un clasificador *naive Bayes Gaussiano* (considera que los valores continuos asociados con cada una de las clases a predecir se distribuyen según una distribución Gaussiana) añadiéndole la capacidad de detectar *concept drifts*, concretamente ***concept drifts* recurrentes**, de forma automática mediante la **teoría de conjuntos aproximado** (*rough set theory*, a diferencia de otras propuestas mencionadas anteriormente), una herramienta matemática para tratar con información y conocimiento impreciso, inconsistente e incompleto (Pawlak (1982)). La teoría de conjuntos aproximados se utiliza para aproximar de forma precisa la región de decisión descrita por los datos calculando una aproximación inferior y otra superior de dicha región. La **aproximación inferior** de la región de decisión abarca aquellas instancias que se encuentran dentro de dicha región, y la **aproximación superior** abarca aquellas instancias que se encuentran fuera de los límites de los datos. Si el ratio entre la aproximación inferior y la aproximación superior es menor que un umbral (la precisión de aproximación), quiere decir que las aproximaciones se diferencian bastante, por lo que se establece que ha ocurrido un *concept drift*. La detección del *concept drift* se integra dentro de un conjunto de pasos que conforman el algoritmo propuesto, de tal forma que el primer paso es construir un clasificador *naive Bayes* Gaussiano inicial mediante la creación de **tablas de información**, donde se almacenan las medias y las varianzas de cada uno de los atributos en cada intervalo de tiempo, que se utilizan para calcular las probabilidades condicionadas del clasificador. A continuación, se comprueba si se **ha producido un *concept drift***; si ocurre, se **seleccionan atributos** mediante la entropía y se actualiza el clasificador utilizando el nuevo conjunto de instancias sin almacenarlas, modificando las tablas de información calculadas previamente. Para llevar a cabo las tareas de clasificación, se utiliza la **probabilidad *a posteriori*** de las diferentes clases junto con una **función objetivo** que tiene en cuenta las métricas de sensibilidad, especificidad y precisión y que ponderan las probabilidades *a posteriori*.

En las propuestas anteriores se han planteado diferentes trabajos utilizando el clasificador *naive Bayes*. A diferencia de dichas propuestas, en Roure (2002) se propone un algoritmo incremental para realizar el aprendizaje de un clasificador **TAN**

con el objetivo de crear una red bayesiana que mejore el desempeño del clasificador *naive Bayes* añadiendo dependencias entre variables. Este algoritmo comienza construyendo un clasificador TAN utilizando el algoritmo de *Chow y Liu* (Chow y Liu (1968)). La dirección de los arcos que establecen en este trabajo tras aplicar dicho algoritmo es escoger uno de los nodos del primer enlace introducido e ir estableciendo la dirección desde el nodo ya presente en la estructura hacia los nodos que se van introduciendo. Una vez construido el TAN inicial, se calculan una **serie de estadísticos suficientes de los datos utilizados para la construcción del clasificador**, que corresponden a un conjunto de contadores de instancias, y se almacenan junto con el orden en el que fueron introducidos los arcos. Cuando llega un nuevo conjunto de datos, **se actualizan los estadísticos suficientes con dichos datos** utilizando las propiedades aditivas de los estadísticos y **se vuelven a calcular los costes de los arcos** (*información mutua*). Tras el cálculo de los nuevos costes, si existe algún arco cuyo coste no es el más alto entre todos los candidatos a ocupar su posición, **se vuelve a aplicar el algoritmo de *Chow y Liu* desde la posición de ese arco** (reconstrucción del árbol desde dicha posición). La heurística que proponen en este trabajo la denominan **ACO** (*Arches in Correct Order*).

Las propuestas abordadas hasta el momento se centran en utilizar un solo tipo de clasificador bayesiano (o *naive Bayes* o *TAN*, uno de los dos). A diferencia de estos trabajos, una propuesta que aborda más de un clasificador bayesiano para adaptarlos a entornos dinámicos es la planteada en Stella y Amer (2012). En este trabajo combinan las características de las redes bayesianas en tiempo continuo con la de los clasificadores Bayesianos *naive Bayes* y *TAN*, de tal manera que los modelos que construyen los denominan **clasificador *naive Bayes* de tiempo continuo** (*CTNB*) y **clasificador *TAN* de tiempo continuo** (*CTTANB*). Concretamente, estos clasificadores utilizan la estructura del clasificador *naive Bayes* y *TAN* respectivamente y, además, representan la evolución en tiempo continuo de las variables que los forman, excepto la variable clase, que no depende del tiempo. Para implementar estos modelos, establecen el objetivo de calcular la probabilidad a posteriori de la clase dadas las instanciaciones de las variables en diferentes instantes de tiempo (*J-evidence-stream*), es decir, $P(C|(\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^J))$, donde $\boldsymbol{x}^j$ es la instanciación de las variables en el instante j. Para calcular esta probabilidad, interviene la probabilidad $P((\boldsymbol{x}^1, \boldsymbol{x}^2, ..., \boldsymbol{x}^J)|C)$ (aparte de la probabilidad a priori de la variable clase), que denominan *verosimilitud temporal* y cuya fórmula se asume que es $\prod_{j=1}^{J-1} P(\boldsymbol{x}^j|C)P(\boldsymbol{x}^{j+1}|\boldsymbol{x}^j, C)$, donde $P(\boldsymbol{x}^j|C)$ representa la probabilidad de que $\boldsymbol{X}$ se mantenga en el estado $\boldsymbol{x}^j$ durante el intervalo de tiempo $(t_{j-1}, t_j]$ dada la variable clase y $P(\boldsymbol{x}^{j+1}|\boldsymbol{x}^j, C)$ es la probabilidad de que $\boldsymbol{X}$ realice una transición desde el estado $\boldsymbol{x}^j$ al estado $\boldsymbol{x}^{j+1}$ dada la variable clase. Tras realizar una serie de modificaciones a esta fórmula, entre ellas la introducción de un término exponencial, se maximiza, pudiendo calcular una **solución exacta** debido al poco coste computacional que supone obtenerlo en los dos tipos de clasificadores comentados utilizando estadísticos suficientes para cada una de las variables. Para ello proponen, aparte de un algoritmo para su aprendizaje, un algoritmo de inferencia para dichos clasificadores.

La Tabla 3.1 muestra la comparativa de las diferentes propuestas de algoritmos de aprendizaje supervisado para flujos de datos basados en clasificadores bayesianos.

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
Incremental weighted naive Bayes (Salperwyck et al. (2015))	No			Sí		No	
WNB-CD (Krawczyk y Wozniak (2015))	Función de decaimiento de los pesos de las instancias			Sí			
ADWIN2 con naive Bayes (Bifet y Gavaldà (2007))	ADWIN2			Sí			
Borchani et al. (2011)	Divergencia Kullback-Leibler						
RGNBC (Kishore Babu et al. (2016))	Teoría de conjuntos aproximados	Sí*		Sí (no es seguro para categóricas)			
ACO (Roure (2002))	No		No	No		No*	No
CTNB y CTTANB (Stella y Amer (2012))	No		No	No	Sí*	No*	No

Tabla 3.1: Algoritmos de aprendizaje supervisado para flujos de datos basados en clasificadores Bayesianos

3.2.2. Árboles de decisión

Los **árboles de decisión** son uno de los algoritmos de aprendizaje automático más estudiados para clasificación de flujos de datos debido a su buen desempeño en los mismos, además de su simplicidad y de la interpretabilidad del modelo. Los desafíos que supone realizar tareas de clasificación en flujos de datos provocan que

los métodos de clasificación basados en árboles de decisión como el **ID3**, el **C4.5** y **CART** vistos con anterioridad no sean efectivos en el tratamiento de dicho tipo de datos. Estos métodos almacenan y procesan el conjunto de datos de entrenamiento enteros, pero en el caso de datos generados continuamente se necesita incrementar los requerimientos de procesamiento y solo se puede realizar hasta cierto punto, ya sea porque se sobrepasan los límites de memoria o porque el tiempo de computación es demasiado largo. Además, aunque estos métodos fueran capaces de manejar todas las instancias del flujo de datos, muchas de ellas puede que no sean útiles para la construcción del modelo debido a cambios en el proceso de generación de datos, y estos métodos no son capaces de manejar dichos cambios.

Dadas estas características, los métodos mencionados previamente reciben el nombre de algoritmos **no incrementales de inducción de árboles**. Existen otros algoritmos basados en árboles de decisión que son **incrementales**, que permiten actualizar un árbol existente utilizando solo instancias recientes, sin tener que volver a procesar instancias pasadas. Entre las primeras versiones incrementales de construcción de árboles de decisión se encuentran el **ID3'** (Schlimmer y Fisher (1986)), el **ID4** (Schlimmer y Fisher (1986)), el **ID5** (Utgoff (1988)), el **ID5R** (Utgoff (1989)) y el **ITI** (Utgoff et al. (1997)).

El **ID3'** es una variante incremental del algoritmo ID3 que se basa en un método de *fuerza bruta* para llevar a cabo la construcción del árbol de decisión, de manera que, cada vez que recibe una nueva instancia, vuelve a construir la estructura de árbol mediante el algoritmo ID3, por lo que, debido al costo computacional que conlleva, no es adecuado para la clasificación de flujos de datos. Por otra parte, el **ID4** es también una versión incremental del ID3, pero, a diferencia del ID3', no reconstruye el árbol cada vez que llega una nueva instancia, sino que lo actualiza. Para ello, almacena el número de instancias positivas y negativas para cada posible valor de cada posible atributo no testeado en cada nodo del árbol de decisión actual y actualiza dicho número con la llegada de nuevos ejemplos, lo que permite testear diferentes atributos en los diferentes nodos del árbol. El algoritmo ID4 no es eficiente puesto que descarta subárboles cada vez que se realiza un cambio del atributo de testeo en un determinado nodo, lo que provoca que el árbol no aprenda ciertos conceptos inherentes a los datos. Además, no garantiza que el árbol construido sea similar al que produce el algoritmo ID3.

Por otra parte, el **ID5**, a diferencia del ID4, no descarta subárboles a la hora de cambiar el nodo donde el atributo de test ya no es el mejor, sino que los actualiza empleando un proceso de reestructuración denominado *pull-up*. El objetivo de este proceso es promocionar el nuevo mejor atributo test en ese nodo hacia la parte superior del subárbol (cuya raíz es dicho nodo), de tal forma que el nuevo mejor atributo se asigne al nodo raíz del subárbol y se degrade al antiguo mejor atributo; no obstante, este algoritmo no garantiza que se obtenga el mismo árbol de decisión que se obtendría con el ID3. Un algoritmo que sí garantiza esto es el **ID5R**, que es una extensión del ID5 y que, a diferencia de este último, tras llevar a cabo la manipulación de la estructura mencionada en el ID5, realiza una reestructuración recursiva de los subárboles que están por debajo del nuevo mejor atributo utilizado

en la manipulación previa. Esta reestructuración recursiva se realiza con el objetivo de que cada uno de dichos subárboles tenga como raíz el mejor atributo test. Este algoritmo puede llegar a ser más lento que ID3 en algunos casos, dependiendo de las operaciones de reestructuración recursiva llevadas a cabo. Con respecto al algoritmo **ITI**, extiende el algoritmo ID5R de tal forma que es capaz de manejar atributos numéricos, ruido y valores faltantes, además de incorporar un mecanismo de poda; sin embargo, no es capaz de tratar con conjuntos de datos masivos y con flujos de datos.

En general los algoritmos incrementales basados en árboles de decisión comentados anteriormente no garantizan obtener el mismo árbol que se obtendría del *batch learning* y almacenan todos los ejemplos utilizados en memoria. No obstante, los problemas de clasificación de flujos de datos requieren que se cumplan una serie de restricciones de recursos computacionales, entre ellas la memoria. De esta manera, el primer algoritmo que se propuso específicamente para resolver los problemas que plantea la clasificación de flujos de datos, entre ellas la memoria limitada, se denomina **very fast decision tree** (VFDT), propuesto en Domingos y Hulten (2000).

Este algoritmo construye un árbol de decisión de forma *online* utilizando una propiedad estadística denominada **Hoeffding bound** (HB), de manera que el árbol de decisión que produce este algoritmo se denomina *Hoeffding tree*. La idea sobre la que se basa este algoritmo es que, a la hora de establecer el mejor atributo a testear en cada uno de los nodos, es suficiente con tener en cuenta un subconjunto de instancias de entrenamiento que pasan por ese nodo. Para averiguar el número de ejemplos necesarios para lograr un determinado nivel de confianza acerca de que el atributo elegido con un subconjunto de ejemplos es el mismo que si escogiéramos el atributo con un número infinito de instancias, se utiliza el *Hoeffding bound*. Esta propiedad estadística establece que si la diferencia entre la métrica de evaluación del mejor atributo (ganancia de información o índice de Gini) teniendo en cuenta un subconjunto de instancias de entrenamiento y aquella del segundo mejor atributo es mayor que un valor determinado por el *Hoeffding bound*, entonces **se garantiza con una determinada probabilidad que ese mejor atributo es la elección correcta**.

Esta propuesta no almacena datos en memoria, sino que solo mantiene una serie de estadísticas que son suficientes para calcular la métrica de evaluación de cada uno de los atributos (las estadísticas se mantienen en las hojas, de tal forma que el árbol de decisión se construye recursivamente sustituyendo hojas por nodos de decisión). Además, el *Hoeffding bound* no se calcula cada vez que llega una nueva instancia, sino que se establece un umbral mínimo de instancias a obtener definido por el usuario puesto que una sola instancia tiene poca repercusión en los resultados. Además, cuando la diferencia entre las métricas de evaluación de dos atributos es muy pequeña, en lugar de esperar a tener un mayor número de instancias para asegurar cuál de ellos es el mejor y cuál el segundo mejor puesto que no implica mucha diferencia entre elegir uno u otro, el algoritmo VFDT permite que el usuario defina un parámetro de ruptura del empate, de tal forma que si la diferencia es menor que ese parámetro, entonces se elige como mejor atributo aquél que lo es en

ese momento. Todo esto, aparte de lo mencionado anteriormente sobre el algoritmo VFDT, permite a éste obtener un árbol de decisión **parecido a los producidos por un algoritmo de aprendizaje que tiene en cuenta todos los ejemplos de entrenamiento para elegir un atributo a testear para cada uno de los nodos del árbol** utilizando una cantidad de memoria y tiempo constante por cada uno de los ejemplos de entrenamiento.

Por otra parte, en Gama et al. (2003) se propone el **algoritmo VFDTc**, que se basa en el algoritmo VFDT y se extiende el mismo incorporando la capacidad de lidiar con atributos continuos y sustituyendo las hojas por un modelo local de predicción, que es el *naive Bayes*, en lugar de utilizar la tradicional clasificación de una instancia en árboles de decisión a la clase más frecuente en una determinada hoja. Para llevar a cabo un testeo de un atributo continuo, puede haber muchas posibilidades, puesto que se trata de buscar el mejor valor que particione el conjunto de datos en instancias cuyo valor en ese atributo sea menor que el establecido en el nodo de decisión y en aquellas cuyo valor sea mayor. Para encontrar el mejor valor de un atributo continuo para particionar el conjunto de datos en un nodo hoja y convertirlo a un nodo decisión cuando haya un nivel de confianza determinado, para cada hoja y atributo continuo se construye un **árbol binario** con el objetivo de almacenar una serie de estadísticas y, a partir de ellas, calcular la distribución de las clases de las instancias en los que el valor de la variable predictiva continua es menor o mayor que el valor escogido para particionar el conjunto de datos. En cuanto a la tarea de predicción del árbol de decisión, para mejorar su desempeño de clasificación en las hojas se insertan clasificadores *naive Bayes* puesto que estos modelos locales funcionan de forma aceptable en el aprendizaje incremental, además de que este modelo tiene en cuenta no solo la distribución a priori de las clases como ocurre en la clasificación de la instancia a la clase mayoritaria en la hoja, sino que además tiene en cuenta **información sobre los valores de los atributos**, concretamente las probabilidades condicionales de los mismos dadas las diferentes clases.

A la hora de establecer el valor del atributo numérico que mejor particiona el conjunto de datos en un determinado nodo de decisión, puede ocurrir que el número de posibles valores para realizar dicha partición sea muy grande, lo que puede conllevar gastos computacionales altos. De esta manera, en Jin y Agrawal (2003), basado en el algoritmo VFDT, se plantea una **poda del árbol en intervalos numéricos** (*Numerical Interval Pruning*, NIP) para reducir el tiempo de procesamiento sin perder precisión a la hora de encontrar el valor de un atributo continuo que particione el conjunto de datos en un nodo de decisión. En concreto, la idea sobre la que se fundamentan es particionar el rango de valores de un atributo continuo en intervalos con la misma amplitud y utilizar pruebas estadísticas para podarlos, de tal forma que se poda un intervalo si es probable que el valor utilizado para particionar el conjunto de instancias **no se encuentre en ese intervalo**, por lo que se reduce el número de posibles valores para llevar a cabo la partición. Por otra parte, otra mejora que proponen es utilizar unas propiedades de las métricas de evaluación de atributos (ganancia de información o índice de Gini) con el fin de obtener la misma cota que la *Hoeffding bound*, pero con un **número de instancias menor**. Para

ello, se basan en el método denominado ***multivariate delta***, que se fundamenta en la idea de que la diferencia entre los valores de ganancia de información (o del índice de Gini) es una **variable aleatoria normal** y calculan las cotas adecuadas utilizando un **test de la distribución normal**.

El algoritmo VFDT tiene la desventaja de que **no maneja el *concept drift***, por lo que, aún teniendo disponible todo el conjunto de datos de entrenamiento para la construcción del árbol, el árbol que construye puede que no sea útil para describir las instancias que lleguen en el futuro debido a cambios en la distribución de probabilidad subyacente a los mismos. De esta manera, en Hulten et al. (2001) se propone el algoritmo **CVFDT**, que extiende el algoritmo VFDT con la capacidad de manejar el *concept drift*, de manera que mantiene un árbol de decisión actualizado aplicando el algoritmo VFDT sobre una **ventana deslizante de instancias de entrenamiento** y construyendo **subárboles alternativos**.

El algoritmo CVFDT utiliza una ventana deslizante fija de ejemplos de entrenamiento para actualizar las estadísticas presentes en **todos los nodos del árbol de decisión** (a diferencia del VFDT, que mantenía estadísticas solo en las hojas para elegir el atributo a testear), de tal forma que incrementa los conteos de las nuevas instancias y decrementa los conteos relacionados con los ejemplos antiguos con el objetivo de **eliminar su influencia en la construcción del árbol**. De esta manera, al cambiar los valores de las estadísticas de cada nodo, puede ocurrir que los atributos que se testean en determinados nodos no sean los mejores. En este sentido, el algoritmo CVFDT comienza a construir **subárboles alternativos** en dichos nodos y, cuando estos subárboles tienen un mejor rendimiento que los actuales, **se reemplazan los actuales por los alternativos**.

Otra propuesta para la clasificación de flujos de datos basada en árboles decisión es la denominada **UFFT** (*Ultra Fast Forest of Trees*), planteada en Gama y Medas (2005). Para problemas multiclase, este algoritmo construye un **bosque de árboles de decisión binarios**, uno para cada posible par de valores de la variable clase (siendo un solo árbol de decisión binario cuando la variable clase solo toma dos valores). A la hora de clasificar una nueva instancia, se proporciona la misma a cada uno de los árboles de decision binarios y la predicción que realizan son **distribuciones de probabilidad de las diferentes clases**, que posteriormente se agregan y se obtiene la clase más probable a la que pertenece la instancia. En cada uno de estos árboles de decisión binarios, para llevar a cabo la clasificación de instancias en las hojas se utilizan **clasificadores *naive Bayes***, además de en los nodos de decisión. Con respecto a los nodos de decisión, por una parte se emplean para detectar cambios en las distribuciones de las clases de las instancias que atraviesan dichos nodos; de esta manera, si el error del clasificador incrementa, entonces la distribución subyacente a los datos ha cambiado, por lo que se **realiza una poda del subárbol que cuelga de ese nodo de decisión** y se **aprende dicho cambio** a partir de un **conjunto de las instancias más recientes** (*short term memory*). Por otro lado, sus predicciones se utilizan para establecer **pruebas para realizar una partición del conjunto de datos** en el caso de que las ganancias de información de los dos mejores atributos no satisfagan el *Hoeffding bound*, de tal forma que un nodo se

expandirá o no en función de si la predicción del clasificador *naive Bayes* es precisa o no.

El algoritmo CVFDT no es suficientemente sensible a la ocurrencia de *concept drifts* puesto que los detecta tras obtener un determinado número de instancias que indiquen que existe un cambio notable en la precisión de un subárbol con el objetivo de cambiarlo por otro subárbol. En general, los algoritmos que comprueban la presencia de *concept drifts* a nivel de instancias o de atributos (como el CVFDT) no presentan una sensibilidad notable frente a dichos cambios. En este sentido, el algoritmo CVFDT no es capaz de detectar un tipo de *concept drift* denominado *concept shift*, que consiste en que dos bloques de datos consecutivos tienen distribuciones opuestas (en un bloque las clases están separadas por un hiperplano y en el siguiente bloque las distribuciones de las clases se encuentran en lados opuestos con respecto al anterior bloque, manteniendo el mismo hiperplano, por ejemplo), pero el valor de la ganancia de información que el algoritmo CVFDT utiliza para detectar el *concept drift* es el mismo en ambos bloques de datos. Por eso, en Tsai et al. (2008) proponen el algoritmo **SCRIPT** (*Sensitive Concept Drift Probing Decision Tree*), que se basa en utilizar la prueba estadística χ^2 para tratar el *concept drift*, una medida para comprobar, en este caso, que las distribuciones de las clases con respecto al valor de un atributo son **similares en dos bloques de datos consecutivos**, de tal forma que el algoritmo SCRIPT lleva a cabo la detección de *concept drifts* a un nivel de detalle mayor que el algoritmo CVFDT. En el caso de que las diferencias entre las distribuciones de las clases teniendo en cuenta el valor de un atributo supere un umbral, se procede a realizar cambios en los subárboles pertinentes de la estructura.

Por otro lado, en Li y Liu (2008) proponen el algoritmo **EVFDT** (*Efficient-VFDT*), que extiende el algoritmo VFDT de dos formas. En primer lugar, para tratar atributos numéricos proponen el método *UINP* (*Uneven Interval Numerical Pruning*), que extiende el propuesto en Jin y Agrawal (2003), de manera que, en lugar de utilizar intervalos de la misma anchura, optan por definir **intervalos continuos de diferente amplitud** con el fin de ganar eficiencia. En segundo lugar, como en Gama y Medas (2005), utilizan clasificadores *naive Bayes* tanto en los nodos de decisión como en las hojas con el fin de mejorar la eficiencia de la construcción del árbol de decisión y hacer que la estructura del mismo sea más compacta descartando instancias que no son útiles para la construcción del árbol de decisión.

Otra propuesta que se fundamenta en el *Hoeffding Tree* del algoritmo VFDT es la planteada en Bifet y Gavaldà (2009), donde se proponen dos métodos para manejar la naturaleza cambiante de los flujos de datos: **Hoeffding Window Tree** (HWT) y **Hoeffding Adaptive Tree** (HAT). El algoritmo HWT se basa en utilizar un **modelo de ventanas deslizante** para manejar el *concept drift* y, para implementarlo, emplea el algoritmo **ADWIN** (propuesto en Bifet y Gavaldà (2007), comentado en la sección 3.2.1), cuyo objetivo es detectar cambios en la distribución subyacente de los datos de forma continua (HWT-ADWIN) utilizando una **ventana de instancias de tamaño variable**. Este algoritmo se diferencia del algoritmo CVFDT en que la construcción de los subárboles alternativos se realiza tan pronto como se detecte un *concept drift*, y su inserción en la estructura se lleva a cabo tan pronto

como los subárboles alternativos tengan un mejor desempeño que los actuales, ambas acciones **sin tener que esperar a que llegue un número determinado de instancias**. Además, a diferencia del CVFDT, **no es necesario que el usuario defina un tamaño de ventana**, puesto que se adapta al ritmo del cambio de la distribución de los datos, y tiene **garantías teóricas en cuanto a su desempeño**, mientras que el algoritmo CVFDT no las tiene. En cuanto al método HAT, se basa en el algoritmo HWT pero, en lugar de tener un tamaño de ventana óptimo y único para todos los nodos, establece un **detector de cambio en cada uno de los nodos** (en lugar de contadores), de manera que se mantiene un tamaño de ventana óptimo para cada uno de ellos.

Por otra parte, las propuestas que utilizan clasificadores *naive Bayes* en las hojas obtienen buenos resultados, pero algunas veces a costa de incrementar el tiempo de ejecución de los algoritmos. Por ello, en Bifet et al. (2010) se propone el algoritmo **Hoeffding Perceptron Tree**, que se fundamenta en la utilización de perceptrones en las hojas del árbol de decisión para llevar a cabo tareas de clasificación, de tal forma que reducen el tiempo de ejecución, al mismo tiempo que se mantiene un buen desempeño del árbol. Esta reducción del tiempo de ejecución se produce porque no necesita estimar la distribución estadística de los atributos numéricos y calcular los valores de densidad basados en la función exponencial, y para los atributos discretos no necesita calcular las divisiones para estimar las probabilidades. Esta propuesta combina las ventajas de los árboles de decisión y de los perceptrones, lo que permite llevar a cabo un procesamiento eficiente de los flujos de datos. Además, contemplan la utilización de tres clasificadores para mejorar aún más la precisión del árbol de decisión, de manera que se combinan sus predicciones mediante votación. Estos clasificadores son el perceptrón, el *naive Bayes* y el voto por mayoría. No obstante, la combinación de estos clasificadores hace que se ralentice el algoritmo.

Existen muchos otros trabajos que desarrollan algoritmos de aprendizaje automático para flujos de datos basados en árboles de decisión, como son los planteados en Rutkowski et al. (2013), Yang y Fong (2013), Rutkowski et al. (2014) y da Costa et al. (2018). En Rutkowski et al. (2013) proponen, a diferencia de aquellas propuestas que se basan en el *Hoeffding bound*, utilizar la **desigualdad de *McDiarmid*** para elegir el mejor atributo para llevar a cabo la división de un nodo del árbol, puesto que establecen que el *Hoeffding bound* no es apropiado para resolver este problema. De esta manera, sustituyen el algoritmo de construcción del árbol planteado en Domingos y Hulten (2000) por la desigualdad de *McDiarmid*, de esta manera denominando al algoritmo ***McDiarmid Tree***. Con respecto al trabajo propuesto en Yang y Fong (2013), desarrollan el algoritmo denominado ***Optimized Very Fast Decision Tree*** (*OVFDT*), que propone, en lugar de un parámetro de desempate fijo definido por el usuario como ocurre en Domingos y Hulten (2000), uno **adaptativo**, de tal forma que se optimiza el número de divisiones de nodos realizados a medida que se lleva a cabo la construcción incremental del árbol con el objetivo de obtener un equilibrio entre la precisión del modelo y su tamaño.

En Rutkowski et al. (2014) proponen el algoritmo denominado ***dsCART***, una adaptación del algoritmo CART para flujos de datos. Para tomar la decisión de

cuál es el mejor atributo para dividir un determinado nodo utilizando una muestra de los datos, proponen un nuevo método basado en el **teorema de Taylor** y las **propiedades de la distribución normal**. Así demuestran que el mejor atributo que eligen utilizando un subconjunto de los datos es el mismo que si se tuvieran en cuenta todos los datos. Por otra parte, en da Costa et al. (2018) desarrollan el algoritmo ***Strict VFDT*** (*SVFDT*), basado en el algoritmo *VFDT*, con el objetivo de controlar el tamaño del árbol sin degradar el desempeño del mismo. Concretamente, proponen dos versiones, *SVFDT-I* y *SVFDT-II*, donde el objetivo del primero es consumir menos memoria y tiempo de entrenamiento y el del segundo es obtener un rendimiento alto.

La Tabla 3.2 muestra la comparativa de las diferentes propuestas de algoritmos de aprendizaje supervisado para flujos de datos basados en árboles de decisión.

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Modelo local en las hojas	Manejo de la aparición de nuevos atributos
ID3' (Schlimmer y Fisher (1986))	No						No	
ID4 (Schlimmer y Fisher (1986))	No						No	
ID5 (Utgoff (1988))	No		No	No		No	No	
ID5R (Utgoff (1989))	No		No	No		No	No	
ITI (Utgoff et al. (1997))	No		Sí	Sí		Sí	No	
VFDT (Domingos y Hulten (2000))	No			No	Sí	Sí*	No	
CVFDT (Hulten et al. (2001))	Sliding window				Sí	Sí*	No	
VFDTc (Gama et al. (2003))	No			Sí	Sí	Sí*	Naive Bayes	
Jin y Agrawal (2003)	No			Sí		Sí*	No	
UFFT (Gama y Medas (2005))	Naive Bayes en los nodos de decisión			Sí (no es seguro para categóricas)			Naive Bayes	
SCRIPT (Tsai et al. (2008))	Estadístico χ^2					Sí	No	
EVFDT (Li y Liu (2008))	No			Sí			Naive Bayes	
HWT y HAT (Bifet y Gavaldà (2009))	ADWIN (principalmente)		Sí*	Sí			Con y sin Naive Bayes	
Hoeffding Perceptron Tree (Bifet et al. (2010))	Sí			Sí			Perceptrón o combinación de perceptrón, Naive Bayes y voto por mayoría	
McDiarmid Tree (Rutkowski et al. (2013))	No			No	Sí	Sí*	No	
OVFDT (Yang y Fong (2013))	No		No*	No	Sí	Sí*	Clasificador *Naive Bayes*	No*
dsCART (Rutkowski et al. (2014))	Ventana deslizante	No*	No*	Sí	Sí	Sí*	No	No*
SVFDT (da Costa et al. (2018))	No			Sí*	Sí	Sí*	No	

Tabla 3.2: Algoritmos de aprendizaje supervisado para flujos de datos basados en árboles de decisión

3.2.3. Inducción de reglas

Los métodos basados en inducción de reglas proporcionan una buena interpretabilidad y flexibilidad para las tareas de aprendizaje automático. La ventaja principal de estos métodos a la hora de adaptarlos a la clasificación de datos dinámicos es que las reglas son **más fáciles de ser adaptadas**. En lugar de reconstruir un nuevo clasificador desde cero al ocurrir cambios en la distribución subyacente a los datos, las reglas que se consideran obsoletas, de forma individual, simplemente **se pueden eliminar del conjunto de reglas**, lo que puede dar lugar a mecanismos de adaptación rápidos. Además, los conjuntos de reglas **no están estructurados de forma jerárquica** como ocurre con los árboles de decisión, lo que favorece una actualización de las mismas sin repercutir mucho en la eficiencia del aprendizaje.

Uno de las primeras propuestas de inducción de reglas para flujos de datos es la que se plantea en Ferrer-Troyano et al. (2004), donde se propone el algoritmo denominado **SCALLOP** (*Scalable Classification ALgorithm by Learning decisiOn Patterns*), que se fundamenta en construir un conjunto reducido de reglas de clasificación actualizadas y ordenadas para cada una de las clases del problema a partir de flujos de datos numéricos en función de una serie de parámetros establecidos por el usuario. Este algoritmo modela solo **aquellas regiones que describan las características en las que está interesado el usuario**, que las define como parámetros de entrada del algoritmo. Cada una de las reglas de clasificación que construye el algoritmo está formada por un conjunto de n intervalos cerrados (uno por cada atributo), de manera que define un **hipercubo** dentro del espacio de búsqueda. Cada uno de estos hipercubos se puede expandir para cubrir nuevos ejemplos hasta un cierto límite puesto que podría ocurrir que las regiones que cubren dichas reglas, que tienen asociada una clase a la que clasificar, se solapen.

Por otra parte, el algoritmo lleva a cabo un **refinamiento de las reglas** cada cierto número de instancias procesadas (definido por el usuario) uniendo reglas relacionadas con la misma clase cuya región resultante de dicha unión sea la **más pequeña** en comparación con otras posibles uniones (las dos reglas más cercanas) en cada iteración del procedimiento de refinamiento. Tras esto, se comprueba que el volumen de la región que se produce tras la unión no intersecta con otras regiones que tienen asociadas otras etiquetas y que se encuentra dentro de unos límites. El algoritmo también lleva a cabo **eliminaciones de reglas** que no interesan al usuario o que están afectadas por ruido; una regla es eliminada si no cubre al menos una instancia de un conjunto determinado de instancias recientes o si su soporte positivo (número de ejemplos que tienen la misma etiqueta que la regla y que son cubiertas por ésta) es menor que un número definido por el usuario. A la hora de clasificar una instancia, si existe una regla que la cubre (en el caso de varias reglas se escoge la primera que la cubre), se le **asigna la etiqueta en la que clasifica dicha regla**; en otro caso, el algoritmo infiere a **qué clases no puede pertenecer la instancia** y la predicción se realiza mediante **votación** entre aquellas reglas que no clasifican a dichas clases y cuya región de decisión se puede expandir para cubrir a la instancia sin solaparse con las regiones de decisión de reglas que clasifican a otras clases distintas a la de dicha regla. En caso de empate entre clases, se decide

atendiendo a la distribución de las mismas (sus frecuencias).

Por otra parte, en Ferrer-Troyano et al. (2005) los mismos autores de la propuesta SCALLOP proponen el algoritmo denominado **FACIL**, que permite obtener un conjunto de reglas de clasificación a partir de flujos de datos numéricos (en Ferrer-Troyano et al. (2006) se extiende dicha propuesta para atributos categóricos). El modelo de decisión que construye el algoritmo se describe no solo con un conjunto de reglas, sino también con un **número de instancias de entrenamiento**. En esta propuesta extienden el trabajo realizado en Ferrer-Troyano et al. (2004) almacenando por cada una de las reglas de clasificación un número de ejemplos positivos y negativos que se encuentran **cerca en los límites de las regiones de decisión** definidos por las reglas con el objetivo de evitar revisiones innecesarias cuando se den *virtual drifts*. En concreto, estos ejemplos se utilizan para comprobar si las reglas son **inconsistentes**, de tal forma que se van captando instancias de las fronteras de las regiones de decisión hasta que se llega a un umbral (definido por el usuario), que corresponde a la **mínima pureza de una regla** (la pureza de una regla es el ratio entre el número de instancias positivas almacenadas que cubre y el número total de instancias almacenadas que cubre, tanto positivas como negativas). Si la pureza de la regla alcanza ese umbral, ésta se bloquea, asi como sus posibilidades de ser generalizada a nuevas instancias, y los ejemplos almacenados se utilizan para construir **nuevas reglas positivas y negativas consistentes**.

En relación a lo anterior, cada vez que llega un nuevo ejemplo, se actualiza el modelo. Para ello, primero se comprueban aquellas reglas que **cubren la instancia** y que **clasifiquen a la misma clase a la que pertenece la misma**; si existen, se incrementa su soporte positivo. En el caso de que no haya ninguna regla que cumpla estas características, se **examina el grado de crecimiento** que deben realizar para cubrir la nueva instancia cada una de las reglas que clasifiquen a la misma clase a la que pertenece la nueva instancia, de tal forma que se elige como regla **candidata** aquella que necesite menor número de cambios para cubrir el ejemplo; el crecimiento de la regla, para poder ser elegida candidata, debe estar por debajo de un umbral determinado. En el caso de que exista una regla candidata, si no intersecciona con ninguna otra regla que tenga una clase diferente a la suya, se utiliza para cubrir la nueva instancia; en caso contrario, se **revisan aquellas reglas que no tengan asociada la misma etiqueta que el ejemplo** y que lo cubren, incrementando de esta manera su soporte negativo y realizando la comprobación de la pureza de la regla tras la adición del nuevo ejemplo. Si no existe ninguna regla que pueda cubrir al nuevo ejemplo, entonces se construye una regla de **máxima especificidad** que la cubra.

En el algoritmo se incorpora un **mecanismo de olvido** de instancias en cada una de las reglas para adaptarse a posibles *concept drifts* (detección de *concept drift* ***blind***), de tal forma que éste puede ser explícito si los ejemplos son más antiguos que un umbral definido por el usuario, o implícito si se eliminan instancias positivas y negativas cuando ya *no se encuentran cercanas las positivas de las negativas* (no influyen en la descripción del concepto en los límites de las regiones de decisión). A la hora de clasificar se utilizan las reglas que la cubren; si son consistentes, se les

asigna la clase en la que clasifican dichas reglas, y si son inconsistentes llevan a cabo la clasificación utilizando una medida de distancia entre los ejemplos almacenados en esas reglas y el nuevo ejemplo, como en el algoritmo vecinos más cercanos; la clasificación final se realiza mediante **votación**. Si el nuevo ejemplo no es cubierto por ninguna regla, se clasifica en la clase a la que pertenezca la regla cuyo crecimiento para cubrir dicha instancia sea mínima y cuya región de decisión no intersecte con la de otras reglas de diferente etiqueta.

Otra propuesta para inducir reglas de clasificación a partir de flujos de datos es la planteada en Gama y Kosina (2011), donde se propone el algoritmo denominado **Very Fast Decision Rules** (VFDR), cuya idea fundamental es aprender conjuntos de reglas de clasificación tanto ordenados como desordenados. En el aprendizaje de conjuntos ordenados de reglas, cada una de las instancias de entrenamiento de entrada se utiliza para actualizar las estadísticas de la **primera regla que la cubre**, mientras que en el aprendizaje de conjuntos desordenados de reglas cada una de las instancias actualiza las estadísticas de **todas las reglas que la cubren**; si el ejemplo no es cubierto por ninguna regla, se actualizan las estadísticas de la regla por defecto, que se utiliza para crear nuevas reglas (no tiene antecedentes y en el consecuente almacena una serie de estadísticas). Para llevar a cabo la construcción de las reglas, se utiliza el *Hoeffding bound* descrito en Hulten et al. (2001), de tal forma que esta métrica define el número de instancias necesarias para inducir o construir una regla con una determinada confianza; la comprobación de este número se realiza tras procesar un número mínimo de instancias. A la hora de realizar la expansión de una regla, se calcula la **entropía de cada uno de los valores de las variables** que aparecen en más de un 10 % de los ejemplos asociados a dicha regla. De esta manera, si la entropía (ecuación 2.9) del mejor valor del mejor atributo es menor que la entropía de no llevar a cabo ninguna expansión con una determinada diferencia establecida por el *Hoeffding bound*, entonces la regla **se expande añadiendo la condición de que ese atributo tiene que tener ese valor**; la clase en la que clasifica la regla es la clase mayoritaria entre los ejemplos que tiene asignados. A diferencia de las propuestas anteriores, no detecta el *concept drift* y, para llevar a cabo la tarea de clasificación, utiliza el clasificador *naive Bayes* en cada una de las reglas de clasificación del modelo. En el caso de que se haya aprendido un conjunto *ordenado* de reglas, para clasificar una nueva instancia se utiliza la **primera regla que la cubra** y, en el caso de un conjunto desordenado de reglas, se utilizan **todas las reglas que la cubren** y se lleva a cabo un **voto ponderado** de los resultados de cada una de las reglas.

Por otra parte, los autores de la propuesta anterior proponen en Kosina y Gama (2012a) una extensión del trabajo realizado en Gama y Kosina (2011) para lidiar con un problema multiclase **descomponiéndolo en un conjunto de problemas de dos clases**, de tal forma que se construye un conjunto de reglas para cada uno de estos problemas con el objetivo de obtener mejores resultados; el algoritmo que plantean lo denominan **VFDR-MC** (VFDR *multi-class*). Concretamente, el algoritmo VFDR-MC se diferencia de VFDR en que aplica la estrategia ***one vs. all***, de manera que las instancias que pertenezcan a una determinada clase se consideran

positivas, mientras que aquellas que pertenecen a una clase distinta se consideran negativas. A la hora de llevar a cabo el aprendizaje de reglas, al igual que en VFDR, tienen en cuenta el caso de conjuntos de reglas ordenadas ($VFDR - MC^o$) y desordenadas ($VFDR - MC^u$). La función de ganancia de información que se utiliza para evaluar la importancia de los diferentes atributos tiene en cuenta el número de instancias **positivas**, descritas anteriormente.

Si se parte de la regla por defecto para aprender una nueva regla, en el caso de reglas ordenadas, el nuevo literal de dicha regla es aquél que tiene la **mejor evaluación atributo-clase** y la clase positiva es aquella que cumple el *Hoeffding bound* y que tiene menor frecuencia (interesa crear una nueva regla para la clase minoritaria); en el caso de reglas desordenadas, se comprueban las expansiones de la regla por defecto para todas las posibles clases. En el caso de que se vaya a expandir una regla con alguna condición en el antecedente, si se aprende un conjunto de reglas ordenadas, la clase a la que clasifica dicha regla se **mantiene como positiva** y se busca el mejor par atributo-valor con respecto a esa clase para expandirla si cumple el criterio establecido por el *Hoeffding bound*. En el caso de reglas desordenadas, si se lleva a cabo la expansión de la regla anterior (con la clase positiva original), se **tienen en cuenta las demás clases** como **positivas** y, para cada una de ellas, se encuentra el mejor par atributo-valor para expandir la regla, de tal forma que **se crean varias reglas** a partir de dicha regla estableciendo diferentes clases como la clase positiva en cada una de ellas. La tarea de clasificación se lleva a cabo de la misma manera que en el algoritmo VFDR.

Los trabajos realizados en Gama y Kosina (2011) y Kosina y Gama (2012a) no tratan explícitamente la adaptación de los modelos construidos a *concept drifts*. Por eso, en Kosina y Gama (2012b) se realiza una extensión del algoritmo *VFDR* con el objetivo de añadirle la capacidad de tratar con datos cuyo concepto subyacente cambia con el tiempo, denominado **AVFDR** (*Adaptive* VFDR). Para ello, a cada una de las reglas del modelo se le incorpora un **mecanismo explícito de detección** de *concept drifts*, de tal forma que cada una de ellas lleva un control de su desempeño a través de métricas de evaluación para detectarlos. El método que se aplica en cada una de las reglas para detectar *concept drifts* se denomina **SPC** (*Statistical Process Control*). La idea en la que se basa consiste en calcular el error de predicción de cada una de las instancias cubiertas por la regla, de tal forma que dicho error se va actualizando con cada uno de los ejemplos. Cada vez que se realiza una actualización de dicho error, se comprueba el **estado del proceso de aprendizaje** en el que se encuentra la regla. Se consideran tres posibles estados: controlado, en alerta o fuera de control. Para averiguar dicho estado, se utilizan el porcentaje de errores (p) y su desviación estándar (s), así como unos valores mínimos de esas medidas (p_{min} y s_{min}) y una serie de ponderaciones. Si la regla se encuentra en estado de alerta, entonces el aprendizaje de la misma se para hasta que se encuentre en estado controlado; si se encuentra en estado de fuera de control, **se elimina del conjunto de reglas** puesto que el nivel de degradación de su desempeño es tal que puede afectar negativamente de forma significativa al rendimiento del modelo. Todo esto permite realizar un **podado de las reglas** para que no crezcan excesivamente.

Otra propuesta que trata la inducción de reglas a partir de flujos de datos es la planteada en Deckert y Stefanowski (2014), donde proponen el algoritmo denominado ***Rule-based IncrementaL Learner*** (RILL). Una de las diferencias más notorias de este algoritmo con respecto al algoritmo VFDR es que el algoritmo RILL comienza con reglas específicas y luego lleva a cabo una generalización de las reglas (método *bottom-up*), mientras que en el algoritmo VFDR se empieza con reglas generales y se van haciendo más específicas. Además, en el algoritmo VFDR se puede realizar el aprendizaje tanto de conjuntos de reglas ordenadas como desordenadas, mientras que en el algoritmo RILL solo se lleva a cabo el aprendizaje de conjuntos de reglas desordenadas. Con respecto al algoritmo FACIL, éste se asemeja al algoritmo RILL puesto que también utiliza una aproximación *bottom-up* para la inducción de reglas, pero el algoritmo FACIL utiliza una ventana de instancias para cada una de las reglas, mientras que en el algoritmo RILL emplean una ventana global, además de que en el algoritmo FACIL el podado de reglas se realiza de forma poco eficiente.

En el algoritmo RILL, cada vez que llega una nueva instancia, se añade la misma a la ventana global y se actualiza la distribución de las clases de la ventana. Tras esto, por cada una de las reglas que cubren a la instancia, tanto si la clase en la que clasifican las reglas es la misma o no a la de la instancia, se actualizan las estadísticas correspondientes de las mismas. Si no existe ninguna regla que prediga la misma clase a la que pertenece la instancia y que la cubra, se lleva a cabo un **proceso de generalización de la regla**, que consiste en buscar la regla más cercana a la instancia y generalizarla para que cubra a dicha instancia eliminando condiciones de atributos categóricos o ampliando las condiciones de los atributos numéricos. Si la generalización no se lleva a cabo debido a que la longitud de la regla (número de antecedentes) que generaliza no es mayor que cero o si la pureza de dicha regla ($\frac{soporte\ positivo}{soporte\ positivo+soporte\ negativo}$) con respecto a la ventana global es menor que la frecuencia relativa de la clase de la regla en relación con dicha ventana, entonces se añade al conjunto de reglas una regla específica que modele dicha instancia. Si la ventana (de tamaño fijo) se llena, se elimina el ejemplo menos reciente de la misma y, para realizar un podado del conjunto de reglas, se eliminan aquellas reglas que **no se hayan utilizado en un determinado periodo de tiempo**, que **tengan una pureza baja** o que **cometan muchos errores de predicción**. Para llevar a cabo la clasificación de una instancia, se elige **aquella regla más cercana a la misma** y **se le asigna su clase**, a diferencia del algoritmo FACIL, que realiza una votación de todas las reglas que la cubren.

La Tabla 3.3 muestra la comparativa de las diferentes propuestas de algoritmos de aprendizaje supervisado para flujos de datos basados en inducción de reglas.

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
SCALLOP (Ferrer-Troyano et al. (2004))	Uniendo reglas cercanas y eliminando reglas anticuadas (cada cierto número de ejemplos nuevos)	Sí*	No*	Sí (no es seguro para categóricas)	No	Sí	
FACIL (Ferrer-Troyano et al. (2005))	Pureza de las reglas y mecanismo de olvido (umbral de tiempo e instancias innecesarias)	Sí*	No*	Sí (no categóricas)	No*	Sí	
FACIL (Ferrer-Troyano et al. (2006))	Pureza de las reglas y mecanismo de olvido (umbral de tiempo e instancias innecesarias)	Sí*	No*	Sí	No*	Sí	
VFDR (Gama y Kosina (2011))	No	Sí*	No*	Sí		Sí*	
VFDR-MC (Kosina y Gama (2012a))	No	Sí*	No*	Sí		Sí*	
AVFDR (Kosina y Gama (2012b))	Algoritmo SPC	Sí*	No*	Sí		Sí*	
RILL (Deckert y Stefanowski (2014))	Podado de reglas y ventana deslizante	Sí*	No*	Sí	No*	Sí*	

Tabla 3.3: Algoritmos de aprendizaje supervisado para flujos de datos basados en inducción de reglas

3.2.4. Redes neuronales

Uno de los algoritmos basados en redes neuronales más conocidos para clasificar flujos de datos es el denominado **eGNN** (*evolving Granular Neural Network*), propuesto en Leite et al. (2009). Este algoritmo se fundamenta en dos pasos: **granular la información numérica de entrada** construyendo conjuntos borrosos y **construir la red neuronal a partir de la información granulada**. La red neuronal recibe como entrada una instancia h, descrita por un conjunto de atributos cuyos valores se denotan con $x_i^{[h]}$, donde i es el i-ésimo atributo. La red que se construye se muestra en la Figura 3.7.

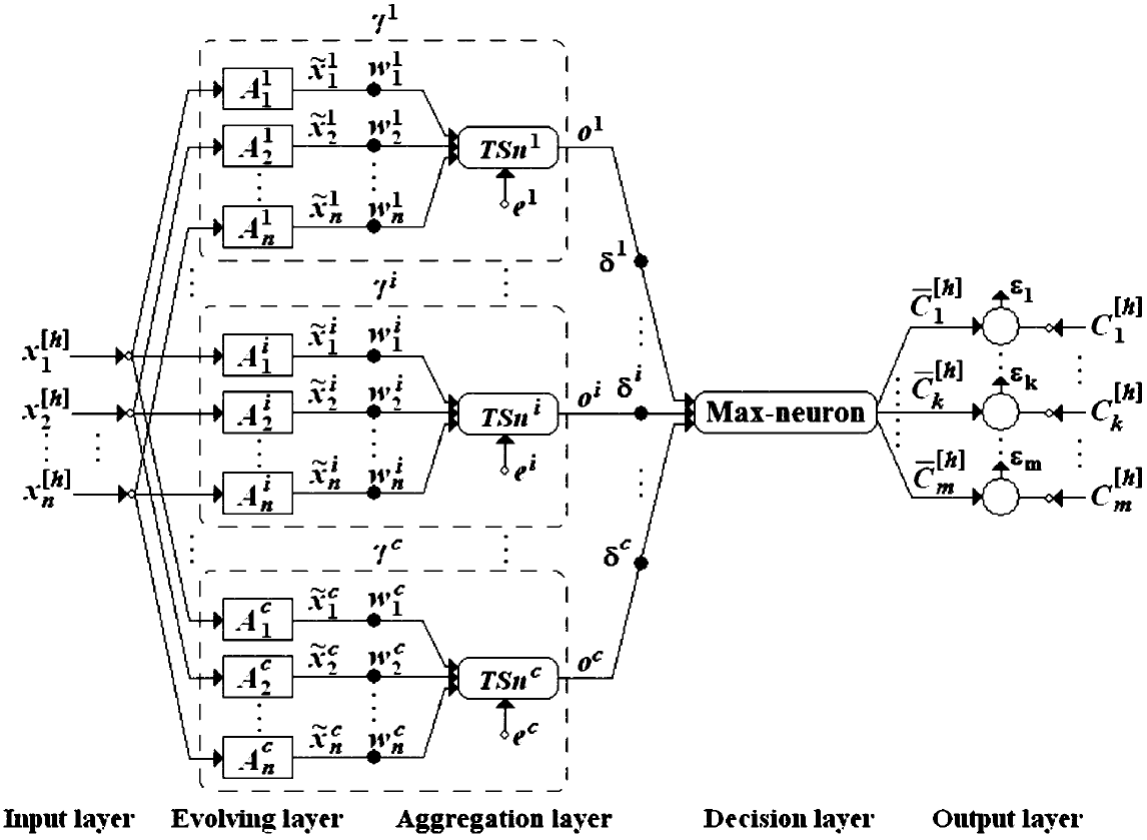

Figura 3.7: Estructura de una red neuronal granular que evoluciona. Fuente: Leite et al. (2009)

Los gránulos obtenidos del primer paso, que tienen asociada una determinada clase, están definidos por **funciones de pertenencia**, que son hiperrectángulos difusos que miden el grado de pertenencia de una instancia a una determinada clase (se encargan de la tarea de clasificación). Cada uno de ellos representa un atributo de entrada y en el esquema de la red neuronal se denotan con A_j^i, donde la i representa el gránulo i y j el j-ésimo atributo de la instancia de entrada; de esta manera, se establecen límites de decisión para discriminar entre diferentes clases. Para construir los gránulos, se utilizan un conjunto de neuronas denominadas **neu-**

ronas T-S (se denotan con TSn^i, donde i representa la clase i que tiene asignada), que son implementaciones neuronales de normas nulas (unas funciones de agregación que incluyen las T-normas y las S-normas como casos frontera, que son funciones aplicadas a conjuntos borrosos) y que se encargan de **agregar la salida de las funciones de pertenencia** A_j^i (la salida de cada A_j^i se denota con $\tilde{x}_j^i$) . Tras el paso de agregación, en la capa de decisión se **comparan los distintos valores de agregación obtenidos de los diferentes gránulos** y aquél que consiga el mayor valor produce como salida un vector $\bar{C}_k^{[h]}$ (siendo k la clase k) en el que se establece un 1 en la posición correspondiente a la clase asociada al gránulo y un 0 en las demás, clasificando de esta manera la instancia en la clase de dicho gránulo. Este vector se compara con el vector de salida deseado $C_k^{[h]}$, llevando a cabo de esta manera el cálculo del error de estimación ε_k.

El número de gránulos con el que trabaja el algoritmo es pequeño y no es necesario predefinirlo, sino que se van creando durante el proceso de evolución del modelo. Si hay un *concept drift*, a medida que van llegando nuevas instancias se **crean nuevos gránulos** o se **actualizan los que existen**. Cuando llega una nueva instancia de entrenamiento, pueden ocurrir tres cosas: la nueva instancia se ajusta a **más de un gránulo**, a **un gránulo** o a **ninguno**. En el primer caso, se averigua cuál es el gránulo cuyo valor de pertenencia de la nueva instancia al mismo (valor de agregación) es el **mayor** y se adapta dicho gránulo a la instancia modificando los parámetros de la red siempre y cuando la clase del gránulo coincida con el de la instancia; en caso contrario, se mira el gránulo con el **segundo mayor valor** y así sucesivamente. En el segundo caso, se adapta el ejemplo al **único gránulo en el que encaja** y, en el tercero, se crea un **nuevo gránulo** que se adecúe al nuevo ejemplo. Los parámetros que se pueden modificar para adaptar los gránulos a los nuevos ejemplos son los **pesos** w_j^i, que establecen la importancia de cada atributo j en cada gránulo i (se decrementa su relevancia de forma constante a medida que los gránulos son menos recientes para preservar la eficiencia del modelo); los **pesos** δ^i, que se establecen en función de la cantidad de información presente en los gránulos (también se van degradando con el transcurso del tiempo) y los **parámetros de las funciones de pertenencia de los gránulos**.

Con respecto a la propuesta anterior, los autores de la misma la extienden en Leite et al. (2010) para añadirle la capacidad de llevar a cabo un **aprendizaje semisupervisado** del modelo. El algoritmo que utilizan para tratar con instancias cuya clase es desconocida consiste en **etiquetarlas** y, cuando esté disponible la clase de la misma, **procesar el ejemplo de forma normal**. Para llevar a cabo la tarea de etiquetado de la instancia, se calcula el punto medio de cada uno de los gránulos y se asigna a la instancia la clase del gránulo cuyo punto medio **se encuentra más cerca de dicha instancia**. Además, en este trabajo **se monitoriza las distancias entre los diferentes gránulos** mediante la construcción de una matriz de distancias, de tal forma que si la distancia entre dos gránulos está por debajo de un umbral predefinido, entonces se pueden **reasignar a la misma clase** o **unir**. La distancia entre gránulos también la utilizan para **detectar valores atípicos**, de tal forma que si la distancia entre un gránulo creado por una nueva instancia y los demás gránulos

es mayor que un determinado umbral, entonces se considera un valor atípico.

En las anteriores propuestas (Leite et al. (2009) y Leite et al. (2010)) la actualización de los pesos de la red se realiza de forma lineal, sin tener en cuenta los datos de entrada, lo que supone una desventaja puesto que no tiene en cuenta las no linealidades que pueden estar presentes en el proceso de clasificación de las instancias y, por tanto, la precisión de clasificación se puede ver afectada. De esta manera, para solventar este inconveniente, en Kumar et al. (2016) proponen una red neuronal granular cuyos pesos se aprenden y se actualizan realizando una **retropropagación del error de salida de la red**, de tal forma que la modificación de los pesos se lleva a cabo teniendo en cuenta los datos. Aparte de esto, en esta propuesta, para granular los datos de entrada, utilizan un método denominado **granulación por clases** (*class based granulation*, CB), que mejora el desempeño del modelo en las tareas de clasificación; debido a estas características, el algoritmo que plantean lo denominan **class based progressive granular neural network** (CBPGNN). La red que se obtiene con este algoritmo es la que se muestra en la Figura 3.8.

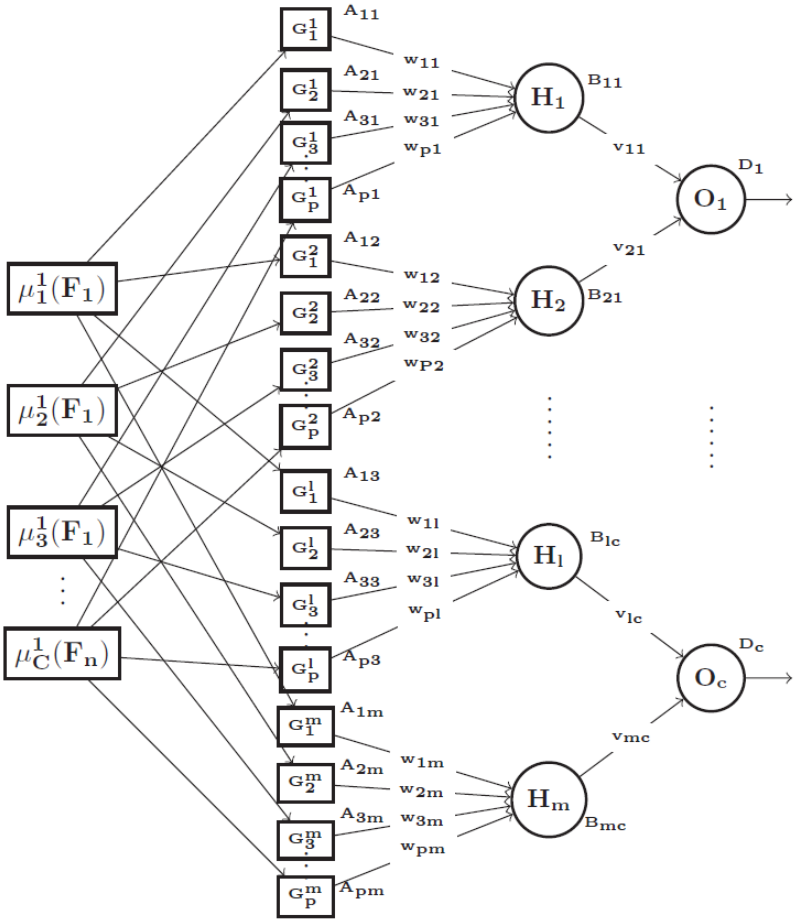

Figura 3.8: Estructura de la red neuronal construida con el algoritmo CBPGNN. Fuente: Kumar et al. (2016)

Este algoritmo lleva a cabo dos fases de granulación, siendo la primera el empleo

del método CB sobre las instancias de entrada. La idea sobre la que se basa este método es representar cada uno de los n atributos de las instancias por su **grado de pertenencia a cada una de las c clases del problema**, de tal forma que se construye una matriz de tamaño $p = n \times c$ a partir de estos valores; cada uno de los elementos de esta matriz corresponden a los **nodos de la capa de entrada**, y se denotan con $\mu_j^i(F_k)$ (i corresponde a la instancia i, j corresponde a la clase j y F_k corresponde al atributo k). Una vez obtenidos los patrones granulados del método CB, a partir de éstos se añaden otros gránulos, que son los **nodos de la capa progresiva** (el número de nodos está definido por el número de características granuladas p y el número de gránulos de alto nivel m, y cada uno de ellos se denota con G_j^i, donde i corresponde al gránulo de alto nivel i al que pertenece el nodo y j corresponde a la característica granulada j proveniente de la matriz vista anteriormente), definidos por **funciones de pertenencia trapezoidal**. Las funciones de pertenencia trapezoidal miden el grado de pertenencia de los patrones a dichos gránulos, que es la salida de los nodos de la segunda capa. Estas salidas se combinan, ponderándolas con los pesos w_{ji} (i corresponde al gránulo i y j corresponde a la característica granulada j), en los **nodos de la capa oculta** (representan los m gránulos de alto nivel, que están conformados por los gránulos de la segunda capa y se denotan con H_i, donde i corresponde al gránulo de alto nivel i). Las salidas de los nodos de la capa oculta se vuelven a agregar en los **nodos de la capa de salida** (denotados con O_j, donde j es el nodo j de la capa de salida), utilizando en cada uno de estos nodos las salidas de aquellos gránulos de alto nivel que pertenezcan a la clase correspondiente. Las salidas que se agregan en los nodos de la capa de salida están ponderadas por los pesos v_{ij}, donde i corresponde al gránulo de alto nivel y j corresponde al nodo de salida j. Los valores que producen los nodos de la última capa se utilizan para **etiquetar el patrón granulado** escogiendo la clase cuyo nodo de la capa de salida tenga la salida más alta, y si la clase predicha no corresponde con la clase verdadera, entonces los pesos de las distintas capas se actualizan para reducir el error global del modelo. Los gránulos de la red neuronal también se actualizan con nuevos patrones granulados que pertenezcan a dichos gránulos.

Por otra parte, en Read et al. (2015) proponen utilizar **redes de neuronas profundas** (*deep learning*) para tratar con flujos de datos, a diferencia de las anteriores propuestas, que construyen redes de neuronas superficiales (con pocas capas). Concretamente, se centran en resolver problemas de flujos de datos semisupervisados. El modelo de *deep learning* en el que se basan se denomina **redes de creencias profundas** (*Deep Belief Networks*), que consisten en máquinas de Boltzmann apiladas. Cada una de las máquinas de Boltzmann está compuesta por una capa visible formada por nodos correspondientes a los valores de los atributos originales de la instancia de entrada y una capa oculta compuesta por nodos que establecen una representación compacta de los patrones subyacentes de los datos de entrada. La finalidad de estos nodos ocultos es **obtener características que estén más relacionadas con las etiquetas de salida**, de manera que es más fácil llevar a cabo un aprendizaje del modelo para realizar tareas de clasificación.

Para obtener las representaciones del modelo anterior solo utilizan los atributos

de la instancia de entrada (de manera no supervisada); para llevar a cabo la clasificación de los ejemplos, realizan dos propuestas: el método **DBN-*h*** y el método **DBN-BP**. En el primero, utilizan las características de más alto nivel provenientes de las redes de creencias profundas como **atributos de entrada** para construir un modelo de clasificación para flujos de datos ya existentes como el kNN y los árboles de decisión incrementales con el objetivo de que su precisión de clasificación sea mayor. En el segundo, **se utiliza directamente la red neuronal para clasificar las instancias** añadiendo una última capa con tantas neuronas como clases existan en el problema, de tal manera que la red se entrena utilizando el algoritmo de retropropagación. Con respecto al aprendizaje semisupervisado que realizan en este trabajo, si la instancia no tiene etiqueta, entonces se utiliza para **actualizar las máquinas de Boltzmann** (no utilizan las clases de las instancias); en caso contrario, se emplea para **actualizar el clasificador correspondiente**. En este sentido, se diferencia en la forma de tratar las instancias no etiquetadas de la propuesta planteada en Leite et al. (2010), en la que se necesita la instancia etiquetada para incorporarla al modelo.

Otra propuesta en la que se aborda un método basado en aprendizaje profundo para la clasificación de flujos de datos es la planteada en Besedin et al. (2017), donde proponen un algoritmo que es capaz de adaptarse a la aparición de **nuevas clases** y que mantiene información aprendida previamente **sin almacenar instancias del pasado**. El algoritmo construye una nueva arquitectura de red neuronal basándose en un tipo de arquitectura profunda de neuronas denominado **Generative Adversarial Network** (GAN), cuya función principal es regenerar datos del pasado para compensar la ausencia de las instancias procesadas, cuya información es valiosa para el aprendizaje en línea del modelo. Concretamente, se utiliza el modelo **DCGAN**, cuya forma de entrenarse es parecido al GAN y posee alguna ventaja sobre éste, como la alta estabilidad en la fase de entrenamiento. El modelo DCGAN tiene la capacidad de representar los datos originales sobre los que se entrena y de generalizar la distribución de los mismos, de manera que los datos que genera se pueden utilizar para entrenar un clasificador que tenga un buen desempeño, en lugar de los datos originales. El algoritmo de aprendizaje que proponen para la construcción del modelo de red neuronal que incorpora un conjunto de DCGANs es el que se expone en la Figura 3.9.

En primer lugar, el flujo de datos se divide en tantas partes como clases existan, siendo S_i el conjunto de datos correspondiente a la clase i, de tal forma que los datos de entrada se van presentando al modelo **clase por clase**. Se proporciona como entrada al algoritmo el conjunto de datos S_1, correspondiente a la primera clase; se entrena un modelo generador G_1 para representar los datos originales pertenecientes a la primera clase (un DCGAN) y se descarta S_1. A continuación, se proporciona el conjunto de datos S_2 correspondiente a la segunda clase, que se utiliza para entrenar el generador G_2 y, una vez hecho esto, se entrena un clasificador que discrimine las dos primeras clases (C_{12}) proporcionándole un conjunto de datos generado por G_1, denominado S_1^*, y el conjunto de datos original S_2; después se descarta S_2. Tras esto, con S_3 se entrena el generador G_3 y se entrena un clasificador que discrimine entre las

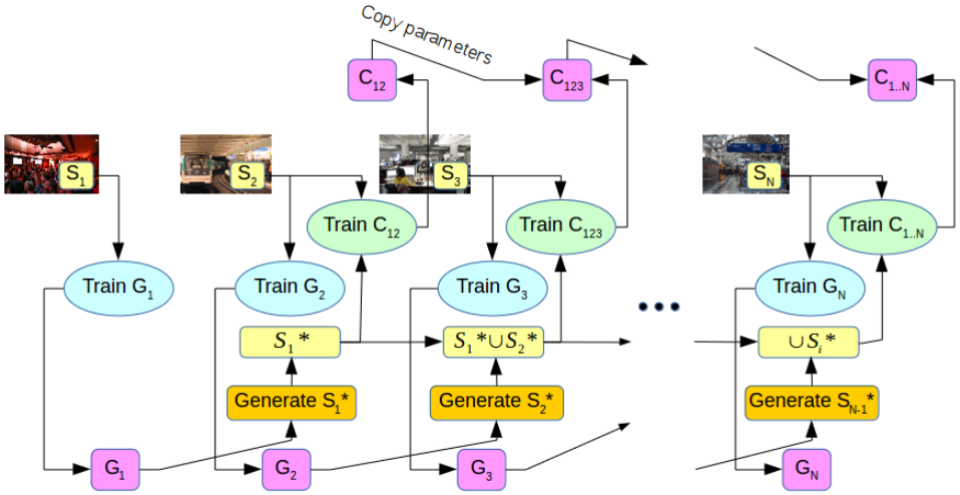

Figura 3.9: Representación esquemática del algoritmo de aprendizaje. Fuente: Besedin et al. (2017)

tres clases procesadas hasta el momento (C_{123}); para ello, se proporciona al mismo el conjunto de datos S_1^* generado por G_1, un conjunto de datos S_2^* generado por G_2 y el conjunto de datos real S_3, además de incorporar en el clasificador los parámetros del clasificador C_{12} (véase Figura 3.10). Estos pasos se repiten con todas las clases presentes en los datos hasta que se obtiene un clasificador que contemple cada una de ellas; si aparece una nueva clase en los datos, se sigue el mismo procedimiento comentado con anterioridad.

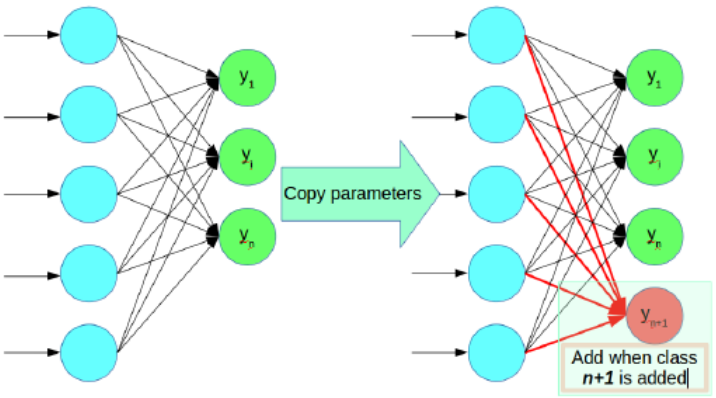

Figura 3.10: Copia de los parámetros del clasificador anterior y adición de la nueva clase. Fuente: Besedin et al. (2017)

La Tabla 3.4 muestra la comparativa de las diferentes propuestas de algoritmos de aprendizaje supervisado para flujos de datos basados en redes neuronales.

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
eGNN (Leite et al. (2009))	Ajuste de los parámetros de los gránulos y de la red, así como eliminación de gránulos inactivos	Sí		Sí (no es seguro para categóricas)	No*		Sí* (El eGNN puede empezar a aprender desde cero y sin ningún conocimiento previo de las propiedades estadísticas de los datos y las clases.)
eGNN (Leite et al. (2010))	Ajuste de los parámetros de los gránulos y de la red, así como eliminación de gránulos inactivos			Sí (no es seguro para categóricas)	No*	Sí*	Sí* (El eGNN puede empezar a aprender desde cero y sin ningún conocimiento previo de las propiedades estadísticas de los datos y las clases.)
DBN-h y DBN-BP	No*			Sí	Sí		No*
CBPGNN	Ajuste de los parámetros de los gránulos y de la red* (no mencionan explícitamente que se eliminen gránulos inactivos, sino que mejora la actualización de los pesos)				Sí (no es seguro para categóricas)		Sí* (nuevas clases)
Besedin et al. (2017)	DCGAN						Si* (nuevas clases)

Tabla 3.4: Algoritmos de aprendizaje supervisado para flujos de datos basados en redes neuronales

3.2.5. k-vecinos más cercanos

A diferencia de las técnicas de aprendizaje automático vistas anteriormente, el paradigma clasificatorio de ***k*-vecinos más cercanos** no construye un modelo y realiza tareas de predicción a partir de dicho modelo, sino que se basa en almacenar instancias y clasificar un nuevo ejemplo a partir de su proximidad con esas instancias, de tal forma que este algoritmo destaca por su simplicidad. En el caso del manejo de flujos de datos para llevar a cabo labores de clasificación, el desempeño del kNN puede llegar a **ser superior a un algoritmo que se basa en una construcción de un modelo** puesto que éste, debido a la naturaleza dinámica de los flujos de datos, puede cambiar rápidamente, de tal forma que hay que modificar el modelo. Además, **no realiza suposiciones sobre la forma de la distribución** y aprende la estructura de la hipótesis **directamente de los datos de entrenamiento**; en los flujos de datos normalmente se tiene un conocimiento previo de la distribución de los datos. No obstante, el proceso para encontrar los k vecinos más cercanos de una nueva instancia a clasificar puede ser **lento**, una característica del algoritmo inconcebible en la clasificación de flujos de datos. Se han planteado diferentes propuestas para acelerar la búsqueda de los k vecinos más cercanos, como los *k-d trees* o proporcionando resultados aproximados con garantías del error que se comete; sin embargo, dadas las propiedades de las mismas, no son adecuadas para tratar flujos de datos.

Para solventar el inconveniente anterior, en Khan et al. (2002) realizan una propuesta de clasificación de flujos de datos espaciales basado en kNN que emplea una estructura que representa los datos espaciales de entrada originales sin pérdida de información y de forma compacta denominada ***Peano Count Trees*** (*P-trees*) (para una implementación del algoritmo más eficiente utilizan una variante denominada *PM-Tree*). Concretamente, esta estructura representa los flujos de datos espaciales *bit a bit* en una disposición recursiva por cuadrantes, de tal forma que permite el cálculo eficiente de los k vecinos más cercanos realizando operaciones lógicas AND/OR en la misma. Usando esta estructura, en este trabajo proponen dos algoritmos basados en dos métricas de distancia para calcular la distancia entre ejemplos: la distancia **max**, que es una distancia de Minkowski con parámetro $p = \infty$ (véase ecuación 3.2), y una nueva distancia que proponen los autores de esta propuesta denominada **HOBS** (*Higher Order Bit Similarity*), que se basa en medir la similitud de dos valores usando los bits más significativos de los m bits de los dos números binarios que los representan (la similitud se muestra en la ecuación (3.3) y la distancia en la ecuación (3.4)) puesto que, al buscar la proximidad de estos valores, los bits menos significativos no adquieren tanta relevancia en este proceso.

$$D(X,Y) = \sqrt[p]{\sum_{i=1}^{n} |x_i - y_i|^p} \overset{p=\infty}{\Longrightarrow} max_{1\leq i\leq n}|x_i - y_i| \tag{3.2}$$

$$HOBS(A,B) = max\{s|i \leq s\, tal esque\, a_i = b_i\} \tag{3.3}$$

$$d(A, B) = m - HOBS(A, B) \tag{3.4}$$

donde a_i y b_i son el i-ésimo bit de los valores A y B respectivamente.

Además de lo comentado previamente, los algoritmos propuestos, en lugar de analizar individualmente cada una de las instancias almacenadas para calcular los k vecinos más cercanos, van realizando una expansión de la vecindad de una nueva instancia a clasificar hasta contener un número k de vecinos más próximos, que puede ser de forma desnivelada en ambos lados de los intervalos de valores de los atributos (con la distancia **HOBS**) o *constante* (***Perfect Centering*** con la distancia **max**); esta última proporciona una mejor precisión de clasificación a costa de aumentar un poco el coste computacional. Esta expansión comienza buscando instancias que tengan **coincidencias exactas** con la nueva instancia a clasificar; si el número de ejemplos encontrados es menor que k, entonces **se expande la vecindad del nuevo ejemplo** estableciendo rangos de valores de los atributos hasta que se obtenga un número k de vecinos más próximos. En este proceso de búsqueda de estos k vecinos más próximos, puede ocurrir que se obtenga un número de vecinos mayor que k y que haya un conflicto a la hora de escoger los vecinos más próximos por estar presentes en la vecindad posibles candidatos que sean equidistantes a la nueva instancia (se encuentran en la frontera de la región de vecindad); para tratar este caso, proponen utilizar en los algoritmos una nueva forma de generar el conjunto de vecinos más cercanos denominada ***closed-kNN***. En este método se tienen en cuenta tanto las instancias dentro de la vecindad como en la frontera de la misma puesto que aquellas que se encuentran en el borde aportan información valiosa para la clasificación del nuevo ejemplo.

Por otra parte, Law y Zaniolo (2005) proponen el algoritmo denominado **ANN-CAD** (*Adaptive NN Classification Algorithm for Data-streams*), que se basa en la idea de realizar una descomposición del espacio definido por los atributos de las instancias de entrenamiento con el objetivo de obtener una **representación multirresolución de los datos** y, a partir de ésta, **encontrar los vecinos más cercanos de una instancia a clasificar de forma adaptativa**. Para llevar a cabo la clasificación del nuevo ejemplo, al igual que en la propuesta Khan et al. (2002), se lleva a cabo una expansión del área cercana a dicho ejemplo hasta poder realizar una predicción de la clase a la que pertenece, pero en este caso dicha expansión se realiza teniendo en cuenta diferentes niveles de resolución del espacio definido por las variables predictivas (en lugar de en un mismo nivel de resolución, como ocurre en Khan et al. (2002)).

Para conseguir lo mencionado con anterioridad, se establecen una serie de pasos. El primero de ellos es particionar el espacio de atributos en espacios discretizados denominados **bloques** (véase Figura 3.11); para ello, se separan las instancias de entrenamiento de las diferentes clases y, para cada una de ellas, se asignan dichas instancias al bloque que les corresponda, de manera que se almacena en cada bloque el **número de instancias que tiene asignadas**. Una vez obtenido dicho número en todos los bloques, se les asocia a los mismos la clase mayoritaria de las instancias que almacenan **de forma jerárquica**, es decir, en diferentes niveles de resolución (véase

Figura 3.12). De esta forma, primero se les asigna a los bloques correspondientes al nivel de resolución más bajo la etiqueta mayoritaria (puede ocurrir que un bloque no tenga ninguna etiqueta asignada). A continuación, se construyen bloques de un nivel de resolución más alto y se les asigna la **clase mayoritaria** si ésta tiene más puntos que la segunda clase mayoritaria en función de un determinado umbral; en caso de que esto no ocurra, se le asigna la etiqueta **M** (*Mixed*). Estos pasos se repiten en niveles de resolución superiores hasta llegar al nivel más alto, compuesto por un solo bloque.

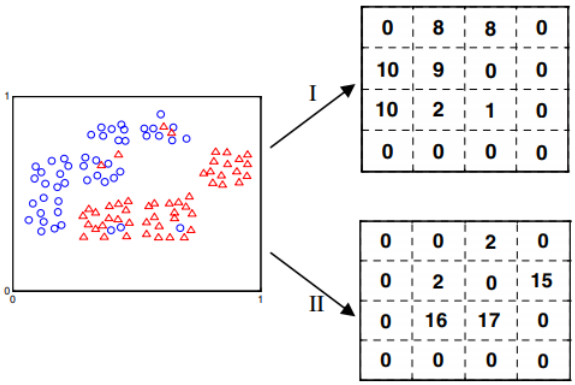

Figura 3.11: Representación del espacio definido por los atributos discretizado en bloques. Fuente: Law y Zaniolo (2005)

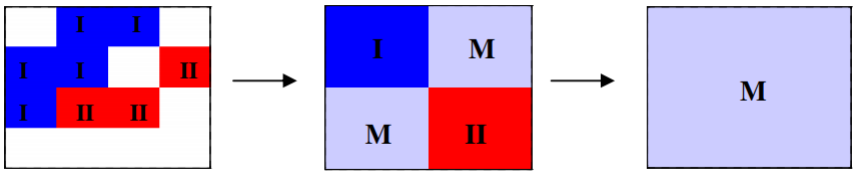

Figura 3.12: Diferentes niveles de resolución para la tarea de clasificación. Fuente: Law y Zaniolo (2005)

Para realizar la clasificación de una instancia, se comienza asignando la misma a un bloque del nivel de resolución más bajo (véase Figura 3.13) y se va subiendo de nivel cuando sea necesario. En el proceso de asignación de una etiqueta a un nuevo ejemplo, se pueden dar tres situaciones. La primera consiste en que el bloque al que pertenece la nueva instancia tenga asociada una **clase**; en este sentido, se clasifica dicha instancia en la clase del bloque. En segundo lugar, puede que el bloque **no tenga asignada ninguna etiqueta** (tanto de clase como con la etiqueta M); en este caso, para poder clasificar la instancia **se sube a un nivel de resolución superior** con el objetivo de encontrar un bloque que tenga una etiqueta asignada. En último lugar, el bloque puede tener asignada la etiqueta **M**; ante esto, se baja un nivel de resolución y **se calculan los bloques más cercanos de dicho nivel a**

la instancia (el bloque al que pertenece la instancia en ese nivel no tiene asignado una etiqueta puesto que la instancia ha tenido que subir un nivel de resolución), de manera que se le asigna la clase mayoritaria de esos vecinos. La actualización del modelo cuando llegan nuevas instancias solo se realiza en aquellos bloques de los diferentes niveles de resolución a los que pertenecen las mismas, y se adapta a la aparición de *concept drifts* mediante la aplicación de un **mecanismo de olvido exponencial a los datos menos recientes**.

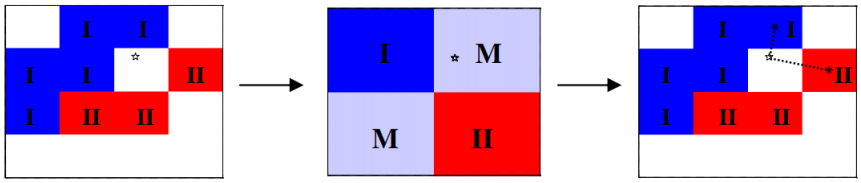

Figura 3.13: Clasificación de una nueva instancia utilizando diferentes niveles de resolución. Fuente: Law y Zaniolo (2005)

Otra propuesta en la que se utiliza el algoritmo de vecinos más cercanos es la planteada en Aggarwal et al. (2006), en la que se desarrolla un **clasificador bajo demanda** utilizando el modelo de *microclustering* definido en Aggarwal et al. (2003) pero adaptándolo al aprendizaje supervisado de patrones subyacentes en flujos de datos y empleando dicho modelo para la clasificación de nuevas instancias utilizando el **algoritmo kNN**. Un *microcluster* supervisado de un conjunto de instancias está definido por un vector en el que se almacena la suma de los cuadrados de cada uno de los atributos de las instancias, la suma de cada uno de los atributos de las instancias, la suma de los cuadrados de los instantes de tiempo en los que llegaron cada una de las instancias, la suma de los instantes de tiempo en los que llegaron cada una de las instancias, el número de instancias que representa el *microcluster* y la clase asociada al *microcluster*. Esta información que se guarda por cada uno de los *microclusters* corresponde a **estadísticas resumidas** de las instancias que contienen (en Law y Zaniolo (2005) en cierta forma también se guarda información resumida de las instancias), de manera que la utilización de dichos *microclusters* supone una ventaja sobre los datos originales puesto que la tarea de clasificación se realiza sobre información más **compacta** (a diferencia de la propuesta Khan et al. (2002), que se basa en los datos originales). La creación inicial de los *microclusters* se realiza **fuera de línea**, de tal forma que se crea el mismo número de *microclusters* para cada una de las clases con el algoritmo *k-means* por separado para cada uno de los conjuntos de instancias pertenecientes a cada una de las clases. Para llevar a cabo el mantenimiento de los *microclusters* a medida que van llegando nuevas instancias, se tiene en cuenta que solo se permite un número máximo de *microclusters*. De esta manera, una nueva instancia que llega se **intenta asociar al *microcluster* más cercano** (utilizando la distancia a su centroide) que tenga su misma clase. En el caso de que no se pueda introducir en un *microcluster* debido a que no está lo suficientemente cerca del más cercano, **se crea un nuevo *microcluster* que lo**

contenga y, si se supera el máximo número de *microclusters* permitido, es necesario **eliminar un *microcluster***, acción que solo se realiza si se demuestra que no tiene una presencia activa en el flujo de los datos. En caso de que no se demuestre, **se unen dos *microclusters* existentes** que tengan la misma clase.

Para llevar a cabo la tarea de clasificación de flujos de datos, en este trabajo proponen **encontrar el horizonte temporal adecuado** en función de cómo evolucione el concepto que describe a los datos. Para averiguarlo, en el proceso de entrenamiento de los *microclusters* (comentado anteriormente) se utiliza la mayoría de los datos, y se deja una pequeña porción del flujo de datos para encontrar el **mejor horizonte de clasificación**. En este proceso de búsqueda, se utiliza una estructura denominada ***geometric time frame***, en la que se guardan **capturas** (*snapshots*) de los estados de los *microclusters* en **diferentes instantes de tiempo con diferentes niveles de granularidad temporal** en función de si son más recientes (mayor granularidad) o menos (menor granularidad). De esta forma, esta estructura permite recuperar los diferentes *microclusters* presentes en un instante de tiempo determinado con el objetivo de testearlo sobre la pequeña porción de datos mencionada anteriormente para calcular el desempeño de la clasificación de los *microclusters* en ese instante de tiempo. Para averiguar dicho desempeño, se utiliza el **algoritmo del vecino más cercano** (1NN), que clasifica cada ejemplo a la clase del *microcluster* más cercano. La clasificación de una nueva instancia se realiza **escogiendo un número de horizontes temporales** que obtienen las mejores precisiones de clasificación en la pequeña porción de datos reservada con anterioridad; por cada uno de los horizontes temporales se aplica el algoritmo del vecino más cercano sobre la nueva instancia y se le asigna la clase mayoritaria de dichos horizontes temporales.

Por otro lado, en Bifet et al. (2013), para tratar el aprendizaje de flujos de datos proponen un algoritmo denominado ***probabilistic adaptive window*** (PAW), cuya finalidad es mejorar el modelo de ventana tradicional incluyendo tanto **instancias recientes como antiguas**. El objetivo de esto es tener en cuenta información del pasado (de *concept drifts* que ocurrieron) que puede ser relevante para el modelo a la vez que se utilizan las instancias recientes para adaptar el modelo a los nuevos posibles cambios del concepto que describe a los datos. En este algoritmo, en lugar de almacenar una ventana de ejemplos recientes limitada, las nuevas instancias que van llegando se introducen en la ventana y cada uno de los ejemplos de dicha ventana tiene una **probabilidad de ser eliminado de la misma**, teniendo menos probabilidad de eliminación aquellos que son más recientes, de manera que se mantiene un compromiso entre el almacenamiento de información del pasado y aquella que es reciente. Para mantener esta ventana de instancias, se basan en un algoritmo denominado ***Morris approximation counting***, que permite guardar la información de la ventana utilizando solo un número logarítmico de las instancias incluidas en la misma. Para realizar la tarea de clasificación, el modelo de ventana PAW lo utilizan con el algoritmo **kNN**. Basándose en este algoritmo de clasificación, proponen tres métodos diferentes: el primero de ellos no realiza una detección explícita de *concept drifts* (kNN con PAW); el segundo utiliza el detector de *concept drifts* denominado

ADWIN (kNN con PAW y ADWIN), que elimina aquellas instancias que no se corresponden con la distribución de los datos actual (en las anteriores propuestas no se lleva a cabo una detección explícita del *concept drift*) y el tercero es un método de *ensemble* con el primer método como base.

Las propuestas vistas hasta ahora sobre el algoritmo kNN para flujos de datos tratan con problemas multiclase. A diferencia de éstas, en Spyromitros-Xioufis et al. (2011) proponen un algoritmo que trata con clasificación multietiqueta para flujos de datos utilizando una versión modificada del algoritmo kNN denominado ***Multiple Windows Classifier*** (MWC). En este algoritmo, por cada una de las clases, se mantienen dos ventanas de tamaño fijo, uno para **ejemplos positivos** y otro para **ejemplos negativos**; esto se realiza debido a que cada una de las clases por separado suele tener su propio ritmo de cambio en el concepto que las describe, es decir, tiene su **propio patrón de cambios de concepto**. De esta manera, tratan cada una de las clases como un problema de aprendizaje diferente; en este sentido, utilizan la aproximación ***binary relevance*** (BR), que transforma un problema multietiqueta en un conjunto de problemas de clasificación binaria. En este trabajo aplican esta aproximación debido a que lidia eficientemente con los *concept drifts* entre las distintas etiquetas y con nuevas clases entrenando un nuevo clasificador binario para las mismas, además de que se puede paralelizar. Este algoritmo, a diferencia de las otras propuestas, trata con el problema de **desbalanceo de clases**; para ello, equilibran las instancias positivas y negativas (con *oversampling* y *undersampling* en función de un parámetro denominado *distribution ratio*.

Además de lo mencionado previamente, las múltiples ventanas que se crean no contienen las instancias originales, sino **referencias a las mismas**, que se guardan en una estructura de cola; las instancias se almacenan una única vez en un búfer compartido. De esta manera, para llevar a cabo la actualización de las ventanas cada vez que llega una nueva instancia, se inserta la misma en las ventanas correspondientes y, si cualquier ventana está llena, se elimina la instancia menos reciente de la misma. Para llevar a cabo la tarea de clasificación, se utiliza el algoritmo kNN adaptándolo a las estructuras explicadas con anterioridad. De esta manera, primero **se calculan las distancias de todas las instancias del búfer a la nueva instancia** y **se ordenan las mismas de menor a mayor distancia** (al igual que en Khan et al. (2002) se trabaja directamente con las instancias). Tras esto, **se recorre la lista que contiene las instancias ordenadas** con el objetivo de encontrar los k vecinos más cercanos para cada una de las clases; para ello, para cada una de las instancias de la lista, se utilizan las ventanas asociadas a cada una de las clases. Una vez obtenidos los k vecinos más cercanos de cada una de las clases, se observa cuál es la clase que ha obtenido más votos.

Otra propuesta en la que utilizan el algoritmo k-vecinos más cercanos para realizar clasificación de flujos de datos es la planteada en Losing et al. (2016), donde proponen el algoritmo **SAM-kNN**, que se basa en una arquitectura que desarrollan denominada SAM (*Self Adjusting Memory*), cuyo objetivo es manejar diferentes tipos de *concept drifts*. Para ello, la arquitectura se fundamenta en emular la estructura de la memoria del ser humano, de manera que proponen utilizar por un lado

una **memoria a corto plazo** (*Short-Term Memory*, STM) y una **memoria a largo plazo** (*Long-Term Memory*, LTM). El STM es una ventana deslizante dinámica que contiene las instancias más recientes, de manera que guarda el conocimiento actual que se tiene de los datos; de manera análoga al ser humano, además del almacenamiento de información reciente, tiene una capacidad limitada. Con respecto al LTM, contiene **conocimiento menos reciente** que es consistente con el contenido del STM y que puede aportar información valiosa para la realización de tareas de predicción en el futuro (por ejemplo para manejar *concept drifts* recurrentes). Para llevar a cabo la predicción de una nueva instancia, además de utilizar las memorias a corto y largo plazo, utilizan una memoria combinada de las dos anteriores. Cada una de estas memorias se utiliza para aplicar el algoritmo kNN sobre **la ventana de instancias más recientes**, de manera que aquella que tenga más precisión de clasificación es la que se utiliza para asignar la clase a la nueva instancia.

Para llevar a cabo la actualización de las memorias a corto y largo plazo se aplican procedimientos diferentes. En el caso de STM, a medida que llegan nuevas instancias, **se va adaptando el tamaño de la memoria**; para ello, se evalúan diferentes tamaños para el STM y se escoge aquel que tenga menor *prequential error*. En el caso del LTM, con la llegada de nuevos datos es necesario **comprobar su consistencia con los datos de la memoria a corto plazo**, es decir, que no haya contradicciones entre las instancias de las dos memorias; una vez hecho esto, si el tamaño de la ventana del STM se disminuye, los ejemplos que se descartan **se insertan en el LSTM**. Si se excede el tamaño de la memoria a largo plazo, se compacta la información de la misma utilizando el algoritmo de agrupamiento *k-means++* sobre cada uno de los conjuntos de instancias pertenecientes a cada una de las clases. En este sentido, tiene semejanza con el método propuesto en Aggarwal et al. (2006) puesto que utiliza un método de agrupamiento para compactar la información, pero en Aggarwal et al. (2006) aplican el método en todo el flujo de datos, guardan la información procedente del agrupamiento con diferentes granularidades temporales y realizan la clasificación escogiendo el mejor horizonte temporal en ese momento. Por otra parte, en Bifet et al. (2013) también tienen en cuenta información tanto del pasado como del presente, pero lo llevan a cabo manteniendo una única ventana que contenga una muestra de las instancias, dando más peso a las instancias recientes que a las del pasado.

La Tabla 3.5 muestra la comparativa de las diferentes propuestas de algoritmos de aprendizaje supervisado para flujos de datos basados en kNN.

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos	Valor k prefijado
kNN P-trees (Khan et al. (2002))	No			Sí (no es seguro para categóricas)				Sí (método de prueba y error)
ANNCAD (Law y Zaniolo (2005))	Mecanismo de olvido exponencial sobre los datos			Sí (dan la posibilidad de utilizar cualquier distancia)		Sí		No
Clasificación bajo demanda (Aggarwal et al. (2006))	Horizontes temporales con la estructura *geometric time frame*	Sí*		Sí (no categóricas)	Sí		Sí (nuevas clases)	Sí (vecino más cercano)
kNN con PAW (Bifet et al. (2013))	PAW y ADWIN			Sí* (no menciona explícitamente si continuas y/o categóricas)				Sí*
MWC (Spyromitros-Xioufis et al. (2011))	Múltiples ventanas deslizantes			Sí* (no menciona explícitamente si continuas y/o categóricas)			Sí (nuevas clases)	Sí
SAM-kNN (Losing et al. (2016))	Memorias a corto y largo plazo		No*	Sí* (no menciona explícitamente si continuas y/o categóricas)	No*	Sí*	Sí*	Sí

Tabla 3.5: Algoritmos de aprendizaje supervisado para flujos de datos basados en kNN

3.2.6. Máquinas de vectores soporte

Los algoritmos de aprendizaje automático basados en **máquinas de vectores soporte**, dado un conjunto de datos de entrenamiento de tamaño N, tienen una complejidad temporal $O(N^3)$ y espacial $O(N^2)$, de manera que no son adecuados para aplicarlos sobre conjuntos de gran tamaño, como ocurre con los flujos de datos. Para adaptar las SVMs a la clasificación de una ingente cantidad de datos, así como a la naturaleza cambiante de los mismos, una de las primeras propuestas realizadas

teniendo en cuenta esas características es la planteada en Klinkenberg y Joachims (2000). En la misma se propone un algoritmo que maneja *concept drifts* con SVMs manteniendo una **ventana de ejemplos de entrenamiento** cuyo tamaño se va ajustando en cada lote de datos con el objetivo de adaptarse al comportamiento dinámico de los datos y minimizar el **error de generalización estimado**, que se define en esta propuesta como el número de errores *leave-one-out* dividido por el número total de instancias utilizadas para calcular el error.

El trabajo que realizan se centra en obtener una manera de seleccionar el tamaño adecuado de la ventana sin la necesidad de que implique la utilización de muchos parámetros que sean dificiles de ajustar. Para ello, emplean un método denominado **estimaciones $\boldsymbol{\xi\alpha}$**. El método recibe este nombre debido a los parámetros que utiliza para calcular las estimaciones, que son $\overrightarrow{\xi}$, el vector de errores de entrenamiento cometidos por las instancias en el cálculo de la solución del problema primal de entrenamiento del SVM, y $\overrightarrow{\alpha}$, la solución del problema dual de entrenamiento del SVM. Este método lleva a cabo una estimación del desempeño de las SVMs; para ello, definen un **límite superior** en el número de errores *leave-one-out* en lugar de calcularlos utilizando todas las instancias puesto que conlleva un costo computacional alto. Esta estimación del error generalizado se realiza sobre diferentes posibles ventanas (diferentes lotes de datos), de manera que, por cada una de las ventanas, se entrena una SVM y se aplica la estimación $\xi\alpha$ sobre un número de instancias correspondientes a las más recientes (el mismo número de instancias en todos los posibles tamaños de ventana). El tamaño de ventana que se elige es aquél que minimice la estimación $\xi\alpha$.

En ocasiones, las instancias provenientes de un flujo de datos puede que no tengan ninguna etiqueta asociada, de manera que no sabemos la clase a la que pertenecen. En la propuesta anterior se asume que se conoce la clase en la que se clasifica cada instancia disponible; no obstante, los ejemplos no etiquetados pueden aportar información relevante para la construcción de una SVM y mejorar el desempeño de la misma. De esta manera, en Klinkenberg (2001) se extiende el trabajo realizado en Klinkenberg y Joachims (2000) para que sea capaz de tratar con **instancias no etiquetadas** y disminuir la necesidad de utilizar ejemplos etiquetados para lidiar con posibles *concept drifts*. Dicha propuesta utiliza **SVMs transductivas** que, a diferencia de las SVMs inductivas (utilizadas en Klinkenberg y Joachims (2000)), son capaces de utilizar en la fase de entrenamiento instancias no etiquetadas, además de tener en cuenta un conjunto de testeo (si el instante de tiempo actual es t, este conjunto corresponde al bloque de datos obtenidos en el instante de tiempo $t+1$, y se supone que las instancias de dicho conjunto no están etiquetadas). De esta manera, las SVMs transductivas se centran en encontrar un etiquetado de las instancias no etiquetadas de tal forma que **se halle un hiperplano que separe todas las instancias con el máximo margen**.

A partir de lo comentado anteriormente, esta propuesta se basa en dos fases. En primer lugar, utilizan el algoritmo planteado en Klinkenberg y Joachims (2000) para encontrar el tamaño adecuado de la ventana para las **instancias etiquetadas** denominado $w_{labeled}$ (ignora las instancias no etiquetadas) utilizando las estimaciones

$\xi\alpha$ para una SVM inductiva. En segundo lugar, se emplea de forma muy similar el mismo algoritmo para hallar el tamaño apropiado para las **instancias no etiquetadas** (teniendo en cuenta las instancias etiquetadas también), denominado $w_{unlabeled}$, utilizando una SVM transductiva. Concretamente, se prueban diferentes tamaños de ventana (aumentando el tamaño desde las instancias más recientes hacia atrás en el pasado), de tal forma que, para cada una de ellas, se entrena una SVM transductiva sobre las instancias de la misma considerando que los ejemplos que residen fuera de la ventana definida por el valor $w_{labeled}$ (ventana de tamaño fijo que se utiliza al probar distintas ventanas sobre las que se realiza el entrenamiento de una SVM transductiva) y los del conjunto de testeo **no están etiquetados** (las instancias dentro de la ventana de tamaño $w_{labeled}$ puede contener tanto instancias etiquetadas como no debido a que para el cálculo del valor $w_{unlabeled}$ sí se tienen en cuenta las instancias no etiquetadas). Tras esto, se calcula la estimación $\xi\alpha$ en las instancias del conjunto de testeo. Se escoge como tamaño de la ventana para instancias no etiquetadas aquél que tenga el **mínimo valor** de la estimación $\xi\alpha$. La construcción de dos ventanas, una para datos etiquetados y otra para datos no etiquetados, se fundamenta en la idea de que el ritmo al que se producen cambios en las distribuciones de probabilidad $p(C|\mathbf{X})$ y $p(\mathbf{X})$ puede ser diferente.

Otra propuesta realizada para tratar con la complejidad tanto temporal como espacial de las *máquinas de vector soporte* con flujos de datos es la planteada en Tsang et al. (2005), donde proponen el algoritmo denominado **Core Vector Machine** (*CVM*). En este trabajo, abordan el problema de optimización en el que se basa el SVM (encontrar el hiperplano que maximice el margen entre clases) formulándolo como un problema denominado ***minimum enclosing ball*** (*MEB*), en el que se busca obtener la **hiperesfera de radio mínimo que contenga un conjunto de puntos determinado**. Los métodos que buscan esta estructura de forma exacta *no escalan bien a medida que aumentan las dimensiones de los datos*, de tal forma que en este trabajo proponen una aproximación del cálculo del *MEB* denominada **aproximación $\mathbf{1+\epsilon}$**, que se puede obtener eficientemente utilizando *core-sets*. Esta aproximación consiste en encontrar un subconjunto de las instancias de entrada (*core-set*), que denominan *core vectors* (corresponden a los vectores soporte de un determinado *SVM*), que permita obtener una buena representación del conjunto de instancias original, donde la aproximación esta definida por un parámetro ϵ; concretamente, un subconjunto de datos X es un ***core-set*** de un conjunto de datos S si la expansión de su *MEB* por un factor de $(1+\epsilon)$ contiene a S. Una vez que calcula el *MEB* sobre dicha aproximación, el problema establecido por el algoritmo *SVM* se aplica directamente sobre el *MEB*. Este trabajo, a diferencia de la propuesta planteada en Klinkenberg (2001), no es capaz de detectar *concept drifts*; obtiene una representación aproximada del conjunto de datos original, pero puede ocurrir que haya instancias de esa representación que **no aporten información relevante para describir el concepto de los datos** en un instante de tiempo determinado.

En la propuesta anterior, cada iteración del algoritmo *CVM* implica resolver un problema definido sobre el *core-set* que, para hacerlo de forma eficiente, se requiere una **resolución numérica complicada** y, si el tamaño del conjunto de datos es

grande, el *core-set* también va a tener un gran tamaño, lo que implica que el algoritmo puede llegar a tener un gran coste computacional. De esta manera, en Tsang et al. (2007) proponen resolver un problema más simple que el *MEB* denominado ***enclosing balls*** (*EB*). En este problema, para hacerlo más sencillo que el *MEB*, establecen el radio de la hiperesfera **fijo**, de tal forma que no se requiere realizar las *resoluciones numéricas anteriores*; se elimina la parte de actualización del radio de la hiperesfera. Además, la hipersefera que se encuentra resolviendo el problema *EB* es similar a la que se obtiene solucionando el problema *MEB* y, por lo tanto, la solución del problema *EB* está **cerca de la solución óptima para el *SVM***.

Los algoritmos propuestos tanto en Tsang et al. (2005) como en Tsang et al. (2007) requieren *múltiples pasadas sobre el conjunto de datos*, característica que no es adecuada para tratar flujos de datos. Por ello, en Rai et al. (2009) presentan un algoritmo de construcción de un modelo *SVM* denominado **StreamSVM**, que revisa cada una de las instancias **una única vez** y, al igual que las otras propuestas, está basado en el **MEB** de los datos (en Tsang et al. (2007) basado en el *EB*). No obstante, en este trabajo establecen que la forma de construir el *core-set* de las propuestas anteriores *no se puede adaptar al caso de los flujos de datos* puesto que requiere inspeccionar los datos de entrenamiento **más de una vez**. De esta manera, el algoritmo que proponen comienza con una sola instancia, de tal forma que el *MEB* inicial tiene radio 0. Si llega una nueva instancia y el *MEB* actual puede cubrirla, se descarta la instancia; en caso contrario, se actualiza el centro y el radio del mismo. Aquellas instancias que sean cubiertas por el *MEB* definen el *core-set* del conjunto de datos original. Para mejorar su propuesta, plantean la utilización de un **conjunto de hiperesferas** con el objetivo de recordar más *información del pasado*. Concretamente, proponen que **todas las hiperesferas menos una tengan radio 0**, de manera que almacenan una *hiperesfera con radio distinto de 0* y las demás en un buffer como si fueran *instancias individuales*. De esta manera, cuando llega una nueva instancia, si no está cubierta por la hiperesfera con radio distinto de 0, se guarda en el buffer. Cuando se llena el buffer, se actualiza el *MEB* con las instancias del mismo. Esta extensión la denominan el algoritmo ***lookahead***.

Por otra parte, en los algoritmos propuestos tanto en Tsang et al. (2005) como en Tsang et al. (2007) los datos se procesan en modo *batch*, de tal forma que, cuando llega una nueva instancia, se tiene que volver a realizar todo el proceso de entrenamiento para adecuar el modelo a la misma; de esta manera, **no son capaces de realizar una adaptación en línea del modelo**. Para solventar esto, en Wang et al. (2010) proponen un algoritmo denominado ***online CVM*** (*OCVM*), que consta de dos fases principales. La primera de ellas se basa en un **proceso fuera de línea de eliminación de instancias** puesto que muchas de ellas son *redundantes* en el proceso de cálculo del *MEB* aproximado del conjunto original de los datos, lo que ahorra tiempo de cómputo. Para identificar las instancias redundantes, se establece un *límite superior* para la distancia entre el centro de la aproximación del *MEB* en cada iteración y el *MEB* exacto, y este límite superior se utiliza para saber qué instancias son cubiertas por el *MEB* exacto, de tal forma que la eliminación de las mismas *no afecta a la construcción del modelo final*.

Basándose en la realización de la primera fase, las instancias seleccionadas, junto con las nuevas instancias que llegan, se utilizan para llevar a cabo un **ajuste en línea del *CVM***, que es posible debido a la eliminación de instancias innecesarias. Concretamente, se realiza un *ajuste adaptativo* del *MEB*. Esta fase se compone de tres pasos, siendo el primero de ellos un **entrenamiento fuera de línea del SVM inicial con las instancias disponibles al principio** mediante la construcción del *MEB* aproximado. En el segundo paso, a medida que llegan nuevas instancias, se realiza una **expansión del *MEB*** si las instancias no son cubiertas por la hiperesfera que lo define. Una vez que se lleva a cabo este paso, en base al nuevo *MEB* aproximado, **se actualizan los coeficientes del clasificador SVM**. En la propuesta Rai et al. (2009), aunque solo se realice una revisión de cada una de las instancias, las utiliza **todas**, mientras que en ese trabajo no es necesario contemplar todo el conjunto de datos.

Otra propuesta que aborda el aprendizaje de máquinas de vector soporte para flujos de datos es la planteada en Krawczyk y Woźniak (2013), donde desarollan el algoritmo denominado ***Weighted One-Class Support Vector Machine*** (*WOCSVM*). En este trabajo abordan la clasificación *one-class* (distinguir entre los datos provenientes de una determinada distribución objetivo que pertenecen a una determinada clase y valores atípicos), al igual que lo tratan las propuestas Tsang et al. (2005) y Wang et al. (2010). El fundamento de este algoritmo es **construir una hiperesfera que encierre a las instancias pertenecientes a una determinada clase**, de tal forma que si una nueva instancia se encuentra dentro de dicha hiperesfera, se clasifica a dicha clase; en caso contrario, se identifica como *valor atípico*. De esta forma, principalmente se diferencia de la clasificación *one-class* realizada en Tsang et al. (2005) y Wang et al. (2010) en que genera una *hiperesfera*, mientras que en los otros trabajos construyen un *hiperplano*. Para realizar un aprendizaje incremental de las máquinas de vector soporte y tratar con la aparición de *concept drifts*, asignan un *peso* a cada una de las instancias para establecer su influencia en el entrenamiento del modelo. De esta forma, cuando llegan nuevas instancias, se modifican los límites de decisión definidos por la hiperesfera *realizando cambios en los pesos asignados a las instancias a medida que son menos recientes* (modelo de ventana *damped*). Para asignar los pesos iniciales de las nuevas instancias, proponen dos formas: **asignar el peso más alto posible** (peso de 1) o realizar la asignación **de forma distinta** (en base a un valor medio de las instancias existentes). Para su modificación incremental, proponen dos maneras: realizar un decremento gradual de los pesos **teniendo en cuenta su importancia inicial** o **sin tener en cuenta la importancia inicial**. El algoritmo de aprendizaje incremental que utiliza estos pesos puede ser cualquiera de los métodos incrementales propuestos para *OCSVM*.

Por otra parte, en Nathan y Raghvendra (2014), al igual que en Rai et al. (2009), proponen un algoritmo de aprendizaje en línea de máquinas de vector soporte denominado ***Blurred Ball SVM***. De la misma manera que en Rai et al. (2009), se fundamenta en formular el aprendizaje de un *SVM* como un problema *MEB* (*Minimum Enclosing Ball*), pero en este trabajo llevan a cabo una aproximación del *MEB* utilizando el concepto del *Blurred Ball cover* propuesto en K. Agarwal y Raghven-

dra (2015), que se basa en utilizar diferentes *MEBs*, de tal forma que se actualizan cuando las nuevas instancias se encuentran fuera de la unión de la expansion $1+\varepsilon$ de todos los *MEBs*. La clasificación que realizan en este trabajo es *binaria*, para lo que usan los *MEBs* previos con el objetivo de realizar un voto por mayoría en función del número de *MEBs* que contengan a la instancia a clasificar.

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
Klinkenberg y Joachims (2000)	Estimaciones $\xi\alpha$ para calcular el tamaño de ventana			Si (creo que categóricas no)	Si*		
Klinkenberg (2001)	Estimaciones $\xi\alpha$ para calcular los tamaños de ventana para instancias etiquetadas y no etiquetadas			Si (creo que categóricas no)	Si*		
CVM	No	No*		Si (creo que categóricas no)	Si (el *core-set* obtenido no depende de la dimensión de los datos)		No*
SCVM (Tsang et al. (2007))	No	No*		Si (creo que categóricas no)	Si (el *core-set* obtenido no depende de la dimensión de los datos)		No*
StreamSVM	No*			Si (creo que categóricas no)	Si*		
OCVM	No*			Si (creo que categóricas no)	Si		No*
WOCSVM	Si	Si	No*	Si (creo que categóricas no)	Si	Si	No*
Blurred Ball SVM	No*		No*	Si (creo que categóricas no)	Si		No*

Tabla 3.6: Algoritmos de aprendizaje supervisado para flujos de datos basados en máquinas de vector soporte

3.2.7. Regresión logística

En la literatura de modelos de aprendizaje automático para flujos datos la *regresión logística* no recibe mucha atención puesto que no existe una amplia gama de artículos que la traten. Una de las propuestas planteadas para adaptar la *regresión logística* a la naturaleza de los flujos de datos es la desarrollada en Anagnostopoulos et al. (2009). Concretamente, en este artículo proponen realizar una estimación de la regresión logística en línea de forma adaptativa utilizando **factores de olvido**. Debido a los problemas que surgen a la hora de encontrar los estimadores de máxima verosimilitud del modelo de *regresión logística* de forma exacta con la llegada de nuevas instancias, se propone utilizar un método de aproximación de la función de verosimilitud logarítmica denominado **aproximación de Taylor**. La aproximación que realizan *requiere que se revise todo el flujo de datos*, característica que no es adecuada en el tratamiento en línea de los mismos, de tal forma que realizan una modificación de la actualización de los parámetros que intervienen en la aproximación para que contemplen únicamente la **contribución de la nueva instancia**. A las ecuaciones que definen la actualización de los parámetros se les añade **factores de olvido**, de manera que se va disminuyendo de forma *exponencial* la influencia de las estimaciones de los parámetros en tiempos pasados. Para adaptar los factores de olvido a la presencia de *concept drifts*, utilizan un **método de descenso del gradiente estocástico**.

Por otra parte, otra propuesta que tiene en cuenta los posibles cambios en el concepto que describe a un conjunto de datos y que utiliza el modelo de *regresión logística* es la planteada en Liao y Carin (2009), donde desarrollan el algoritmo denominado ***migratory logistic regression*** (*MigLogit*). En este trabajo abordan el problema de aprender un modelo de clasificación de un conjunto de datos de entrenamiento, generada por una determinada distribución, con el objetivo de *generalizarlo a otro conjunto de instancias de testeo* que procede de una **distribución distinta a los datos de entrenamiento**. En esta propuesta, el conjunto de entrenamiento (D^a) tiene todas las instancias etiquetadas, y el conjunto de testeo (D^p) se compone de un conjunto de datos etiquetados (D^p_l) y un conjunto de datos no etiquetados (D^p_u); de esta forma, el objetivo de este trabajo es utilizar el conjunto de datos $\boldsymbol{D^a \bigcup D^p_l}$ para entrenar un clasificador que prediga las clases del conjunto $\boldsymbol{D^p_u}$.

Durante la fase de entrenamiento, debido a que D^a y D^p provienen de diferentes distribuciones, para medir la influencia de las instancias de D^a para entrenar el clasificador con el objetivo de predecir la clase de los ejemplos de D^p_u se añaden unas variables auxiliares denotadas por $\boldsymbol{\mu_i}$, que se asocian a cada una de las instancias de D^a. Específicamente, estas variables auxiliares miden el **grado de disparidad** entre cada uno de los ejemplos de D^a y el conjunto de datos D^p, de tal forma que, cuanto mayor es su valor para una determinada instancia, *mayor es dicha disparidad*, por lo que la instancia influye menos en el proceso de cálculo de los pesos del clasificador. El entrenamiento del modelo de *regresión logística* se realiza utilizando un método denominado *block-coordinate ascent*, donde **se optimizan alternativamente los pesos del clasificador y las variables auxiliares**, fijando los pesos al optimizar las variables auxiliares y viceversa. En la propuesta Anagnostopoulos et al. (2009)

utilizan factores de olvido exponencial sobre una serie de parámetros para tratar los *concept drifts*, mientras que en este trabajo utilizan variables auxiliares que miden la influencia de las instancias sobre el proceso de entrenamiento.

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
Anagnostopoulos et al. (2009)	Factores de olvido exponencial		No*	Si (creo que categóricas no)			
MigLogit	Variables auxiliares sobre las instancias		No*	Si (creo que categóricas no)			

Tabla 3.7: Algoritmos de aprendizaje supervisado para flujos de datos basados en regresión logística

3.2.8. Combinación de métodos de aprendizaje

Aparte de las ventajas que supone la utilización de *una combinación de métodos aprendizaje* frente al uso de un solo clasificador en tareas de predicción, una de las características más importantes de estos métodos con respecto a la clasificación de flujos de datos es su gran capacidad para tratan con el problema de la aparición de **concept drifts**. Su gran relevancia en el manejo de la evolución de la distribución subyacente a los datos ocasiona que su presencia en la literatura de aprendizaje automático para flujos de datos sea *alta*. Esto se refleja en la existencia de revisiones extensas que abordan la combinación de métodos de aprendizaje para flujos de datos, como se puede apreciar en Gomes et al. (2017a) y en Krawczyk et al. (2017)). Debido a que estas revisiones incluyen un gran número de propuestas, en esta sección se van a abordar otros trabajos que *no están incluidos en dichas revisiones* con el objetivo de **complementar las mismas**.

La mayor parte de los algoritmos que tratan la combinación de métodos de aprendizaje utilizan una aproximación **basada en fragmentos** (*chunk-based*), es decir, dividen el flujo de datos en *fragmentos* y entrenan un *clasificador* para cada uno de ellos; para llevar a cabo la clasificación de una nueva instancia, *combinan los resultados de los modelos* entrenados en cada uno de los fragmentos. Este tipo de técnicas, sean capaces o no de detectar la aparación de nuevas clases, no tienen la capacidad de detectar **clases recurrentes**, es decir, clases que *aparecen en el pasado*, *desaparecen por un largo periodo de tiempo* de manera que los modelos construidos dejan de contemplarla y *reaparece en el flujo de datos*. Si los métodos basados en fragmentos *no detectan nuevas clases*, si una etiqueta deja de existir y desaparece de los modelos, si vuelve a aparecer, **ninguno de los modelos es capaz de detectarla**; en el caso de aquellos que *si pueden detectar las nuevas clases*, las clases

recurrentes las tratan como **clases nuevas**, lo que conlleva un gasto computacional innecesario y un incremento del error de clasificación. Ante estos inconvenientes, en Al-Khateeb et al. (2012) proponen el algoritmo denominado ***CLAss-based Micro classifier ensemble*** (*CLAM*) que, en lugar de ser un algoritmo *chunk-based*, se define como *class-based*, que se fundamenta en la idea de mantener, por cada una de las clases vistas hasta un determinado momento, un ***ensemble* de un número fijo de micro-clasificadores**. En este trabajo el objetivo es *detectar nuevas clases de forma eficiente* y *distinguir entre clases recurrentes y nuevas clases*.

El algoritmo propuesto en este trabajo comienza construyendo un **número determinado de micro-clasificadores iniciales** para cada una de las clases que constituyen el *ensemble* de cada una de ellas. Para construir un micro-clasificador correspondiente a una determinada clase, se escoge un fragmento de datos de entrenamiento, se escogen aquellas instancias que pertenecen a esa clase y, utilizando dichas instancias, se utiliza el algoritmo *k-means* para construir un conjunto de k *micro-clusters*, donde cada uno de ellos representa una **hiperesfera** y la unión de ellas representa el *límite de decisión del micro-clasificador*. Para mantener actualizado los *ensembles* de cada una de las clases, cuando llega un nuevo fragmento de datos, se construye un micro-clasificador para cada una de las etiquetas *teniendo en cuenta las instancias de dicho fragmento pertenecientes a cada una de las clases*, se incluyen dentro de los respectivos *ensembles* y, en cada uno de estos *ensembles*, se elimina aquel micro-clasificador que obtenga el **mayor error de clasificación en el nuevo fragmento de datos**.

Para llevar a cabo la predicción de una nueva instancia, se comprueba si la misma se incluye dentro del límite de decisión definido por cada uno de los *ensembles*, que corresponde a la unión de los límites decisión establecidos por los micro-clasificadores que los componen. Una vez se averiguan los *ensembles* que cubren a la instancia, por cada uno de estos se halla la **mínima distancia entre la instancia y los *micro-clusters* de cada uno de los micro-clasificadores** que los constituyen. Tras esto, se asigna a la instancia la clase a la que pertenece el *ensemble* cuya distancia mínima sea la **menor entre todos los *ensembles***. Puede ocurrir que algunas instancias no sean cubiertas de por ningún ensemble; en este caso, se consideran *universal outliers*, de tal forma que se almacenan y se revisan periódicamente con el objetivo de comprobar si hay muchos *outliers* que están juntos puesto que puede indicar que hay presente una **nueva clase** (los *outliers* no están etiquetados). Para realizar esta comprobación, utilizan una métrica denominada **q-Neighborhood Silhouette Coefficient**, que calcula una serie de micro-clusters de *outliers*, su distancia a los *micro-clusters* de los micro-clasificadores existentes y, en base a estas distancias, unos pesos cuya suma, si es mayor que un determinado valor, establece que existe una nueva clase y se asigna la misma a las instancias no etiquetadas que pertenecen a esa nueva clase. En este trabajo, las *clases recurrentes* se identifican como **clases existentes**.

Otra propuesta de *ensemble* que se basa en cada una de las clases por separado para crear clasificadores, al igual que en Al-Khateeb et al. (2012), es la planteada en Czarnowski y Jędrzejowicz (2014), donde desarrollan el algoritmo denominado

Weighted Ensemble with one-class Classification based on Updating of data chunk (*WECU*). Este trabajo se fundamenta en la idea de descomponer un problema multiclase en un *ensemble* de clasificadores que se centren en *identificar las instancias de cada una de las clases por separado* (***one-class classification***). Concretamente, construyen una matriz de tamaño fijo en la que *guardan clasificadores para cada una de las clases de diferentes instantes de tiempo*. Para construir los clasificadores de un instante de tiempo determinado a partir de un fragmento de datos S_t, se crea un subconjunto S_t^l por cada clase l presente en los datos, que está compuesto por un conjunto de instancias positivas PS_t^l y un conjunto de instancias negativas US_t^l; de esta manera, se construye un clasificador para la clase l en ese instante de tiempo utilizando PS_t^l, es decir, utilizando **solo instancias positivas** (al igual que en Al-Khateeb et al. (2012)).

A diferencia del trabajo realizado en Al-Khateeb et al. (2012), en esta propuesta **se asigna un peso a cada uno de los clasificadores**, que mide *su influencia en la clasificación de una nueva instancia* y se calcula a partir de su desempeño y del tiempo que lleva incluido en la matriz. La actualización del modelo de *ensemble* se lleva a cabo normalmente **sustituyendo los clasificadores más antiguos de la matriz por los nuevos** obtenidos del nuevo fragmento de datos, excepto si se considera que los clasificadores más antiguos siguen siendo importantes en el *ensemble*; si la suma de los pesos de los clasificadores más antiguos es *mayor que la media de los pesos* del *ensemble*, se mantienen en el modelo y se busca otro conjunto de clasificadores de otro instante de tiempo para llevar a cabo la sustitución. La clasificación de una nueva instancia se determina a través del **voto por mayoría ponderada** y, a diferencia del trabajo realizado en Al-Khateeb et al. (2012), en los experimentos utilizan *árboles decisión* como clasificador base, aunque se pueden utilizar otro tipo de clasificadores.

Las propuestas anteriores están diseñadas para trabajar en entornos donde las instancias llegan en *fragmentos* y para evaluar los componentes de los *ensembles* que construyen *de forma periódica*, sustituyendo los clasificadores más débiles de los mismos por otros nuevos tras recibir un bloque de instancias. Los algoritmos que se basan en estas ideas son capaces de adaptarse a *concept drifts* graduales, pero no a ***concept drifts*** **abruptos**. A diferencia de los algoritmos de *ensemble* basados en bloque, existe otro tipo denominado *online ensemble*, que se basa en actualizar los componentes de los *ensembles* cada vez que se recibe una nueva instancia, pero tiene un **coste computacional alto** y **no introducen nuevos clasificadores periódicamente**. Para solventar estos inconvenientes, en Sun et al. (2016) proponen un algoritmo de *ensemble* denominado ***Adaptive Windowing based Online Ensemble*** (*AWOE*), que se fundamenta en la combinación de la evaluación periódica de los componentes del *ensemble* con el objetivo de introducir nuevos clasificadores en el mismo y realizar actualizaciones incrementales a medida que llegan nuevas instancias (un *ensemble* híbrido de los dos tipos de *ensemble*). La finalidad de esto es mejorar las reacciones del *ensemble* tanto a *concept drifts* **graduales** como **abruptos**.

En este trabajo, en todo momento se mantiene un número fijo de clasificadores

dentro del *ensemble* que, a diferencia de las propuestas anteriores, **no se entrenan para cada una de las clases**. El *ensemble* que proponen incluye un **detector de *concept drifts* basado en una ventana adaptativa**; para ello, cada vez que se añade una instancia a la ventana utilizan la distancia *Kullback-Leibler* con el objetivo de medir la diferencia entre dos *subventanas* de instancias de igual tamaño que componen la ventana total, de tal forma que si la distancia es mayor que el valor establecido por el *Hoeffding bound*, se detecta un *concept drift* y se elimina la subventana que es menos reciente. El detector de *concept drifts* se utiliza para **establecer el tamaño del bloque de instancias con el que se va a entrenar cada uno de los clasificadores** del *ensemble*.

Para llevar a cabo el entrenamiento de los clasificadores del *ensemble*, a medida que llegan nuevas instancias, éstas se van almacenando en un buffer B, de manera que si el buffer se llena o se detecta un *concept drift*, entonces *se entrena un nuevo clasificador sobre las instancias incluidas en el mismo*; de esta manera, el bloque de datos con el que se entrena cada clasificador es *distinto*. Si en el *ensemble* no caben más clasificadores, al igual que en Al-Khateeb et al. (2012), se sustituye aquel que tenga peor desempeño en ese momento por el nuevo clasificador. Además de los clasificadores que componen el *ensemble*, en el algoritmo se construye de forma incremental un *online learner* con todas las instancias que vienen del flujo de datos con el objetivo de **incluir los ejemplos más recientes en la realización de la predicción de una nueva instancia**; si se detecta un *concept drift*, el *online learner* se reinicializa con B y se sigue entrenando con las instancias que vienen posteriormente. La clasificación se realiza por la **regla de votación por mayoría ponderada** (cada clasificador tiene asociado un peso que se calcula cada vez que llega una nueva instancia y tiene en cuenta unos errores de predicción de cada clasificador). En los experimentos utilizan como clasificador base el *Hoeffding Tree*, aunque se puede utilizar cualquier otro algoritmo de aprendizaje en línea.

Por otro lado, en Gomes et al. (2017b) se propone el algoritmo denominado ***adaptive random forests*** (ARF), que se basa en adaptar el algoritmo tradicional *Random Forest* (Breiman (2001)) para realizar tareas de clasificación de flujos de datos. El algoritmo *Random Forest* se basa en combinar un conjunto de árboles de decisión, en el caso de este trabajo *Hoeffding trees*, de manera que dicho conjunto se construye **utilizando la idea del *bagging*** y **realizando selecciones aleatorias de un subconjunto de los atributos que describen a las instancias**. El algoritmo propuesto en este trabajo crea un número determinado de *Hoeffding trees* iniciales, y cada uno de ellos tiene asignado un **peso** que mide su influencia en la clasificación de una nueva instancia. A continuación, cada vez que llega una instancia del flujo de datos, par cada uno de los *Hoeffding Trees* se actualiza **su peso** en base a *la predicción que realiza sobre esa instancia* y **su estructura** utilizando la misma. Para actualizar la estructura de cada *Hoeffding tree*, utilizan una **versión del *bagging* en línea** (Oza y Russell (2001)); en el *bagging* tradicional, si el tamaño del conjunto de entrenamiento tiende a infinito, la probabilidad de que una instancia sea elegida un número k de veces para que forme parte del conjunto de entrenamiento de un clasificador del *ensemble* se distribuye como una **Poisson**. De esta forma,

la actualización de un *Hoeffding tree* con una instancia se realiza utilizandola un número k de veces según una distribución de *Poisson*, concretamente para actualizar los contadores del nodo hoja correspondiente.

Tras llevar a cabo la actualización de los *Hoeffding trees*, se aplica la detección de *concept drifts*. Para ello, en este trabajo se da la posibilidad de poder utilizar cualquier tipo de detector, aunque en los experimientos emplean *ADWIN* y *Page Hinkley Test*. Basándose en el detector, en esta propuesta distinguen entre si se da un ***warning*** o si el ***concept drift ocurre***. Si se produce un *warning*, se comienza a crear un **árbol de decisión alternativo**, de tal forma que si más adelante se detecta un *concept drift* se sustituye el árbol de decisión actual por el alternativo (en la propuesta planteada en Hulten et al. (2001) también crean árboles alternativos cuando se da la presencia de *concept drifts*, pero son subárboles en lugar de un nuevo árbol). Para llevar a cabo la tarea de predicción de una nueva instancia, se utiliza una **votación ponderada** usando los *Hoeffding trees* construidos por el *Random Forest*.

Por otra parte, en Sun et al. (2019) proponen un algoritmo de *ensemble* que, a diferencia de las otras propuestas, trata con flujos de datos **multi-etiqueta**, y se denomina ***Multi-Label ensemble with Adaptive Windowing*** (*MLAW*). Este trabajo se basa en ciertos aspectos en la propuesta realizada en Sun et al. (2016). El algoritmo que proponen, para detectar cambios en el concepto que describe a los datos, utiliza un método basado en **dos ventanas**, que se basa en el empleo de la *divergencia Jensen-Shannon* para medir la *disimilitud entre dos ventanas de ejemplos*, una representando instancias menos recientes y otra de instancias más recientes; utilizan esta divergencia en lugar de la distancia *Kullback-Leibler* debido principalmente a que la primera tiene la propiedad de ser *simétrica*, mientras que la segunda no. Para llevar a cabo la detección de *concept drifts* tanto *repentinos* como *recurrientes*, primero se inicializan las dos ventanas con el mismo número de instancias, una ventana detrás de otra. Una vez realizado esto, cada vez que llega una nueva instancia, se comprueba la *divergencia Jensen-Shannon* entre las dos ventanas; si esta medida es menor o igual que un umbral definido por el *Hoeffding bound*, se desplaza la segunda ventana una instancia en el flujo de datos y se continúa con la siguiente instancia; en caso contrario, se detecta un *concept drift* y se vuelven a definir las dos ventanas a partir de la instancia del instante de tiempo en el que se produce el mismo. En el caso específico en el que la divergencia sea igual a 0, se establece que se ha encontrado un *concepto recurrente*.

En el algoritmo *MLAW*, cada una de las instancias del flujo de datos se van almacenando en la ventana que incluye a las dos ventanas comentadas anteriormente. Si se detecta un *concept drift*, **se crea un nuevo clasificador**, en este caso *Multi-label Hoeffding Trees* (aunque se puede utilizar otro tipo de clasificador que aborde el aprendizaje en línea), a partir de la segunda ventana; en el entrenamiento de los *Multi-label Hoeffding Trees* se tienen en cuenta **dependencias entre etiquetas** y, para mejorar el rendimiento del clasificador, llevan a cabo una poda de las combinaciones de etiquetas que no son usuales mediante el método **pruned sets** (*PS*), utilizado en los nodos hoja de los árboles. Tras construir el clasificador, si existe es-

pacio en el *ensemble* y el *concept drift* detectado no es recurrente, se introduce en el mismo. En el caso de que no exista espacio, se elimina del *ensemble* aquel que tenga el **peor desempeño en ese momento**. En el caso de que el *concept drift* detectado sea recurrente, se contempla la reutilización de clasificadores dentro del *ensemble*. La tarea de clasificación se realiza mediante **voto ponderado** de los clasificadores del *ensemble*; estos pesos se utilizan para tratar con *concept drifts* **graduales** y se actualizan cada vez que llega una nueva instancia, calculándose de la misma forma que en Sun et al. (2016).

Algoritmo	Clasificador base	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
CLAM	Micro-clasificadores compuestos por *micro-clusters*	Adición de nuevos micro-clasificadores y eliminación de otros cuyo desempeño de clasificación no sea bueno	Si		Si (creo que categóricas no)	Si*	Si*	
WECU	Árboles de decisión (aunque se pueden utilizar otros tipos de clasificadores)	Sustitución de clasificadores de un instante de tiempo (normalmente los más antiguos) por otros nuevos	No*		Si	Depende del clasificador utilizado*		
AWOE	Hoeffding Tree (aunque se pueden utilizar otros tipos de clasificadores)	Ventana adaptativa con la distancia *Kullback-Leibler* y el *Hoeffding bound*			Si (creo que categóricas no)	Depende del clasificador utilizado*	Si	
ARF	Hoeffding tree	Cualquier método de detección de *concept drifts* (en los experimentos utilizan *ADWIN* y *Page Hinkley Test*)			No	Si	Si*	
MLAW	Multi-label Hoeffding Trees (aunque se puedn utilizar otros tipos de clasificadores)	Pesos de los clasificadores y divergencia de *Jensen-Shannon* en modelo de ventana deslizante			Si (creo que categóricas no)	Depende del clasificador utilizado*	Si*	

Tabla 3.8: Algoritmos de aprendizaje supervisado para flujos de datos basados en la combinación de métodos de aprendizaje

3.3. Algoritmos de aprendizaje no supervisado

Debido a la gran popularidad de los algoritmos de **agrupamiento** para realizar clasificación no supervisada, la mayor parte de la literatura relacionada con el desarrollo de modelos de aprendizaje automático que abordan el paradigma de aprendizaje no supervisado sobre flujos de datos se centra en dicho tipo de algoritmos. De esta manera, existen varias revisiones sobre métodos de agrupamiento para flujos de datos (Nguyen et al. (2015), Silva et al. (2014), Aggarwal (2013), Sharma et al. (2018), Mansalis et al. (2018)). En este trabajo vamos a centrarnos en propuestas que abordan **métodos de agrupamiento particionales y jerárquicos**, además de crear, al igual que para los algoritmos de aprendizaje supervisado, **una tabla comparativa entre las mismas** y exponer algunas propuestas más recientes.

3.3.1. Agrupamiento

Una de las propuestas más populares para llevar a cabo un agrupamiento de grandes cantidades de datos y que utilizan muchos otros trabajos para desarrollar sus algoritmos es la planteada en Zhang et al. (1996), donde proponen el algoritmo denominado **BIRCH**. Este algoritmo se fundamenta en la utilización de una estructura denominada ***Clustering Feature*** (*CF*), cuya función es almacenar información de un *cluster* de forma compacta. En esta estructura se almacenan tres campos: el *número de instancias del cluster*, la *suma de las instancias* y la *suma de los cuadrados de las instancias*; a partir de estos campos, se pueden obtener una serie de parámetros necesarios para el proceso de agrupamiento, como el **centroide** del *cluster*, el **radio** y el **diámetro**. Los *clustering features* que representan a los *clusters* en este algoritmo se organizan en una estructura de árbol denominada **CF Tree**. Cada uno de los nodos intermedios del árbol representa un *cluster* que esta compuesto por un número determinado de *subclusters*, cada uno de ellos representado por un *CF* y un puntero a un nodo hijo, de tal forma que cada uno de estos *subclusters* representa un nodo intermedio o un nodo hoja en un nivel inferior del árbol. La estructura de los nodos hojas es similar a la de los nodos intermedios, solo que debe cumplir una restricción relacionada con el diámetro del *cluster* que representa, que **no puede superar un determinado umbral** y el número de *subclusters* que pueden representar los nodos hoja es distinto al de los nodos intermedios.

A la hora de insertar una nueva instancia, se recorre el árbol calculando qué *cluster*, representado en cada nodo, es el más cercano a la instancia (utilizando el **centroide** del *cluster*) hasta que llega a un nodo hoja. Si un *CF* de un *subcluster* del nodo hoja puede absorber la instancia **sin violar la restricción del umbral**, ésta se incluye dentro del *subcluster*. En caso contrario, se crea un nuevo *subcluster* para incluir esa instancia. Si no existe espacio para crear un nuevo *subcluster*, es necesario **dividir el nodo hoja**, convirtiendo a éste en un nodo intermedio; para ello, se escogen los dos *subclusters* del nodo hoja cuya distancia entre ellos sea la mayor y **se crean dos nodos hojas** (hijos del antiguo nodo hoja), de manera que

el resto de los *subclusters* se acoplan en esos nuevos nodos hoja teniendo en cuenta los dos *subclusters* anteriores como *semillas*. Tras este proceso, se actualizan los nodos intermedios por los que pasa la nueva instancia para añadir las características de la misma al árbol y para los posibles cambios que pueda haber realizado en la estructura.

La propuesta anterior tiene el inconveniente de que no tiene **garantías teóricas** en su desempeño con respecto a grandes cantidades de datos. Por ello, un algoritmo que si tiene dichas garantías para tratar con flujos de datos es el denominado **STREAM**, propuesto en O'Callaghan et al. (2002). En este trabajo se centran en resolver el problema de agrupamiento ***k*-medianas**, una variante del algoritmo *k-medias* que se basa en encontrar las k medianas (centros de los *clusters*) que minimicen las distancias entre cada una de las *instancias* y *su mediana más cercana*. Este problema es *NP-duro*, de tal forma que en esta propuesta plantean obtener un aproximación del mismo utilizando un algoritmo denominado **LSEARCH**, un algoritmo de búsqueda local que se fundamenta en resolver una variante del k-medianas denominado ***facility location***. Este problema se diferencia del k-medianas en que no se especifica el número de *clusters* a construir, sino que **se asigna un coste a cada una de las medianas** o *facilities* y se minimiza una función que tiene en cuenta esos costes, que multiplican al número de medianas que haya en cada iteración.

Para resolver el problema de *k-medianas* a partir del problema de *facility location*, utilizan el algoritmo *LSEARCH* con el objetivo de **buscar los costes que den lugar a la construcción de *k* centros**. De esta manera, comienzan con una solución inicial y, mientras no se obtengan k medianas y el parámetro de control ϵ permita seguir mejorando la solución, se van realizando diferentes llamadas a la resolución del problema de *facility location*, utilizando la solución de un problema para resolver el siguiente. El algoritmo *LSEARCH* se aplica en *cada uno de los fragmentos de datos del flujo de datos*, de tal forma que, de cada uno de ellos, se obtienen un *conjunto de medianas* y en memoria solo se mantienen dichas medianas. Cuando la memoria se llena, se vuelve a aplicar el algoritmo *LSEARCH* sobre las medianas anteriores para obtener un *grupo de medianas más reducido*.

En la propuesta anterior, al realizar la unión de los *clusters*, **no se pueden dividir los clusters** cuando la evolución del flujo de datos lo requiera en una etapa posterior; de esta manera, los *clusters* se van construyendo en base a *todo el flujo datos*, lo que puede ser un inconveniente si la información menos reciente no es relevante para el desempeño del modelo en instantes de tiempo posteriores. Por lo tanto, no tiene en cuenta la naturaleza dinámica de los datos y, en este aspecto, en Aggarwal et al. (2003) se propone el algoritmo denominado **CluStream**, que está compuesto por dos componentes. El primero corresponde a un componente *en línea* cuya función es **almacenar de forma periódica información compacta**, y el segundo es un componente *fuera de línea* que utiliza las estructuras proveniente del primer componente para **producir información de más alto nivel**.

Para compactar la información proveniente del flujo de datos en el componente *en línea*, se utilizan unas estructuras denominadas **micro-clusters**, que son una

extensión temporal de la estructura *cluster feature vector* que proponen en Zhang et al. (1996). El *micro-cluster* se adelantó al abordar la propuesta Aggarwal et al. (2006), solo que en este caso no se añade la *etiqueta clase* al *micro-cluster*; de esta manera, en este trabajo un *microcluster* está definido por un vector en el que se almacena la *suma de los cuadrados de cada uno de los atributos de las instancias*, la *suma de cada uno de los atributos de las instancias*, la *suma de los cuadrados de los instantes de tiempo en los que llegaron cada una de las instancias*, la *suma de los instantes de tiempo en los que llegaron cada una de las instancias* y el *número de instancias que representa el microcluster*. Estas estructuras están almacenadas en diferentes instantes de tiempo denominados ***snapshots***, que se almacenan en diferentes niveles de granularidad temporal en una estructura denominada ***pyramidal time frame***, de tal forma que en los instantes de tiempos recientes la granularidad temporal es *mayor*. Inicialmente se crean un número determinado de *micro-clusters* con el algoritmo *k-means* y, cuando llega una nueva instancia, se intenta asignar a un *micro-cluster* existente; en el caso de que no sea posible debido a que supera el límite máximo de todos los *micro-clusters*, obtenido por la información almacenada en ellos, **se crea un nuevo micro-cluster**. En el caso de que no se puedan crear más *micro-clusters*, se contempla si es seguro eliminar uno de ellos teniendo en cuenta la *información temporal de los mismos*; si no se puede llevar a cabo la eliminación de un *micro-cluster*, se procede a la unión de los dos *micro-clusters* existentes más cercanos.

En base a las estructuras construidas en el componente *en línea*, en el componente *fuera de línea* se crean **agrupamientos de mayor nivel** con el algoritmo *k-means*. Este componente requiere que se le pase como entrada, aparte de las estructuras del componente *en línea*, el **número de *clusters* que se quiere obtener** y el **horizonte temporal a partir del cuál se quieren calcular** (intervalo de tiempo). Para crear los *clusters* en un horizonte temporal h determinado, se utiliza la estructura *pyramidal time frame*, de tal forma que se realiza la resta entre los *clusters* presentes en el instante de tiempo actual t y los *clusters* que existían en un instante de tiempo justo anterior a $t - h$.

Otra propuesta que aborda un método de agrupamiento particional es el planteado en Ackermann et al. (2010), en el que desarrollan el algoritmo denominado **StreamKM++**. Este algoritmo se fundamenta en la utilización de *coresets*, que están constituidos por un *subconjunto de instancias del conjunto original* de tal forma que el coste del agrupamiento que se haga sobre ellos es una **aproximación** del coste de agrupar los datos del conjunto original con un error pequeño; así, en este trabajo realizan el agrupamiento de los datos del *coreset* con el objetivo de obtener una **solución aproximada para el conjunto de datos original de forma eficaz**. Para llevar a cabo la construcción de los *coresets*, emplean un método similar a *k-means++* (algoritmo que se utiliza para la selección eficiente de los valores iniciales o *semillas* en el algoritmo *k-means*) y, para hacerlo más eficiente, utilizan una estructura de datos denominada *coreset-tree*, que permite reducir el espacio de búsqueda de los *coresets*. Esta estructura es un árbol binario que se relaciona con un *agrupamiento jerárquico divisivo* de todo el conjunto de datos, de tal forma que

los representantes de las instancias incluidas en cada una de las hojas del árbol representan **los puntos del *coreset***.

En esta propuesta, para mantener un pequeño *coreset* sobre el que llevar a cabo el agrupamiento particional, utilizan una técnica denominada ***merge and reduce***. En esta técnica se mantienen un conjunto de *cubos*, de tal forma que en cada uno de ellos se van introduciendo instancias. Cuando se llenan, se procede a **unir cubos** y se aplica una **técnica de reducción** sobre los mismos, que consiste en construir los *coresets* empleando la estructura de datos *coreset-tree* mencionada con anterioridad. De esta forma, se puede obtener en cualquier instante de tiempo un agrupamiento de k *clusters* a partir del *coreset* utilizando el algoritmo *k-means* (se van construyendo diferentes *coresets* cuando se unen cubos pero en cada instante de tiempo se mantiene uno solo, que se forma en base a *coresets* anteriores). Este trabajo tiene similitud con la propuesta planteada en Aggarwal et al. (2003) puesto que el proceso de agrupamiento se realiza sobre información más compacta; no obstante, en Aggarwal et al. (2003) se realiza sobre estructuras que *resumen un conjunto de puntos*, mientras que en este trabajo se realiza sobre un *subconjunto de los datos*. Además, al crear los *coresets* a partir de otros *coresets*, se está incluyendo información del pasado que podría afectar negativamente al desempeño del algoritmo debido a la aparición de *concept drifts*, al igual que en O'Callaghan et al. (2002).

Por otra parte, otra propuesta relacionada con agrupamiento de flujos de datos es la planteada en Fichtenberger et al. (2013), donde se propone el algoritmo denominado **BICO**, que combina las estructuras de datos utilizadas en la propuesta Zhang et al. (1996) con la utilización de los *coresets* vistos en Ackermann et al. (2010). Concretamente, *BICO* mantiene una estructura de árbol similar a *BIRCH*, pero en cada uno de los nodos mantiene un **representante**, de forma similar al *StreamKM++*. A la hora de introducir una nueva instancia en la estructura de árbol, la idea de *BICO* es mejorar la toma de decisión de donde colocar dicha instancia de tal forma que se **minimice el coste del conjunto de instancias representadas por cada uno de los *CF*** (la suma de los cuadrados de las distancias de las instancias al centroide). De esta forma, cuando llega una nueva instancia, el algoritmo busca en el nivel actual el *cluster* más cercano a la misma y, si se encuentra fuera de un radio R establecido como umbral con respecto al centroide más cercano o si no hay ningún *CF* en el nivel actual, **se crea un nuevo *CF* como un nodo hijo** del padre del *CF* actual más cercano. En caso contrario, si al añadir la instancia al *CF* más cercano el coste del *CF* no supera un umbral T, **se asigna la instancia al mismo**; en caso contrario, se va a los nodos hijos del *CF* actual más cercano a la nueva instancia y se repite el proceso anterior.

En el caso de que el árbol crezca enormemente, **se duplica el umbral de coste** T y **se reconstruye el árbol** uniendo *subclusters* cuyo coste al fusionarse esté por debajo de T. En el algoritmo *BIRCH*, al llevar a cabo la reconstrucción del árbol, puede que tenga como resultado uno diferente al que se hubiera obtenido con el umbral modificado que utiliza el algoritmo pero desde el principio; en cambio, en el algoritmo *BICO*, al modificar el umbral de coste para disminuir el tamaño del árbol, la reconstrucción que llevan a cabo es **más cercana al que se hubiera**

construido con el umbral modificado desde el principio que en *BIRCH*. El algoritmo *BICO* no calcula una solución, sino que representa las instancias del flujo de datos de forma compacta; para obtener un agrupamiento de los datos, utilizan el algoritmo *k-means++* sobre un *coreset* compuesto por **los centroides de todos los *CF* presentes en el árbol**.

En las propuestas anteriores es necesario establecer el número k de *clusters* a obtener; no obstante, los conceptos que subyacen a los flujos de datos pueden cambiar, por lo que esta estrategia *no es adecuada para las características de estos datos*. Por lo tanto, en Anderson y Koh (2015) proponen el algoritmo denominado **StreamXM**, que no requiere una selección previa del número de *clusters* a construir; concretamente, proponen dos variantes: *StreamXM with Lloyds* y *StreamXM*. En el primero, se compacta un fragmento de instancias en un *coreset* de forma parecida al *StreamKM++* y, tras esto, se aplica sobre el *coreset* el algoritmo denominado **X-means** con el objetivo de **hallar el agrupamiento óptimo dado un rango de posibles valores para** k; para ello, va realizando particiones recursivas con k igual a 2 y utilizando la métrica *BIC* (*Bayesian Information Criterion*) para evaluar que partición es la más adecuada. Tras llevar a cabo esto, se ejecuta el algoritmo *k-means++* en el *coreset* cinco veces (por aspectos de rendimiento) con el valor k igual al obtenido con el algoritmo *X-means*. Se escoge el agrupamiento de k clusters de la ejecución que **mejor coste tenga de las cinco ejecuciones del algoritmo *k-means++* y de la ejecución del *X-means***.

La segunda variante es similar a la primera, pero en este caso no utilizan el algoritmo *k-means++*. En su lugar, aplican el algoritmo *X-means* sobre el *coreset* **un número de iteraciones determinado**, de manera que esto permite obtener diferentes agrupamientos sin tener establecido un número k de *clusters*, como ocurría en la primera variante; se escoge el agrupamiento que tenga **el mínimo coste**. En las dos variantes el algoritmo se aplica para cada uno de los fragmentos de datos, y se utiliza la técnica ***merge reduce***, como en la propuesta *StreamKM++*.

Otra propuesta en la que abordan un agrupamiento particional con el algoritmo *k-means* es el planteado en Liberty et al. (2016) donde proponen dos algoritmos *k-means*, uno ***semi-online*** y otro ***fully online***. En el primero, se conoce el *número de instancias* que van a llegar y un *límite inferior* para el valor de la solución óptima y, al igual que en O'Callaghan et al. (2002), abordan el problema del agrupamiento como un problema de *facility location*, de tal forma que se establecen costes bajos al principio para permitir la presencia de muchos *clusters* (cada vez que se abre un *facility centre* es como crear un nuevo *cluster*) y, a medida que transcurre el tiempo, se va incrementando el coste para que las instancias se asignen a alguno de los *clusters* presentes. Es decir, en el caso en el que el algoritmo detecte que se abren muchos *facilites*, quiere decir que el coste establecido es alto, por lo que se **dobla** el coste para que haya menos *facilities*. En el algoritmo *fully online*, el número de instancias es **desconocido**, y el límite inferior de la solución óptima se tiene que generar utilizando un **pequeño conjunto inicial de las instancias del flujo de datos** para definirlas como centroides independientes. En este trabajo establecen una serie de garantías de la aproximación de la solución que llevan a cabo; en el caso

del algoritmo *fullly online*, se garantiza que se obtiene un coste cercano al óptimo que se obtendría aplicando un *k-means* fuera de línea con un límite superior del número de *clusters* creado que tiene en cuenta el número de k clusters deseado.

Con respecto al trabajo desarrollado en Zhang et al. (1996), existe una propuesta que también utiliza una estructura de árbol balanceada en la que se almacenan diferentes *clustering features* que representan agrupamientos de objetos en diferentes niveles (una jerarquía de *microclusters* con diferentes niveles de granularidad), y es la planteada en Kranen et al. (2009), donde proponen el algoritmo denominado ***ClusTree***. Este algoritmo es **no paramétrico**, de tal forma que no es necesario especificar el número de *clusters* a encontrar puesto que se adapta dinámicamente; es decir, no hay que fijar el tamaño del modelo, como si ocurre por ejemplo en Aggarwal et al. (2003), donde se limita el número máximo de *microclusters*. Además, a diferencia de las propuestas vistas hasta ahora, en este trabajo proponen un algoritmo ***anytime***, es decir, que es capaz de dar una solución de calidad **en cualquier momento** aunque sea interrumpido y, si se dispone de más tiempo, de **refinar el modelo**; de esta forma, el algoritmo tiene la capacidad de manejar flujos de datos con **diferentes velocidades de llegada**. Concretamente, si llega una nueva instancia mientras se está insertando en la estructura jerárquica una instancia anterior que no ha llegado a un nodo hoja para insertarlo en un *microcluster*, con el objetivo de insertar la nueva instancia en ese momento, se *para el proceso de inserción de la anterior instancia*. En esta interrupción, se almacena la instancia anterior en un ***buffer*** asociado al nodo donde se quedó en el proceso de inserción (cada nodo interno del árbol tiene un *buffer*) y se prosigue con la siguiente instancia; en el futuro, cuando se vuelva a acceder con el proceso de inserción de una nueva instancia al nodo donde se quedó almacenada la instancia en el *buffer*, esta última **tendrá la oportunidad de seguir descendiendo** (juega el papel de *hitchhiker*, hacer dedo). De esta forma, la utilización de este *buffer* permite al algoritmo adaptarse a la *velocidad del flujo de datos* insertando instancias en cualquier momento y proporcionar **una solución de buena calidad en cualquier momento**.

Los *buffers* de los nodos anteriores, en lugar de trabajar con instancias, manejan **agregaciones locales** para mejorar la velocidad del algoritmo. El problema de estas agregaciones es que pueden haber instancias que no sean similares y se encuentren en la misma agregación local, de tal forma que, al descender esta agregación, haya información de la misma que deba pertenecer a un subárbol distinto al que desciende dicha agregación. De esta forma, para que estas agregaciones locales estén formadas por instancias similares, se define **una distancia máxima entre instancias**, de tal manera que *se crean nuevas agregaciones locales cuando se viole esa condición*. Por otra parte, para mantener el modelo actualizado, utilizan una **función de desvanecimiento** que tiene en cuenta información temporal en los nodos internos del árbol para disminuir la influencia de datos menos recientes. Esta *función de desvanecimiento* se utiliza en el cálculo de la información incluida en los *clustering features* y también se utiliza para evitar **divisiones de nodos** eliminando entradas de los mismos que no realizan una contribución relevante al algoritmo; estas divisiones, si no se dispone de tiempo, **se unen las dos entradas del nodo hoja más cercanas**.

También proponen para detectar *concept drifts* y *outliers* utilizar aproximaciones propuestas en la literatura, como el *pyramidal time frame* (Aggarwal et al. (2003)). El agrupamiento resultante del *ClusTree* son los *clustering features* almacenados en los nodos hoja (componente *en línea*), sobre los que se puede aplicar un algoritmo de k centros u otro tipo de algoritmo que encuentre *clusters* de diferentes formas (componente *fuera de línea*).

Por otra parte, en la literatura de algoritmos de agrupamiento para flujos datos existe una propuesta que extiende el algoritmo *ClusTree*, y es la planteada en Kranen et al. (2011), donde proponen el algoritmo denominado **LiarTree**. Una de las extensiones que se desarrollan en este trabajo es **lidiar con solapamientos de las entradas (*clusters*) de los nodos internos**. En el proceso de inserción de una nueva instancia, si se detecta este solapamiento, se utiliza un método local de *look ahead* **mirando los nodos hijos más cercanos de los nodos padres**, **intercambiando los padres de esos nodos hijos** y, si se reduce el radio cubierto por los nodos padres, **se actualizan los *clustering features* de los nodos padres** y se prosigue con la inserción utilizando dicha actualización. Otra extensión que realizan es el **manejo de ruido** puesto que en el algoritmo *ClusTree* todas las instancias se tratan igual.

Para manejar el ruido, añaden en cada nodo interno del árbol un ***buffer* de ruido**, que almacena instancias que se consideran *ruidosas* con respecto al nodo en el que se encuentren en cada momento utilizando una probabilidad que se calcula con información del nodo y viendo si supera un determinado umbral. La agregación de estas instancias ruidosas puede dar lugar a la aparición de un *nuevo concepto* representado con un nuevo *cluster*; para comprobarlo, se calcula *la densidad media de las entradas del nodo* y *la densidad media del buffer de ruido* y se comparan con el objetivo de **tomar la decisión de la aparición de un nuevo concepto**. De esta manera, en este trabajo, a diferencia del algoritmo *ClusTree* que crea nuevos *clusters* en los nodos hojas y el árbol siempre está balanceado, se pueden crear nuevos *clusters* también en nodos intermedios. Para evitar que se construya un árbol no balanceado a partir de la creación de nuevos *clusters* utilizando instancias ruidosas, los nodos que representan estos *clusters* se tratan de forma diferente al resto: se realiza un crecimiento de los subárboles que cuelgan de dichos nodos (denominados *liar subtrees*) utilizando las nuevas instancias hasta que los nodos hojas **están a la misma altura que el resto de hojas**. Cuando esto ocurre, estos subárboles se consideran parte de la estructura jerárquica (el subárbol deja de tener la etiqueta *liar*).

Otro trabajo que realiza una extensión de la propuesta desarollada en Kranen et al. (2009) es el planteado en Hesabi et al. (2015), donde proponen, a diferencia de los trabajos vistos hasta ahora, un algoritmo que lleva a a cabo un agrupamiento de **múltiples flujos de datos**. Concretamente, plantean construir una estructura de árbol jerárquica que permita **el acceso concurrente**, de tal forma que posibilite la extracción de correlaciones entre los flujos de datos con el objetivo de obtener *micro-clusters* de **mayor calidad** (agrupamiento concurrente), es decir, *micro-clusters* de mayor granularidad, en lugar de realizar el agrupamiento de forma individual por

cada uno de los flujos de datos. Este acceso concurrente a la estructura se realiza mediante la utilización de un conjunto de **procesadores**, donde cada uno de ellos se encarga de construir *micro-clusters* de forma paralela, pero siendo gestionados por un *mecanismo de control de la concurrencia*. Cada uno de estos procesadores explora diferentes subárboles al mismo tiempo, de tal forma que, a diferencia del algoritmo *ClusTree*, cuando llegan nuevas instancias **no se interrumpe el proceso de inserción de las instancias actuales**, por lo que eliminan los *buffers* de los nodos presentes en el algoritmo *ClusTree*; solo se lleva a cabo la interrupción en el caso en el que más de un procesador tenga que modificar un mismo *nodo hoja*. En esta última situación, **se bloquea el nodo hoja para un solo procesador** con la finalidad de mantener la consistencia de los datos; también se bloquean los distintos nodos de la estructura jerárquica cuando son visitados por cada uno de los procesadores. En el proceso de inserción puede ocurrir que un procesador inserte una instancia en un nodo hoja y haga que éste se divida, y por otro lado otro procesador vaya a insertar una instancia en dicho nodo hoja y no reconozca la división realizada; en este caso, a cada nodo se le asigna un identificador denominado ***Logical Sequence Number*** (LSN), que permite reconocer la división realizada y saber cómo atravesar la estructura jerárquica.

Algoritmo	Tipo de agrupamiento	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
BIRCH	Jerárquico aglomerativo	No	Si		Si (creo que categóricas no)		Si	No*
STREAM	Particional	No	No		Si	No	Si*	No*
CluStream	Particional	Creación, eliminación y unión de *micro-clusters*, así como la utilización de la estructura *pyramidal time frame*	Si		Si (creo que categóricas no)	No		No*
StreamKM++	Particional (aunque utiliza un agrupamiento aglomerativo para aplicar el *k-means++*)	No			Si (creo que categóricas no)	Si	Si*	No*
BICO	Particional (aunque utiliza un agrupamiento aglomerativo para aplicar el *k-means++*)	No			Si (creo que categóricas no)	Si	Si	No*
StreamXM	Particional	Número de clusters variable			Si (creo que categóricas no)	Si	Si*	No*
Liberty et al. (2016)	Particional	No		No*	Si (creo que categóricas no)	No*	No*	No*
ClusTree	Jerárquico aglomerativo	Función de desvanecimiento, unión y división de *micro-clusters* y *buffers* de ruido, así como otras propuestas de la literatura	Si (utilizando propuestas presentes en la literatura)	No	Si (creo que categóricas no)	Si	No	No*

Tabla 3.9: Algoritmos de aprendizaje no supervisado basados en agrupamiento (1)

Algoritmo	Tipo de agrupamiento	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
LiarTree	Jerárquico aglomerativo	Función de desvanecimiento y unión y división de *micro-clusters*, además de la utilización de otras propuestas de la literatura	Si (utilizando propuestas presentes en la literatura)	No	Si (creo que categóricas no)	Si	Si	No*
Hesabi et al. (2015)	Jerárquico aglomerativo	Función de desvanecimiento, unión y división de *micro-clusters*, así como otras propuestas de la literatura	Si (utilizando propuestas presentes en la literatura)	No	Si (creo que categóricas no)	Si	No	No*

Tabla 3.10: Algoritmos de aprendizaje no supervisado basados en agrupamiento (2)

3.4. Redes bayesianas para el descubrimiento de conocimiento

Una propuesta desarrollada para adaptar las redes bayesianas para el descubrimiento de conocimiento a la naturaleza de los flujos de datos es la planteada en Chen et al. (2001). En este trabajo abordan el problema de construir una red bayesiana a partir de datos distribuidos alojados en **bases de datos heterogéneas**, es decir, con diferentes esquemas. Para ello, primero llevan a cabo *la construcción de redes bayesianas locales* a partir de los datos almacenados en cada uno de los sistemas, de tal forma que estas redes bayesianas locales solo se construyen con **las instancias de una determinada base de datos** y con las **variables locales que las describen**. Tras llevar a cabo este paso, en cada uno de los sistemas se identifican las instancias que son relevantes a la hora de llevar a cabo **un acoplamiento entre las variables de las distintas redes bayesianas locales construidas**; es decir, las instancias que permitan encontrar relaciones entre las variables de las redes bayesianas locales. Para ello, se basan en establecer que aquellas instancias que *no encajen adecuadamente en los modelos locales* (baja probabilidad en el mo-

delo local correspondiente y que no supere un determinado umbral) son adecuadas para encontrar las relaciones mencionadas anteriormente. Estas instancias se envían a un sistema central para enlazar variables de distintas redes bayesianas locales y, uniendo las redes bayesianas locales y los enlaces encontrados, se obtiene una **red bayesiana colectiva**.

A partir de lo comentado anteriormente, realizan una adaptación de este trabajo a *flujos de datos múltiples*; para ello, **solo realizan una actualización de los parámetros de la red bayesiana**, asumiendo que se mantiene la estructura de la misma. Para llevar a cabo esta actualización con nuevas instancias, modifican los parámetros de las redes bayesianas locales utilizando un *modelo multinomial sin restricciones*, que se basa en actualizarlos empleando propiedades aditivas de una serie de contadores. Tras esto, se transfieren las instancias relevantes para el acoplamiento de variables de distintas redes bayesianas locales al sistema central y se actualizan los parámetros de los enlaces que relacionan variables de dichas redes *de la misma forma que en local*. Por últmo, se vuelven a combinar las estructuras para obtener una red colectiva actualizada.

En la propuesta anterior solo se lleva a cabo la actualización de los parámetros de la red bayesiana colectiva, *sin modificar la estructura de la misma*. En este sentido, un trabajo que aborda la actualización de la estructura de la red bayesiana, pero no de los parámetros, es el planteado en Yasin y Leray (2013), donde proponen el algoritmo denominado ***Incremental Max-Min Hill Climbing*** (*iMMHC*), que se basa en el algoritmo *Max-Min hill climbing* (*MMHC*, Tsamardinos et al. (2006)). El algoritmo *MMHC* se basa en aplicar la heurística *Max-Min parent children* (*MMPC*) con el objetivo de **aprender la estructura local de cada una de las variables**, es decir, los posibles padres e hijos de cada una de las variables (*CPC*); tras esto, realiza una búsqueda voraz del modelo global más óptimo posible mediante el algoritmo *hill climbing*, de tal manera que se van añadiendo (teniendo en cuenta el *CPC* de cada variable, es decir, el algoritmo tiene *restricciones*) y quitando arcos, así como cambiando su sentido; estas modificaciones se realizan teniendo en cuenta que en cada momento se obtiene un *DAG*. En esta propuesta (*iMMHC*), para llevar a cabo la identificación de la estructura local de cada una de las variables a medida que llegan nuevas instancias, utilizan una variante del algoritmo *MMPC* denominada ***iMMPC***, una versión incremental del mismo que reduce el espacio de búsqueda utilizando conocimiento previo de la estructura y considerando solo **dependencias fuertes entre variables** para usarlas en el proceso incremental. Para llevar a cabo la búsqueda voraz del modelo global, parten de la estructura construida y no solo *añaden arcos* y *se cambia su sentido*, sino también *eliminan aquellos que quedan obsoletos*.

El algoritmo *iMMHC* visto anteriormente asume que el concepto que describe a los flujos de datos es *estacionario* y utliza un modelo de ventana *landmark*; de esta manera, no se adapta adecuadamente a las características cambiantes de los flujos de datos. Por ello, en la tesis Yasin (2013) proponen dos extensiones del algoritmo *iMMHC*: uno que utiliza el modelo de ventana **deslizante** y otro que emplea el modelo de ventana **damped**. En la aproximación que utiliza el modelo

de ventana *deslizante* almacenan información del pasado de forma compacta como *estadísticas suficientes*; para ello, utilizan unos **contadores de frecuencia** que, junto con la estructura de la red bayesiana actual y unos contadores de instancias de la ventana actual, *se construye la nueva estructura de la red*. Este método solventa el inconveniente de guardar toda la información del pasado, pero no aborda la aparición de nuevos conceptos en los datos (el concepto de ventana *deslizante* se utilza para resumir información), por lo que en el método que utiliza el modelo de ventana *damped* introducen un factor de desvanecimiento α, que se le asigna a cada uno de los contadores de frecuencia de **todas las ventanas del flujo de datos**, a partir de los cuales se obtienen los contadores de frecuencia de la ventana actual. En este último método se tienen en cuenta los contadores de frecuencia de todas las ventanas, mientras que el método con la ventana *deslizante* mantiene solo una parte de ellas que resume la información del pasado.

Por otro lado, otra propuesta que se basa en aprender la estructura de la red bayesiana utilizando el algoritmo *MMHC* es la planteada en Trabelsi et al. (2013), aunque en este caso abordan el aprendizaje de **la estructura de una red bayesiana dinámica**, por lo que denominan a la propuesta **Dynamic MMHC** (*DMMHC*); concretamente, aprenden una estructura específica denominada *2-Time slice BN* (*2T-BNs*), una red bayesiana dinamica que cumple la *propiedad de Markov de primer orden*. Esta propuesta adapta el algoritmo *MMHC* teniendo en cuenta la dimensión temporal que se añade a la red bayesiana, de tal forma que consta de dos pasos principales. En el primero de ellos aplica una versión modificada del algoritmo *MMPC* denominada **DMMPC**, en el que se identifica la vecindad de una variable en el instante t teniendo en cuenta que, debido a la propiedad de Markov de primer orden, se puede encontrar a partir del conjunto de variables formado por aquellas presentes en el instante $t-1$ (**posibles padres**), en el instante t (**posibles padres e hijos**) y en el instante $t+1$ (**posibles hijos**). Al igual que en el algoritmo *MMPC*, en el algoritmo *DMMPC* llevan a cabo, tras el paso anterior, una *corrección simétrica*, de manera que una variable no puede ser vecina de otro si una de ellas no contiene a la otra en su vecindad. Después de realizar la búsqueda de las estructuras locales de las variables, se procede a realizar una búsqueda voraz de la misma manera que en el algoritmo *MMHC*, pero en este caso la adición de arcos se puede limitar debido a las restricciones que establece la estructura (*restricciones temporales*).

En la propuesta anterior construyen una red bayesiana dinámica *2T-BN*, de tal forma que ese modelo se mantiene durante el transcurso del tiempo. De esta manera, asumen que los datos son generados por un *proceso estacionario*, es decir, que tanto la estructura como los parámetros no evolucionan con el tiempo (asumen **homogeneidad** en el modelo). Incorporar esta idea en el modelo no es adecuado para modelar flujos de datos, cuya distribución de probabilidad subyacente puede cambiar. Por lo tanto, en Robinson y J. Hartemink (2008) proponen un algoritmo denominado ***non-stationary dynamic Bayesian networks*** (*nsDBN*), en el que permiten que las dependencias condicionales presentes en la estructura cambien durante el transcurso del tiempo. De esta forma, hay que llevar a cabo **el aprendizaje de diferentes estructuras de redes bayesianas en diferentes instantes de**

tiempo; los instantes de tiempo en los que la estructura del *DBN* cambia se denominan *tiempos de transición* y el periodo temporal entre estos tiempos de transición se denominan *épocas* (ver Figura 3.14). Para llevar a cabo la búsqueda de las diferentes estructuras utilizan una variante de la métrica *BDe* denominada **nsBDe**, que tiene en cuenta que un nodo puede tener varios conjuntos de padres activos en diferentes momentos del tiempo añadiento una serie de *intervalos* (cada uno de ellos incluye varias épocas). Al calculo de esta métrica se le añade información *a priori* de que la estructura de las redes construidas *evoluciona de forma suave y lentamente*; esta información se incluye como exponenciales teniendo en cuenta el número de épocas y del número de cambios entre redes consecutivas.

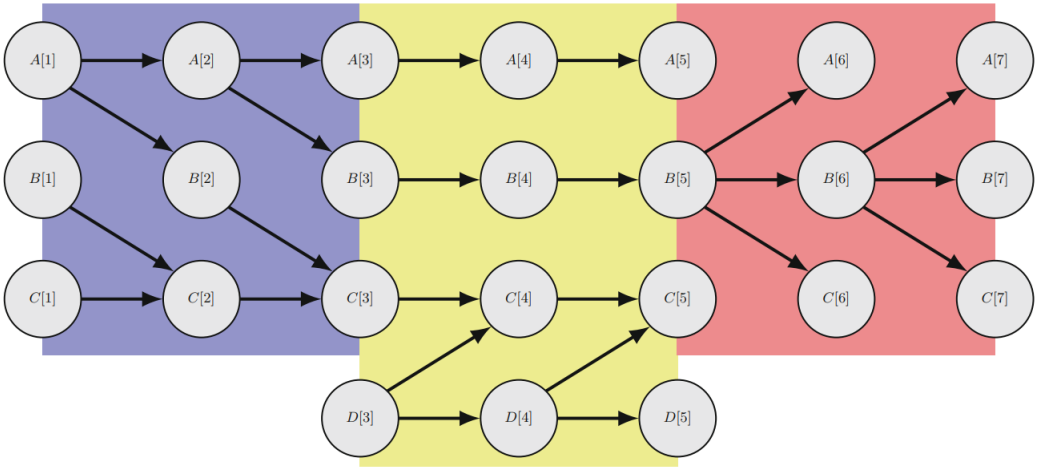

Figura 3.14: Estructura de un *nsDBN*. Los colores representan las *épocas* y los cambios de color los diferentes *tiempos de transición*. Fuente: Hourbracq et al. (2016)

.

En el proceso de búsqueda de las diferentes estructuras, tienen en cuenta diferentes posibles configuraciones en función de si el **número de transiciones** es conocido o no y si los **tiempos de transición** son conocidos o no. Cuando ambos son conocidos, se procede a **maximizar la métrica *nsBDe*** realizando una serie de adiciones y eliminaciones de arcos. En el caso de que se conozca el número de transiciones pero no los tiempos de transición, **se añade un movimiento local que permita modificar un tiempo de transición** con restricciones con respecto a tiempos de transición adyacentes. Si no se conoce ni el número de transiciones ni los tiempos de transición, **se introducen en el proceso de búsqueda movimientos de *split* y *merge*** sobre los conjuntos de arcos de diferentes redes (unión o división de las redes que forman la estructura del *nsDBN*), de tal forma que disminuya o aumente el número de transiciones durante dicho proceso. Todo el proceso de búsqueda se realiza mediante un **método de cadenas de Markov Monte Carlo** (*MCMC*).

El problema de la propuesta anterior es que *no escala bien en el número de nodos* y *es propenso a sobreajustarse*. Por ello, en Song et al. (2009) proponen un algoritmo denominado ***time-varying dynamic Bayesian networks*** (*TV-DBN*).

Este algoritmo descompone el problema del aprendizaje de las estructuras de las redes bayesianas dinámicas que constituyen la red bayesiana global en *un conjunto de subproblemas más simples*. Para ello, utiliza un **modelo autorregresivo**, de tal forma que el estado de las variables en un instante de tiempo t depende del estado de las variables en el instante $t-1$ mediante una *matriz de coeficientes*. En esta matriz los valores distintos de cero corresponden a **los arcos que modelan la dependencia de las variables del instante de tiempo t con respecto a las del instante de tiempo $t-1$**. Cada una de las matrices de coeficientes asociadas a diferentes instantes de tiempo y a diferentes nodos se optimiza de forma separada planteando *problemas de regresión ponderada individuales*. Al igual que en Robinson y J. Hartemink (2008), en este trabajo asumen que los cambios en las estructuras de las redes bayesianas dinámicas de dos instantes de tiempo consecutivos son suaves y, por ello, realizan **una ponderación de las instancias de manera gradual** en cada uno de los instantes de tiempo, de tal forma que los pesos disminuyen a medida que las instancias son menos recientes. Para evitar el sobreajuste del modelo a los datos, utilizan *la regularización L1*.

En los trabajos realizados en Robinson y J. Hartemink (2008) y Song et al. (2009) se asumen una serie de suposiciones acerca de la evolución de las redes bayesianas dinámicas en diferentes instantes de tiempo. Una propuesta que no tiene en cuenta ninguna suposición es la planteada en Gonzales et al. (2015). Concretamente, para desarrollar el algoritmo de esta propuesta, utilizan el trabajo realizado en Robinson y J. Hartemink (2008); se basan en la métrica *nsDBN* que proponen. Para desarrollar esta métrica, en Robinson y J. Hartemink (2008) asumen que los parámetros de las redes bayesianas dinámicas son mutuamente independientes de los de otras épocas. De esta manera, en este trabajo proponen tener en cuenta en la métrica las **dependencias entre los parámetros de diferentes intervalos de tiempo**; concretamente, tienen en cuenta la fuerza de los arcos utilizando la *información mutua de un nodo y su padre* y utilizan los parámetros anteriores como información *a priori* para los nodos que no cambian de un modelo a otro. Además, para hallar los tiempos de transición reduciendo el espacio de búsqueda, utilizan el **estadístico** χ^2 sobre cada una de las variables utilizando un nuevo fragmento de datos que llegue en un instante de tiempo determinado, de manera que si alguno de los estadísticos indica que dicho fragmento de datos se ha generado a partir de una red bayesiana dinamica *diferente a la del instante anterior*, quiere decir que hemos llegado a **un nuevo tiempo de transición**. Antes de la utilización del estadístico, si el nuevo fragmento de datos no contiene las mismas variables que la red bayesiana del intervalo de tiempo anterior o si aparecen valores de las variables que no se han contemplado, se considera también que hay un nuevo tiempo de transición. De esta forma el algoritmo, en cada intervalo de tiempo, si se demuestra lo anterior, **lleva a cabo el aprendizaje de la nueva red bayesiana en ese intervalo** utilizando la métrica *nsDBN* teniendo en cuenta la dependencia entre los parámetros previos y los actuales; en caso contrario, **se actualizan los parámetros de la red bayesiana del intervalo de tiempo anterior con los nuevos datos**.

Otra propuesta que tiene en cuenta en la construcción de la estructura de las

redes bayesianas dinámicas la no estacionariedad de los datos durante el transcurso del tiempo es la planteada en Hourbracq et al. (2016). En este trabajo desarrollan un algoritmo que utiliza una **ventana deslizante para llevar a cabo la elección de una red bayesiana en cada uno de los instantes de tiempo**. La red bayesiana que se escoge en cada instante de tiempo puede ser o **una presente en algún instante de tiempo anterior** o **una nueva red bayesiana que se construye**; en este sentido, una diferencia de este trabajo con respecto al algoritmo propuesto en Gonzales et al. (2015) consiste en que se puede reutilizar alguna red bayesiana dinámica del pasado, mientras que en Gonzales et al. (2015) solo se contempla la reutilización de la red bayesiana del instante de tiempo anterior. Además, no asume que los cambios que se producen entre las redes bayesianas de diferentes instantes de tiempo sean suaves, a diferencia del trabajo realizado en Robinson y J. Hartemink (2008). Para contemplar la reutilización de alguna *DBN* del pasado, se calculan *una serie de verosimilitudes asociadas a cada una de las redes con respecto a la ventana de datos actual* y, utilizando un umbral, se almacenan aquellas redes cuya **verosimilitud lo superan**, eligiendo como la *DBN* del instante de tiempo actual aquella que tenga el mayor valor de verosimilitud y se actualizan tanto los parámetros como la estructura de la misma con los nuevos datos. En caso de que ninguna supere el umbral, **se construye una nueva *DBN***.

En el caso de que la *DBN* del instante de tiempo actual sea una nueva red bayesiana u otra del pasado que no sea la del instante anterior, en este trabajo hallan **el instante de tiempo dentro de un rango temporal con respecto a la ventana actual donde se produce el cambio del concepto de los datos**, de tal forma que una parte de la ventana se utiliza para *actualizar la DBN del instante de tiempo anterior* y la otra parte para *actualizar o construir la red bayesiana del instante de tiempo actual*. Además, al igual que en Gonzales et al. (2015), tienen en cuenta la posible aparición de nuevos atributos o desaparición de atributos existentes. Para ello, a la hora de elegir una *DBN* para el instante de tiempo actual, establecen, para cada una de las redes candidatas antes de calcular sus verosimilitudes, que si la red tiene una variable que no está presente en la nueva ventana de datos, **se actualizan los parámetros de la misma sin tener en cuenta esa variable**; en el caso de que exista alguna variable que esté presente en la ventana de datos pero no en la red, **se descarta dicha variable**. En el trabajo desarrollado en Gonzales et al. (2015), si los valores de las variables o los atributos difieren del modelo del instante anterior, directamente se construye una nueva red, sin realizar ninguna actualización, a diferencia de esta propuesta. Además, tienen en cuenta nuevos valores de las variables utilizando los parámetros *a priori* de la distribución de *Dirichlet* y creando nuevos parámetros de la red en base a esos parámetros *a priori*, asi como llevando a cabo la actualización de los parámetros de la red existentes. Cabe mencionar también que tanto en esta propuesta como en anteriores se manejan *variables discretas*, pero hay propuestas que manejan variable continuas en redes bayesianas dinámicas no estacionarias, como en Grzegorczyk y Husmeier (2009).

Otro de los tipos de redes bayesianas para modelar procesos dinámicos visto con anterioridad son las *redes bayesianas en tiempo continuo*. Un trabajo relevante que

aborda el aprendizaje de este tipo de redes bayesianas es el planteado en Nodelman et al. (2003), cuyos autores son los que plantearon las redes bayesianas en tiempo continuo en Nodelman et al. (2002). Para llevar a cabo este aprendizaje, primero establecen que la red bayesiana que representa la distribución inicial se construya con **un algoritmo estándar de aprendizaje de una red bayesiana**. Por otra parte, para aprender el modelo de transición continua que caracteriza a las redes bayesianas en tiempo continuo, establecen que la distribución que subyace a las transiciones de las distintas variables consta de dos partes: una *distribución exponencial* sobre los instantes de tiempo en los que van a ocurrir las transiciones y una *distribución multinomial* sobre el siguiente estado de las variables.

Para hallar los parámetros, suponiendo que la estructura es fija, una de las posibilidades que tienen en cuenta es utilizar el **método de máxima verosimilitud**; para ello, utilizan los estadísticos suficientes $T[x|u]$, que es la cantidad de tiempo en el que una variable X se mantiene en un estado x condicionado a una instanciación de los padres u, y $M[x, x'|u]$, que es el número de transiciones que realiza la variable X desde el estado x al estado x' condicionado a u. Otra manera que contemplan para aprender los parámetros es utilizando la **aproximación bayesiana**, de tal forma que establecen una distribución *a priori* sobre los mismos que sea **conjugada**; de esta forma, para la distribución exponencial utilizan como distribución conjugada *a priori* la *distribución Gamma*, mientras que para la multinomial emplean la *distribución Dirichlet*. De esta manera, se mantienen las distribuciones de los distintos parámetros **en forma cerrada**, es decir, la actualización de los mismos se lleva a cabo con una fórmula cerrada en la que se utilizan los hiperparámetros oportunos y los estadísticos suficientes anteriores. Para llevar a cabo el aprendizaje de la estructura, utilizan una aproximación *score-based*. Para definir la métrica, asumen tanto **independencia local** como **global** de los parámetros, además de las propiedades de **modularidad de los parámetros** y **de la estructura**. En el proceso de búsqueda de la estructura aplican la búsqueda voraz *hill climbing* con operaciones de adición y eliminación de arcos.

La propuesta de aprendizaje de una red bayesiana en tiempo continuo previa no tiene en cuenta que la estructura de la misma puede sufrir cambios debido a la evolución del concepto de los datos. Una propuesta que si lo aborda es la planteada en Stella y Villa (2016), donde proponen un algoritmo denominado ***structurally non-stationary continuous time Bayesian network*** (*nsCTBN*), que permite modificar los padres de las distintas variables en determinados *tiempos de transición*. Para llevar a cabo el aprendizaje, se basan en una variante de la métrica bayesiana propuesta en Nodelman et al. (2003), donde tienen en cuenta que las variables pueden tener *diferentes padres activos* en cada una de las épocas o en uniones de ellas (intervalos); esto se refleja en las matrices de intensidades condicionales que proponen y en la adaptación de las fórmulas propuestas en Nodelman et al. (2003).

A la hora de realizar el proceso de aprendizaje, tienen en cuenta tres posibles configuraciones (al igual que en Robinson y J. Hartemink (2008)): **los tiempos de transición son conocidos**, **el número de transiciones es conocido** y **el número de transiciones es desconocido**. En el caso en el que *se conozcan los tiempos*

de transición, para encontrar la estructura óptima maximizan los componentes de la métrica bayesiana asociados a cada uno de los nodos de forma individual (tras realizar la descomposición de la métrica bayesiana para cada una de las variables) utilizando un **algoritmo de optimización exacto basado en programación dinámica**. Si *se conoce el número de transiciones* pero *no los tiempos de transición*, utilizan el algoritmo *simulated annealing* con el objetivo de hallar una solución aproximada a los tiempos de transición, de tal forma que, asumiendo que estos tiempos son cercanos a los valores verdaderos de los tiempos de transición, **se aplica el algoritmo de la configuración de tiempos de transición conocidos**. Si *no se conoce el número de tiempos de transición*, se realiza lo mismo que en el caso de que solo se conozca el número de transiciones pero se añaden operaciones de *merge* y *split* para realizar el proceso de búsqueda del número de tiempos de transición (al igual que en Robinson y J. Hartemink (2008)).

Por otra parte, la complejidad del modelo establecido por las redes bayesianas en tiempo continuo es exponencial con respecto a los nodos padres de las variables, dando lugar a problemas de escalabilidad del mismo. Por ello, en Perreault y Sheppard (2019), aunque no tengan en cuenta la no estacionariedad de la estructura de las redes bayesianas en tiempo continuo como en Stella y Villa (2016), llevan a cabo una **representación compacta de la misma** para disminuir la complejidad del modelo utilizando dos aproximaciones: **MCIM** y **TCIM**. Para ello, en la aproximación *MCIM* primero realizan un **agrupamiento jerárquico aglomerativo de las matrices de intensidades** que pueden ser tratadas como procesos de Markov equivalentes dentro de cada una de las *matrices de intensidades condicionales* (*CIM*); tras esto, se realiza un mapeo entre las diferentes *instanciaciones de los padres* y *las matrices de intensidades obtenidas del agrupamiento* (varias instanciaciones de los padres pueden mapearse a una misma matriz de intensidad obtenida a partir de un *cluster*), que se definen como la media de las matrices de intensidades de cada uno de los *clusters* construidos. Esta representación compacta utiliza todas las combinaciones posibles de los padres para asociarlas a a las matrices de intensidades de los *clusters* construidos, por lo que, para disminuir aun más la complejidad del modelo, llevan a cabo el mapeo mencionado anteriormente utilizando **instanciaciones de subconjuntos de los padres de las variables** teniendo en cuenta la propiedad de *equivalencia contextual*, es decir, instanciaciones de subconjuntos de los padres de las variables que tienen la misma instanciación en el resto de los padres; esto permite realizar una descomposición de la función de mapeo anterior en diferentes funciones cuya complejidad espacial es menor. Con respecto a la aproximación *TCIM*, para representar las matrices de intensidades obtenidas por los *clusters* de forma compacta, en lugar de llevar a cabo la descomposición de la aproximación *MCIM*, utiliza una **estructura de árbol**, cuyo objetivo es capturar un conjunto de independencias contextuales entre los diferentes *clusters*.

Con respecto a las redes bayesianas de nodos temporales, en Hernandez-Leal et al. (2013) proponen un algoritmo de aprendizaje de este tipo de redes que denominan **LIPS** (*Learning Intervals Parameters and Structure*). Aparte de la estructura y de los parámetros de estas redes, en este algoritmo de aprendizaje ponen énfasis en el

hallazgo de **los intervalos asociados a cada uno de los nodos temporales**; en otros algoritmos de aprendizaje de redes bayesianas no tienen en cuenta el aprendizaje de estos intervalos, por lo que no son aplicables a la construcción de redes bayesianas de nodos temporales. El primer paso de este algoritmo es obtener *una serie de intervalos iniciales para los nodos temporales* a partir del conjunto de datos; para ello, utilizan una **discretización uniforme** en cada uno de los nodos temporales o el algoritmo ***k-means*** (mejor algoritmo de aproximación de intervalos que la discretización uniforme), cuyos centroides se convierten en intervalos temporales.

Tras la obtención de los intervalos iniciales, el siguiente paso del algoritmo es el **aprendizaje estructural** utilizando dichos intervalos, para lo que emplean el algoritmo *K2* (Cooper y Herskovits (1992)) debido a que requiere un parámetro de ordenamiento de las variables; es decir, se utiliza la información temporal disponible para dar una ordenación de las variables, por lo que se coloca al inicio de la ordenación las variables cuyos nodos son *instantáneos* y posteriormente las variables cuyos nodos son *temporales*. Una vez realizado el aprendizaje de la estructura, se lleva a cabo **el aprendizaje de los intervalos de los nodos temporales** mediante la aplicación de un *algoritmo de aprendizaje de intervalos* en cada uno de dichos nodos en dos fases. La primera consiste en utilizar un *modelo de mixtura de Gaussianas* para encontrar una primera aproximación de los intervalos utilizando **los parámetros de las diferentes Gaussianas** (se encuentran con el algoritmo *EM* y el número de Gaussianas es el número de intervalos); para elegir el mejor conjunto de intervalos, se utiliza la métrica *Brier score*, que evalúa el desempeño de la red. Esta primera fase no utiliza información de la red, por lo que en la segunda fase se lleva a cabo un refinamiento de los intervalos utilizando **la información de los padres de cada uno de los nodos temporales** (instanciaciones de los padres).

Algoritmo	Detección de *concept drift*	Manejo de *outliers*	Manejo de datos faltantes	Manejo de variables continuas	Manejo de datos de alta dimensión	Manejo del ruido	Manejo de la aparición de nuevos atributos
Chen et al. (2001)	No (aunque elimina arcos obsoletos)	No		No			No
iMMHC	No			No	Si		No
Yasin (2013) (sliding window)	No			No	Si		No
Yasin (2013) (damping window)	Damping window			No	Si		No
DMMHC	No			No	Si		No
nsDBN	Métrica nsBDe		No*	No	No	Si*	No
TV-DBN	Pesado de las instancias y matrices de coeficientes	No*	No*	No	Si		No
Gonzales et al. (2015)	Métrica nsBDe con adición de dependencia entre parámetros y utilización del estadístico χ^2 para los tiempos de transición		No*	No	No*		Si
Hourbracq et al. (2016)	Ventana deslizante adaptativa y verosimilitudes de las *DBN*	Si*	No*	No	Si*	Si*	Si
Nodelman et al. (2003)	No		No	No	No*		No
nsCTBN	Diferentes padres de las variables en cada uno de los tiempos de transición		No	No	No*	Si*	No
MCIM y TCIM	No		No	No	Si	Si*	No
LIPS	No		No	No (solo las variables temporales)	No*	No*	No

Tabla 3.11: Redes bayesianas para el descubrimiento de conocimiento

3.5. Conjuntos de datos frecuentes en flujos de datos

En la literatura de aprendizaje automático para flujos de datos existe una variedad de conjuntos de datos que se utilizan con asiduidad dadas sus propiedades. A continuación se exponen los más frecuentes:

- **SEA Concepts Generator**. Es un conjunto de datos artificial definido por *tres atributos*, donde los dos primeros son relevantes, y *dos clases*, además de haber presente un 10 % de ruido. Además, contiene 60,000 ejemplos, presenta *concept drift* abrupto y todos los atributos tienen valores entre 0 y 10. Los ejemplos se dividen en *cuatro bloques*, cada uno de ellos con diferentes conceptos, de tal forma que, en cada bloque, una instancia pertenece a la clase 1 si $f1 + f2 \leq \theta$, donde $f1$ y $f2$ representan los dos primeros atributos y θ es un umbral establecido entre las dos clases. Los umbrales utilizados en cada uno de los bloques son 8, 9, 7 y 9.5.

- **Rotating Hyperplane**. La orientación y posición de un hiperplano en el espacio d-dimensional se cambia para producir un *concept drift*. Este conjunto de datos tiene características iguales a las de *SEA*, pero contiene un *concept drift* gradual. Concretamente, en este conjunto de datos las etiquetas de clase dependen de la *ubicación de los puntos bidimensionales* en comparación con un hiperplano que rota durante el curso del flujo de datos. La rotación comienza con un cierto ángulo cada 1000 instancias, comenzando después de las primeras muestras de 10,000 instancias, siendo los ángulos 20, 30, y 40.

- **Random RBF Generator**. Es un generador de datos provenientes de una **función de base radial**. Comienza generando un número fijo de **centroides aleatorios**. Cada centro tiene una *posición aleatoria*, una *única desviación estándar*, *etiqueta de clase* y *peso*. Se generan nuevos ejemplos seleccionando un centro al azar, teniendo en cuenta los pesos de tal manera que los centros con mayor peso tengan *más probabilidades de ser elegidos*. Se elige una dirección aleatoria para desplazar los valores de los atributos desde el punto central. La longitud del desplazamiento se extrae aleatoriamente de una *distribución gaussiana* con una desviación estándar determinada por el centroide elegido. El centroide elegido también determina la *etiqueta de clase del ejemplo*. Esto crea una **hiperesfera de ejemplos normalmente distribuida** alrededor de cada punto central con densidades variables.

- **LED Generator**. Este generador produce **dígitos** que se muestran en una pantalla LED de siete segmentos descritos por 24 atributos binarios, donde 17 de ellos son *irrelevantes* (los relevantes son los 7 atributos correspondientes a cada uno de los segmentos). Además, por cada uno de los dígitos, añade ruido, de manera que cada uno de los atributos tiene una probabilidad del 10 % de que su valor sea *invertido*.

- **Forest Covertype**. Contiene el tipo de cubierta forestal para celdas de 30 x 30 metros obtenido a partir de datos del *Servicio Forestal de los Estados Unidos*. Contiene 581012 instancias y 54 atributos cartográficos continuos y categóricos. El objetivo es predecir el tipo de cubierta forestal de una determinada área.

- **KDDCUP 99**. Este conjunto de datos se utilizó en la competición *KDD Cup 1999*. Contiene cuatro millones de *registros de conexión* TCP de dos semanas del tráfico de red LAN gestionado por *MIT Lincoln Labs*. Cada registro puede corresponder a una conexión *normal* o a una *intrusión* (o ataque). Los ataques se dividen en *22 tipos* y, como resultado, los datos contienen un total de *23 clases*, incluida la clase para conexiones normales. La mayoría de las conexiones en este conjunto de datos son normales, pero, ocasionalmente, puede haber una ráfaga de ataques en ciertos momentos. Además, cada registro de conexión de este conjunto de datos contiene *42 atributos*, tanto continuos como categóricos. Los ataques se engloban en cuatro categorías: *DOS* (denegación de servicio), *R2L* (acceso no autorizado desde una máquina remota), *U2R* (acceso no autorizado a privilegios de superusuario local) y *sondeo* (vigilancia y otros sondeos). Los datos de *test* no son de la misma distribución de probabilidad que los datos de entrenamiento y también hay nuevos tipos de ataques que no están en los datos de entrenamiento, concretamente 14.

- **Waveform**. Este conjunto de datos contiene 5,000 instancias y 40 atributos, de los cuales los 19 últimos son *atributos ruidosos*, cuyo ruido esta definido con media 0 y varianza 1. Además, contiene 3 clases, que corresponden a **tres tipos diferentes de ondas**, de manera que las instancias de cada una de las clases se generan a partir de una *combinación de ondas base*, y cada una de ellas se corrompe con ruido con media 0 y varianza 1. De esta manera, el objetivo es determinar el **tipo de onda**. El conjunto de datos consta de un 33 % de instancias pertenecientes a cada una de las clase.

Capítulo 4

Conclusiones y líneas futuras de trabajo

En este trabajo se ha llevado a cabo una revisión extensa del estado del arte relacionado con algoritmos de aprendizaje automático para flujos de datos. Para ello, hemos divido las diferentes propuestas en algoritmos de aprendizaje supervisado, aprendizaje no supervisado y redes bayesianas para el descubrimiento del conocimiento, haciendo hincapié en las redes bayesianas que modelan la evolución del estado de las distintas variables en el tiempo. Además, para cada uno de los algoritmos, se han llevado a cabo comparaciones entre diferentes trabajos utilizando como apoyo una tabla comparativa de las características de los mismos.

Durante el desarrollo de este trabajo, hemos advertido que en muchos artículos donde se realizan propuestas de aprendizaje automático para flujos de datos **no está disponible el código fuente de los experimentos realizados en los mismos**. Esto adquiere especial relevancia puesto que, para poder constatar que estos trabajos son rigurosos, es necesario que el código que han desarrollado para mostrar los resultados. Además, la disponibilidad del código desarrollado es fundamental para poder proponer mejoras de los algoritmos y realizar comparativas con menos dificultades, puesto que en muchos casos es necesario recurrir a los autores para la adquisición del código.

Una de las extensiones de este trabajo que se propone para el futuro es **la implementación de una nueva propuesta para la clasificación de flujos de datos basada en clasificadores bayesianos**, así como su comparación con otras propuestas en datos reales. Para llevar a cabo esto, existen diferentes paquetes de software dedicados al aprendizaje automático para flujos de datos. Uno de los más completos es el denominado *scikit-flow*, desarrollado en Python y basado en *frameworks* de código abierto, incluyendo *scikit-learn*, *MOA* (otro paquete de aprendizaje automático para flujos de datos) y MEKA (para aplicar aprendizaje automático en flujos de datos multi-etiqueta). Este paquete, además de permitir implementar algoritmos, posibilita la realización de experimentos a través de *generadores de flujos de datos*, *utilización de algoritmos disponibles*, *detectores de cambio en el concepto de los datos* y *métodos de evaluación*.

Por otra parte, se pretende extender este trabajo añadiendo propuestas relacionadas con **la predicción de series temporales**. Específicamente, se propone abordar distintos trabajos que tengan en cuenta el pronóstico de observaciones de series temporales cuyo proceso de generación puede cambiar. Además, con respecto a los algoritmos de agrupamiento, en este trabajo hemos abordado métodos particionales y jerárquicos, por lo que en el futuro se pretende añadir métodos *probabilísticos*, además de *density-based* y *grid-based*.

Apéndice A

Anexos

Bibliografía

M. R. Ackermann, M. Märtens, C. Raupach, K. Swierkot, C. Lammersen, y C. Sohler. StreamKM++: A clustering algorithm for data streams. *ACM Journal of Experimental Algorithmics*, 17:173–187, 2010.

C. C. Aggarwal. A survey of stream clustering algorithms. In *Data Clustering: Algorithms and Applications*, pages 457–482. CRC Press, 2013.

C. C. Aggarwal. A survey of stream classification algorithms. In *Data Classification: Algorithms and Applications*, pages 245–268. Springer, 2014.

C. C. Aggarwal, J. Han, J. Wang, y P. S. Yu. A framework for clustering evolving data streams. In *Proceedings of the 29th International Conference on Very Large Data Bases*, volume 29, pages 81–92. VLDB Endowment, 2003.

C. C. Aggarwal, J. Han, J. Wang, y P. S. Yu. A framework for on-demand classification of evolving data streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(5):577–589, 2006.

T. Al-Khateeb, M. M. Masud, L. Khan, C. Aggarwal, J. Han, y B. Thuraisingham. Stream classification with recurring and novel class detection using class-based ensemble. In *2012 IEEE 12th International Conference on Data Mining*, pages 31–40, 2012.

C. Anagnostopoulos, D. K. Tasoulis, N. Adams, y D. Hand. Temporally adaptive estimation of logistic classifiers on data streams. *Advances in Data Analysis and Classification*, 3:243–261, 2009.

R. Anderson y Y. S. Koh. StreamXM: An adaptive partitional clustering solution for evolving data streams. In *Big Data Analytics and Knowledge Discovery*, pages 270–282. Springer, 2015.

A. Besedin, P. Blanchart, M. Crucianu, y M. Ferecatu. Evolutive deep models for online learning on data streams with no storage. In *ECML/PKDD 2017 Workshop on Large-Scale Learning from Data Streams in Evolving Environments*, 2017.

C. Bielza y P. Larrañaga. Discrete Bayesian network classifiers: A survey. *ACM Computing Surveys*, 47:1–43, 2014.

A. Bifet y R. Gavaldà. Adaptive learning from evolving data streams. In *Advances in Intelligent Data Analysis VIII*, pages 249–260. Springer, 2009.

A. Bifet y R. Gavaldà. Learning from time-changing data with adaptive windowing. In *Proceedings of the 2007 SIAM International Conference on Data Mining*, volume 7, pages 443–448, 2007.

A. Bifet, G. Holmes, B. Pfahringer, y E. Frank. Fast perceptron decision tree learning from evolving data streams. In *Advances in Knowledge Discovery and Data Mining*, pages 299–310. Springer, 2010.

A. Bifet, B. Pfahringer, J. Read, y G. Holmes. Efficient data stream classification via probabilistic adaptive windows. In *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pages 801–806. ACM, 2013.

C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

H. Borchani, P. Larrañaga, y C. Bielza. Classifying evolving data streams with partially labeled data. *Intelligent Data Analysis*, 15(5):655–670, 2011.

M. Boullé. MODL: A Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65:131–165, 2006.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

L. Breiman. Random Forests. *Machine Learning*, 45(1):5–32, 2001.

L. Breiman, J. H. Friedman, R. A. Olshen, y C. J. Stone. *Classification and Regression Trees*. Wadsworth Publishing Company, 1984.

R. Chen, K. Sivakumar, y H. Kargupta. An approach to online bayesian learning from multiple data streams. In *Proceedings of Workshop on Mobile and Distributed Data Mining*, pages 31–45, 2001.

D. Cheng, S. Zhang, Z. Deng, Y. Zhu, y M. Zong. kNN algorithm with data-driven k value. In *Advanced Data Mining and Applications*, pages 499–512. Springer, 2014.

C. Chow y C. Liu. Approximating discrete probability distributions with dependence trees. *IEEE Transactions on Information Theory*, 14(3):462–467, 1968.

W. W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann Publishers Inc., 1995.

G. F. Cooper y E. Herskovits. A Bayesian method for the induction of probabilistic networks from data. *Machine Learning*, 9(4):309–347, 1992.

I. Czarnowski y P. Jędrzejowicz. Ensemble classifier for mining data streams. *Procedia Computer Science*, 35:397–406, 2014.

V. G. T. da Costa, A. C. P. de Leon Ferreira de Carvalho, y S. B. Junior. Strict very fast decision tree: A memory conservative algorithm for data stream mining. *Pattern Recognition Letters*, 116:22–28, 2018.

T. Dean y K. Kanazawa. A model for reasoning about persistence and causation. *Computational Intelligence*, 5:142–150, 1989.

M. Deckert y J. Stefanowski. RILL: Algorithm for learning rules from streaming data with concept drift. In *Foundations of Intelligent Systems*, pages 20–29. Springer, 2014.

A. P. Dempster, N. M. Laird, y D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39 (1):1–38, 1977.

T. G. Dietterich. Ensemble methods in machine learning. In *Multiple Classifier Systems*, pages 1–15. Springer, 2000.

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

U. M. Fayyad y K. B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pages 1022–1027. Morgan Kaufmann Publishers Inc., 1993.

F. Ferrer-Troyano, J. Aguilar-Ruiz, y J. Riquelme. Discovering decision rules from numerical data streams. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 649–653. ACM, 2004.

F. Ferrer-Troyano, J. Aguilar-Ruiz, y J. Riquelme. Incremental rule learning and border examples selection from numerical data streams. *Journal of Universal Computer Science*, 11:1426–1439, 2005.

F. Ferrer-Troyano, J. Aguilar-Ruiz, y J. Riquelme. Data streams classification by incremental rule learning with parameterized generalization. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 657–661. ACM, 2006.

H. Fichtenberger, M. Gillé, M. Schmidt, C. Schwiegelshohn, y C. Sohler. BICO: BIRCH meets coresets for k-means clustering. In *Algorithms – ESA 2013*, pages 481–492. Springer, 2013.

E. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics*, 21(3):768–769, 1965.

Y. Freund y R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *Computational Learning Theory*, pages 23–37. Springer, 1995.

J. Fürnkranz y G. Widmer. Incremental reduced error pruning. In *Proceedings of the Eleventh International Conference on Machine Learning*, pages 70–77. Morgan Kaufmann Publishers Inc., 1994.

S. F. Galán, G. Arroyo-Figueroa, F. J. Díez, y L. E. Sucar. Comparison of two types of event Bayesian networks: A case study. *Applied Artificial Intelligence*, 21:185–209, 2007.

J. Gama y P. Kosina. Learning decision rules from data streams. In *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, pages 1255–1260. AAAI Press, 2011.

J. Gama y P. Medas. Learning decision trees from dynamic data streams. *Journal of Universal Computer Science*, 11:1353–1366, 2005.

J. Gama y C. Pinto. Discretization from data streams: Applications to histograms and data mining. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, pages 662–667. ACM, 2006.

J. Gama, R. Rocha, y P. Medas. Accurate decision trees for mining high-speed data streams. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 523–528. ACM, 2003.

J. Gama, R. Sebastião, y P. Rodrigues. Issues in evaluation of stream learning algorithms. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 329–338. ACM, 2009.

H. M. Gomes, J. P. Barddal, F. Enembreck, y A. Bifet. A survey on ensemble learning for data stream classification. *ACM Computing Surveys*, 50:23:1–23:36, 2017a.

H. M. Gomes, A. Bifet, J. Read, J. P. Barddal, F. Enembreck, B. Pfharinger, G. Holmes, y T. Abdessalem. Adaptive random forests for evolving data stream classification. *Machine Learning*, 106(9):1469–1495, 2017b.

C. Gonzales, S. Dubuisson, y C. E. Manfredotti. A new algorithm for learning non-stationary dynamic Bayesian networks with application to event detection. In *Florida Artificial Intelligence Research Society Conference*, pages 564–569, 2015.

M. Greenwald y S. Khanna. Space-efficient online computation of quantile summaries. *SIGMOD Record*, 30(2):58–66, 2001.

A. Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.* O'Reilly Media, Inc., 2017.

M. Grzegorczyk y D. Husmeier. Non-stationary continuous dynamic bayesian networks. In *Advances in Neural Information Processing Systems 22 (NIPS 2009)*, volume 22, pages 682–690. Curran Associates, Inc., 2009.

P. Hernandez-Leal, J. A. Gonzalez, E. F. Morales, y L. E. Sucar. Learning temporal nodes Bayesian networks. *International Journal of Approximate Reasoning*, 54 (8):956 – 977, 2013.

Z. R. Hesabi, T. Sellis, y X. Zhang. Anytime concurrent clustering of multiple streams with an indexing tree. In *Proceedings of the 4th International Conference on Big Data, Streams and Heterogeneous Source Mining: Algorithms, Systems, Programming Models and Applications*, volume 41, pages 19–32. JMLR, 2015.

M. Hourbracq, P.-H. Wuillemin, C. Gonzales, y P. Baumard. Real time learning of non-stationary processes with dynamic Bayesian networks. In *16th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, pages 338–350. Springer, 2016.

G. Hulten, L. Spencer, y P. Domingos. Mining time-changing data streams. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 97–106. ACM, 2001.

E. B. Hunt, P. J. Stone, y J. Marin. *Experiments in induction.* Academic Press, 1966.

M. V. Jiménez. Ejemplos de análisis cluster. Departamento de Estadística e Investigación Operativa, Universidad de Granada, 2019. URL `https://www.ugr.es/~mvargas/3.DosEjesanalisisclusteryCCAA.pdf`. Accessed: 04-05-2019.

R. Jin y G. Agrawal. Efficient decision tree construction on streaming data. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 571–576. ACM, 2003.

P. K. Agarwal y S. Raghvendra. Streaming algorithms for extent problems in high dimensions. *Algorithmica*, 72(1):83–98, 2015.

I. Khamassi, M. Sayed Mouchaweh, M. Hammami, y K. Ghédira. Discussion and review on evolving data streams and concept drift adapting. *Evolving Systems*, 9: 1–23, 2016.

M. Khan, Q. Ding, y W. Perrizo. K-nearest neighbor classification on spatial data streams using P-trees. In *Proceedings of the 6th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, pages 517–518. Springer, 2002.

D. Kishore Babu, Y. Ramadevi, y K. V. Ramana. RGNBC: Rough gaussian naïve Bayes classifier for data stream classification with recurring concept drift. *Arabian Journal for Science and Engineering*, 42:705–714, 2016.

R. Klinkenberg. Using labeled and unlabeled data to learn drifting concepts. In *Workshop notes of the IJCAI-01 Workshop on Learning from Temporal and Spatial Data*, pages 16–24. AAAI Press, 2001.

R. Klinkenberg y T. Joachims. Detecting concept drift with support vector machines. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 487–494. Morgan Kaufmann Publishers Inc., 2000.

P. Kosina y J. Gama. Very fast decision rules for multi-class problems. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pages 795–800. ACM, 2012a.

P. Kosina y J. Gama. Handling time changing data with adaptive very fast decision rules. In *Machine Learning and Knowledge Discovery in Databases*, volume 7523, pages 827–842. Springer, 2012b.

P. Kranen, I. Assent, C. Baldauf, y T. Seidl. Self-adaptive anytime stream clustering. In *2009 Ninth IEEE International Conference on Data Mining*, pages 249–258, 2009.

P. Kranen, F. Reidl, F. Sanchez Villaamil, y T. Seidl. Hierarchical clustering for real-time stream data with noise. In *Proceedings of the 23rd International Conference on Scientific and Statistical Database Management*, volume 6809, pages 405–413. Springer, 2011.

B. Krawczyk y M. Woźniak. Incremental learning and forgetting in one-class classifiers for data streams. In *Proceedings of the 8th International Conference on Computer Recognition Systems*, pages 319–328. Springer, 2013.

B. Krawczyk y M. Wozniak. Weighted naïve Bayes classifier with forgetting for drifting data streams. In *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pages 2147–2152, 2015.

B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski, y M. Woźniak. Ensemble learning for data stream analysis: A survey. *Information Fusion*, 37:132–156, 2017.

D. Kumar, K. Padma Kumari, y S. Meher. Progressive granular neural networks with class based granulation. In *2016 IEEE Annual India Conference*, pages 1–6, 12 2016.

P. Larrañaga y C. Bielza. Basics of Bayesian networks. Departamento de Inteligencia Artificial, Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, 2019a. URL `https://drive.google.com/open?id=1Mjwu8taSGhZdGGjpm8uSQr6AK0DpwEQB`. Accessed: 06-07-2019.

P. Larrañaga y C. Bielza. Inference in Bayesian networks. Departamento de Inteligencia Artificial, Escuela Técnica Superior de Ingenieros Informáticos, Universidad Politécnica de Madrid, 2019b. URL `https://drive.google.com/open?id=1abQxhG5KkRXl8P108nd0sl4GGIXji4uC`. Accessed: 06-07-2019.

P. Larrañaga, I. Inza, y A. Moujahid. Tema 5. Clasificadores K-NN. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad del País Vasco, 2007. URL `http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t9knn.pdf`.

P. Larrañaga, D. Atienza, J. Diaz Rozo, A. Ogbechie, C. Puerto-Santana, y C. Bielza. *Industrial Applications of Machine Learning*. CRC Press, 2018.

Y.-N. Law y C. Zaniolo. An adaptive nearest neighbor classification algorithm for data streams. In *Knowledge Discovery in Databases*, pages 108–120. Springer, 2005.

D. Leite, P. Costa Jr, y F. Gomide. Evolving granular neural network for semi-supervised data stream classification. In *The 2010 International Joint Conference on Neural Networks*, pages 1–8, 2010.

D. F. Leite, P. Costa, y F. Gomide. Evolving granular classification neural networks. In *2009 International Joint Conference on Neural Networks*, pages 1736–1743, 2009.

V. Lemaire, C. Salperwyck, y A. Bondu. A survey on supervised classification on data streams. In *Business Intelligence: 4th European Summer School*, pages 88–125. Springer, 2015.

F. Li y Q. Liu. An improved algorithm of decision trees for streaming data based on VFDT. In *2008 International Symposium on Information Science and Engineering*, volume 1, pages 597–600, 2008.

X. Liao y L. Carin. Migratory logistic regression for learning concept drift between two data sets with application to UXO sensing. *IEEE Transactions on Geoscience and Remote Sensing*, 47:1454–1466, 2009.

E. Liberty, R. Sriharsha, y M. Sviridenko. An algorithm for online K-means clustering. In *Proceedings of the Eighteenth Workshop on Algorithm Engineering and Experiments*, pages 81–89, 2016.

S. P. Lloyd. Least squares quantization in PCM. Technical report, Bell Laboratories, 1957.

S. P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28(2):129–137, 1982.

V. Losing, B. Hammer, y H. Wersing. KNN classifier with self adjusting memory for heterogeneous concept drift. In *2016 IEEE 16th International Conference on Data Mining*, pages 291–300, 2016.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297. University of California Press, 1967.

S. Mansalis, E. Ntoutsi, N. Pelekis, y Y. Theodoridis. An evaluation of data stream clustering algorithms. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 11:167–187, 2018.

S. Mohanty, K. Nathrout, S. Barik, S. Das, y A. Prof. A study on evolution of data in traditional RDBMS to big data analytics. *International Journal of Advanced Research in Computer and Communication Engineering*, 4:230–232, 2015.

E. Morales y H. J. Escalante. Reglas de asociación. Instituto Nacional de Astrofísica, Óptica y Electrónica, 2009. URL `https://ccc.inaoep.mx/~emorales/Cursos/NvoAprend/Acetatos/reglasAsociacion.pdf`. Accessed: 08-07-2019.

A. Moujahid, I. Inza, y P. Larranaga. Tema 9. Inducción de Reglas. Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad del País Vasco, 2015. URL `http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t9-reglas`.

V. Nathan y S. Raghvendra. Accurate streaming support vector machines. *CoRR*, 2014.

H.-L. Nguyen, Y.-K. Woon, y W. K. Ng. A survey on data stream clustering and classification. *Knowledge and Information Systems*, 45:535–569, 2015.

U. Nodelman, C. R. Shelton, y D. Koller. Continuous time Bayesian networks. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, pages 378–387. Morgan Kaufmann Publishers Inc., 2002.

U. Nodelman, C. Shelton, y D. Koller. Learning continuous time Bayesian networks. In *Proceedings of Uncertainty in Artificial Intelligence 2003*, pages 451–458. Morgan Kaufmann Publishers Inc., 2003.

E. Ntoutsi, M. Schubert, y A. Zimek. Lecture 8: Velocity, Data Streams, Clustering. Ludwig-Maximilians-Universität München, Institut für Informatik, 2015. URL `http://www.dbs.ifi.lmu.de/Lehre/KDD_II/WS1516/skript/KDD2-4-DataStreamsClustering.pdf`.

L. O'Callaghan, N. Mishra, A. Meyerson, S. Guha, y R. Motwani. Streaming-data algorithms for high-quality clustering. In *Proceedings 18th International Conference on Data Engineering*, pages 685–694, 2002.

A. Oguntimilehin y O. Ademola. A review of big data management, benefits and challenges. *Journal of Emerging Trends in Computing and Information Sciences*, 5:433–438, 2014.

N. Oza y S. Russell. Online bagging and boosting. *Proceedings of Artificial Intelligence and Statistics*, pages 105–112, 2001.

Z. Pawlak. Rough sets. *International Journal of Computer & Information Sciences*, 11(5):341–356, 1982.

L. Perreault y J. Sheppard. Compact structures for continuous time Bayesian networks. *International Journal of Approximate Reasoning*, 109:19–41, 2019.

A. Pesaranghader, H. L. Viktor, y E. Paquet. McDiarmid drift detection methods for evolving data streams. *2018 International Joint Conference on Neural Networks*, pages 1–9, 2018.

phuong. Diagram of an artificial neural network, 2013. URL `https://tex.stackexchange.com/questions/132444/diagram-of-an-artificial-neural-network`.

J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., 1993.

P. Rai, H. Daumé III, y S. Venkatasubramanian. Streamed learning: One-pass SVMs. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence*, pages 1211–1216. Morgan Kaufmann Publishers Inc., 2009.

J. Read, F. Perez-Cruz, y A. Bifet. Deep learning in partially-labeled data streams. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 954–959. ACM, 2015.

J. W. Robinson y A. J. Hartemink. Non-stationary dynamic Bayesian networks. In *Advances in Neural Information Processing Systems 21*, volume 3, pages 1369–1376. Curran Associates, Inc., 2008.

J. Roure. Incremental learning of tree augmented naive Bayes classifiers. In *Proceedings of the 8th Ibero-American Conference on AI: Advances in Artificial Intelligence*, volume 2527, pages 32–41. Springer, 2002.

L. Rutkowski, L. Pietruczuk, P. Duda, y M. Jaworski. Decision trees for mining data streams based on the McDiarmid's bound. *IEEE Transactions on Knowledge and Data Engineering*, 25:1272–1279, 2013.

L. Rutkowski, M. Jaworski, L. Pietruczuk, y P. Duda. The CART decision tree for mining data streams. *Information Sciences*, 266:1–15, 2014.

C. Salperwyck, V. Lemaire, y C. Hue. Incremental weighted naive Bayes classifiers for data stream. In *Data Science, Learning by Latent Structures, and Knowledge Discovery*, pages 179–190. Springer, 2015.

J. C. Schlimmer y D. Fisher. A case study of incremental concept induction. In *Proceedings of the Fifth National Conference on Artificial Intelligence*, pages 496–501. AAAI Press, 1986.

A. Sharma. What is the difference between k-means and hierarchical clustering?, 2018. URL `https://www.quora.com/What-is-the-difference-between-k-means-and-hierarchical-clustering`. Accessed: 03-05-2019.

N. Sharma, S. Masih, y P. Makhija. A survey on clustering algorithms for data streams. *International Journal of Computer Applications*, 182:18–24, 2018.

J. Silva, E. Faria, R. Barros, E. Hruschka, A. de Carvalho, y J. Gama. Data stream clustering: A survey. *ACM Computing Surveys*, 46:13:1–13:31, 2014.

A. Smola y S. Vishwanathan. *Introduction to Machine Learning*. Cambridge University Press, 2008.

L. Song, M. Kolar, y E. P. Xing. Time-varying dynamic Bayesian networks. volume 22, pages 1732–1740, 2009.

E. Spyromitros-Xioufis, M. Spiliopoulou, G. Tsoumakas, y I. Vlahavas. Dealing with concept drift and class imbalance in multi-label stream classification. pages 1583–1588, 2011.

F. Stella y Y. Amer. Continuous time Bayesian network classifiers. *Journal of Biomedical Informatics*, 45(6):1108–1119, 2012.

F. Stella y S. Villa. Learning continuous time Bayesian networks in non-stationary domains. *Journal of Artificial Intelligence Research*, 57:1–37, 2016.

Y. Sun, Z. Wang, H. Liu, C. Du, y J. Yuan. Online ensemble using adaptive windowing for data streams with concept drift. *International Journal of Distributed Sensor Networks*, 2016:1–9, 2016.

Y. Sun, H. Shao, y S. Wang. Efficient ensemble classification for multi-label data streams with concept drift. *Information*, 10:158, 2019.

G. Trabelsi, P. Leray, M. Ben Ayed, y A. M. Alimi. Dynamic MMHC: A local search algorithm for dynamic Bayesian network structure learning. In *Advances in Intelligent Data Analysis XII*, pages 392–403. Springer, 2013.

C.-J. Tsai, C.-I. Lee, y W.-P. Yang. An efficient and sensitive decision tree approach to mining concept-drifting data streams. *Informatica*, 19:135–156, 2008.

I. Tsamardinos, L. E. Brown, y C. F. Aliferis. The max-min hill-climbing Bayesian network structure learning algorithm. *Machine Learning*, 65(1):31–78, 2006.

I. W. Tsang, J. Kwok, y P.-M. Cheung. Very large SVM training using core vector machines. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, pages 349–356, 2005.

I. W. Tsang, A. Kocsor, y J. T. Kwok. Simpler core vector machines with enclosing balls. In *Proceedings of the 24th International Conference on Machine Learning*, pages 911–918. ACM, 2007.

A. Unagar y A. Unagar. Support vector machines, 2017. URL `https://medium.com/data-science-group-iitr/support-vector-machinessvm-unraveled-e0e7e3ccd49b`.

P. E. Utgoff. ID5: An Incremental ID3. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 107–120. Morgan Kaufmann Publishers Inc., 1988.

P. E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4(2):161–186, 1989.

P. E. Utgoff, N. C. Berkman, y J. A. Clouse. Decision tree induction based on efficient tree restructuring. *Machine Learning*, 29(1):5–44, 1997.

D. Wang, B. Zhang, P. Zhang, y H. Qiao. An online core vector machine with adaptive MEB adjustment. *Pattern Recognition*, 43(10):3468–3482, 2010.

J. Wang, P. Neskovic, y L. N. Cooper. Neighborhood size selection in the k-nearest-neighbor rule using statistical confidence. *Pattern Recognition*, 39(3):417–423, 2006.

Wikipedia. Incremental learning, 2019a. URL `https://en.wikipedia.org/wiki/Incremental_learning`.

Wikipedia. Sigmoid function, 2019b. URL `https://en.wikipedia.org/wiki/Sigmoid_function`. Accessed: 02-05-2019.

Wikipedia, 2019c. URL `https://es.wikipedia.org/wiki/Minería_de_datos`.

D. H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

H. Yang y S. Fong. Incremental optimization mechanism for constructing a decision tree in data stream mining. *Mathematical Problems in Engineering*, 2013:1–14, 2013.

A. Yasin. *Incremental Bayesian network structure learning from data streams.* PhD thesis, Université de Nantes, 2013.

A. Yasin y P. Leray. Incremental Bayesian network structure learning in high dimensional domains. In *5th International Conference on Modeling, Simulation and Applied Optimization*, pages 1–6. IEEE, 2013.

S. Zhang, X. Li, M. Zong, X. Zhu, y R. Wang. Efficient kNN classification with different numbers of nearest neighbors. *IEEE Transactions on Neural Networks and Learning Systems*, 29(5):1774–1785, 2018.

T. Zhang, R. Ramakrishnan, y M. Livny. BIRCH: An efficient data clustering method for very large databases. *SIGMOD Record (ACM Special Interest Group on Management of Data)*, 25:103–114, 1996.

Y. Zhu y D. Shasha. Chapter 32 - StatStream: Statistical monitoring of thousands of data streams in real time. In *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, pages 358–369. Morgan Kaufmann Publishers Inc., 2002.

I. Zliobaite. Learning under concept drift: An overview. *CoRR*, abs/1010.4784, 2010.