UCLV
Universidad Central
"Marta Abreu" de Las Villas

MFC
Facultad de Matemática
Física y Computación

Departamento de Ciencia de la Computación

TRABAJO DE DIPLOMA

Título: Estudio e implementación en MOA de una nueva propuesta del método Bagging para combinar clasificadores base.

Autor: Rosa María Campos Gómez

Tutores: Ing. Alberto Verdecia Cabrera
Dra. Yanet Rodríguez Saravia

Santa Clara, Junio 2018

Copyright©UCLV

Declaración

Este documento es Propiedad Patrimonial de la Universidad Central “Marta Abreu” de Las Villas, y se encuentra depositado en los fondos de la Biblioteca Universitaria “Chiqui Gómez Lubian” subordinada a la Dirección de Información Científico Técnica de la mencionada casa de altos estudios. Se autoriza su utilización bajo la licencia siguiente: Atribución- No Comercial-Compartir Igual

Para cualquier información contacte con: Dirección de Información Científico Técnica. Universidad Central “Marta Abreu” de Las Villas. Carretera a Camajuaní. Km 5½. Santa Clara. Villa Clara. Cuba. CP. 54 830 Teléfonos.: +53 01 42281503-1419

Me gustaría dedicar este trabajo a mi familia y a todas las personas que de una forma u otra aportaron en mi formación y crecimiento como estudiante universitaria.

AGRADECIMIENTOS

Me gustaría agradecer a mi mamá en primer lugar pues siempre me ha apoyado en todas mis decisiones y me ha ayudado a enfrentarlas cualquiera que fueran sus resultados.

Quiero darles las gracias también a todos mis compañeros de aula especialmente aquellos con los que siempre estudiaba que me permitieron entrar en su grupo y así hacer que yo estudiara un poco más para poder lograr este objetivo tan deseado, en especial a Francisco Noelvis Riestra y a Dario Deya que nunca tuvieron un no por respuesta a la hora de aclararme una duda.

A los profesores en general pues con mucho sentido de pertenencia y respeto a la carrera me impartieron clases.

Y a mi tutor por indicarme el procedimiento a seguir para lograr la realización de este trabajo.

En la actualidad, el volumen de los datos generados por sensores, Internet, dispositivos de localización, telefonía y muchos otros, esta en constante aumento. El tamaño de estos datos es potencialmente infinito debido a su constante generación, por lo que es necesario procesarlos de forma eficiente utilizando un límite escaso de memoria en un límite reducido de tiempo. Los Sistemas Multiclasificadores proporcionan un mecanismo eficaz de combinar los resultados de un conjunto de clasificadores obteniendo un modelo más complejo que un clasificador simple logrando mejor precisión en la predicción. Existen varios tipos de multiclasificadores en este trabajo se profundiza en el estudio de los multiclasificadores incrementales Bagging y Boosting, el método Bagging combina los resultados de clasificadores entrenados con conjuntos de entrenamientos generados por la técnica de bootstraping, los clasificadores obtenidos mediante esta técnica son independientes entre sí. Bagging obtiene los resultados de la clasificación a través de una votación siguiendo el consenso de prulabilidad. En estudios realizados en el año 1997 Lam y Suen [30] estudiaron el voto mayoritario como método de combinación de un número par o impar de clasificadores que pueden votar por una de las k clases o abstenerse. Con el estudio del voto mayoritario obtienen algunos resultados interesantes, llegan a la conclusión que el voto mayoritario no es una buena técnica de combinación cuando nos encontramos en presencia de clasificadores individuales.

Análisis Masivo Online(MOA) es un entorno de software para la implementación de algoritmos permitiéndonos llevar a cabo experimentos para el aprendizaje en línea con flujos de datos. MOA incluye una colección de métodos tradicionales y online, así como herramientas para la evaluaciófile. Entre estos métodos se encuentran versiones en linea del método Bagging, se ha hecho evidente la necesidad de estudiar y experimentar distintas formas de combinación para este método pues obtiene clasificadores independientes. Para dicho estudio se propone en este trabajo la realización de una nueva versión en linea del algoritmo Bagging para la herramienta MOA que no combine el resultado de los clasificadores mediante la votación mayoritaria, se proponen como métodos de combinación Naıve Bayes (NB), EWMA, Entropy Weighting (EW) y Dempster–Shafer (DS). Se logra incluir este nuevo método mediante la realización de un nuevo clasificador quedando implementado en el subpaquete moa.classifiers.meta, donde la clase principal que la representa se denomina AbstractClassifier y fue compilado con la versión 1.8 de la máquina virtual de Java.

Para la evaluación de este nuevo método fue comparado empíricamente con varios algoritmos de ensamble de la familia bagging, utilizando Naive Bayes como clasificador base. Los experimentos incluyeron conjuntos de datos artificiales y reales. Todos los algoritmos fueron probados frente a los tipos de cambios comunes (abruptos y graduales). Los experimentos mostraron que el

nuevo algoritmo es una buena opción para el aprendizaje a partir de flujos de datos con cambios de concepto.

ABSTRACT

At the present time, the volume of the data generated by sensors, Internet, localization devices, telephony and many other, this in constant increase. The size of these data is potentially infinite due to its constant generation, for what is necessary to process them in an efficient way using a scarce limit by heart in a reduced limit of time. The Systems Multiclasificadores provides an effective mechanism of combining the results of a group of classifiers obtaining a more complex model that a simple classifier achieving better precision in the prediction. Several multiclasificadores types exist in this work it is deepened in the study of the incremental multiclasificadores Bagging and Boosting, the method Bagging combines the results of classifiers trained with groups of trainings generated by the bootstraping technique, the classifiers obtained by means of this technique are independent to each other. Bagging obtains the results of the classification through a voting following the prulabilidad consent. In studies carried out in the year 1997 Lam and Suen [30] did they study the majority vote as method of combination of an even or odd number of classifiers that you/they can vote for one of those?? classes or to abstain. With the study of the majority vote they obtain some interesting results, they arrive to the conclución that the majority vote is not a good combination technique when we are in presence of individual classifiers. Massive analysis Online(MOA) it is a software environment for the implementation of algorithms allowing us to carry out experiments for the on-line learning with flows of data. MOA includes a collection of traditional methods and online, as well as tools for the evaluation. Among these methods they are versions in it lines of the method Bagging, it has become evident the necessity to study and to experience different combination forms for this since method obtains independent classifiers. For this study he/she intends in this work the realization of a new version in it lines of the algorithm Bagging for the tool MOA that doesn't combine the result of the classifiers by means of the majority voting, they intend as combination methods Naive Bayes (NB), EWMA, Entropy Weighting (EW) and Dempster–Shafer (DS). it is possible to include this new method by means of the realization of a new classifier being implemented in the subpaquete moa.classifiers.meta, where the main class that represents it is denominated AbstractClassifier

and it was compiled with the version 1.8 of the virtual machine of Java. For the evaluation of this new method it was compared empirically with several algorithms of it assembles of the family bagging, using Naive Bayes like classifier bases. The experiments included groups of artificial and real data. All the algorithms were proven in front of the types of common changes (abrupt and gradual). The experiments showed that the new algorithm is a good option for the learning starting from flows of data with concept changes.

ÍNDICE

FIGURAS **13**

TABLAS **15**

INTRODUCCIÓN **1**

1. Acerca de los multiclasificadores y métodos de combinación **5**

1.1. Clasificadores individuales 5

1.1.1. Redes neuronales artificiales 5

1.1.2. Naïve Bayes 6

1.1.3. Árboles de decisión 7

1.2. Multiclasificadores incrementales 7

1.2.1. Métodos para la generación de sistemas multiclasificadores 8

1.2.1.1. Bagging y Boosting 10

1.3. Métodos para combinar clasificadores base 11

1.3.1. Funciones de combinación 14

1.3.1.1. Combinación de salidas de tipo abstracto 14

1.3.1.2. Combinación de salidas de tipo rango 16

1.4. Consideraciones finales del capítulo 18

2. Herramienta MOA: Bagging y combinación de clasificadores base **21**

2.1. Descripción de la herramienta MOA 21

2.1.1. Creación de un nuevo clasificador en MOA. 23

2.1.2. MOA a partir de WEKA 25

2.2. Nuevo enfoque para el uso de métodos de combinación de clasificadores base en bagging. 26

2.2.1. Naıve Bayes (NB) 26
2.2.2. EWMA 26
2.2.3. Entropy Weighting (EW) 27
2.2.4. Dempster–Shafer (DS) 27
2.2.5. Algoritmo propuesto 28
2.3. Diseño e implementación de los métodos de combinación de clasificadores base. 29
2.4. Conclusiones finales del capítulo 33

3. Validación de bagging con la incorporación de varias formas de combinación de salidas de clasificadores 35
3.1. Evaluación de los algoritmos incrementales 35
3.1.1. Métricas 36
3.1.2. Metodología 37
3.2. Diseño experimental 39
3.3. Características principales de los conjuntos de datos a utilizar en la experimentación 40
3.3.1. Configuración de los algoritmos 40
3.3.2. Experimentos frente a cambios abruptos y graduales 43
3.3.3. Experimentos con conjuntos de datos reales 45
3.4. Conclusiones finales del capítulo 46

4. Concluciones 49

REFERENCIAS 51

FIGURAS

2.1. Ciclo de clasificación de flujo de datos 22

3.1. Ranking de los algoritmos. Los algoritmos que no son significativamente diferente están conectados 42
3.2. Precisión de los algoritmos frente a cambios abruptos. Se generaron 10 cambios cada 25,000 instancias. 44
3.3. Precisión de los algoritmos en conjuntos de datos reales. 44
3.4. Ranking de los algoritmos. Los algoritmos que no son significativamente diferente están conectados. El ranking se calculó con respecto a las Tablas 3.2, 3.3, 3.4, 3.5 y 3.6. 45

TABLAS

3.1. Principales características de los conjuntos de datos utilizados en los experimentos 41
3.2. Resultados los algoritmos frente a cambios abruptos 42
3.3. Resultados de los algoritmos frente a cambios graduales. El número de instancias en el período de transición fue 100 42
3.4. Resultados de los algoritmos frente a cambios graduales. El número de instancias en el período de transición fue 500 43
3.5. Resultados de los algoritmos frente a cambios graduales. El número de instancias en el período de transición fue 1000 45
3.6. Resultados de los algoritmos frente a conjuntos de datos reales. 46

INTRODUCCIÓN

INTRODUCCIÓN

Antecedentes En la actualidad, el volumen de los datos generados por sensores, Internet, dispositivos de localización, telefonía y muchos otros, esta en constante aumento. El tamaño de estos datos es potencialmente infinito debido a su constante generación, por lo que es necesario procesarlos de forma eficiente utilizando un límite escaso de memoria en un límite reducido de tiempo. Para este procesamiento es factible el uso de técnicas de aprendizaje automático.En los campos de reconocimiento de patrones y aprendizaje de máquinas la clasificación supervisada es una de las tareas más demandadas. Se está en presencia de un problema de clasificación supervisada cuando se necesita saber a qué clase pertenece un objeto nunca antes visto, teniendo en cuenta cierta experiencia previa..La clasificación es un proceso que consta de dos pasos fundamentales. En el primer paso es donde se construye un modelo que describe un conjunto predeterminado de datos, clases o conceptos. Este modelo se construye tupla a tupla, describiendo en cada caso los atributos de la base de datos que se analiza. Se asume que cada tupla pertenece a una clase predefinida, que está determinada por uno de los atributos llamado atributo clase. Las tuplas dentro de la clasificación son también conocidas como ejemplos, instancias o experiencias. El conjunto de ejemplos analizados para construir el modelo forman el conjunto de datos de entrenamiento. Los ejemplos individuales que forman el conjunto de entrenamiento son seleccionados aleatoriamente de la población muestra asignándole a cada uno, una etiqueta de clase, teniendo lugar el aprendizaje supervisado.

Los Sistemas Multiclasificadores en la actualidad han cobrado importancia ya que proporciona un mecanismo que combina de manera eficaz un conjunto de clasificadores obteniendo un modelo más complejo que un clasificador simple pero también más preciso. La principal tarea de los sistemas multiclasificadores es la mejora de la precisión con respecto a un clasificador individual (Clasificador Básico) mediante la combinación con otros modelos tratando así de que se eliminen errores en la predicción de observaciones (Hansen &Salamon 1990) en el modelo

obtenido. Esta mejora tiene su explicación lógica, podemos decir que un clasificador individual puede equivocarse ante una observación en concreto, mientras que al usar un conjunto de clasificadores existe la posibilidad de que la mayoría de los clasificadores logren alcanzar la predicción correcta, estas predicciones dadas por la mayoría se pueden combinar para así evitar y reducir errores individuales. Según la homogeneidad de los modelos que componen el sistema multiclasificador(Gama 1999), se diferencian los que combinan clasificadores básicos homogéneos y los que combinan clasificadores heterogéneos. En cuanto a los modelos que pueden formar parte de del sistema multiclasificador siempre que sean precisos y diferentes entre ellos: redes neuronales (Hansen &Salamon 1990), clasificadores ingenuos de Bayes Zheng (1998), árboles de decisión Dietterich (2000c), máquinas de vectores soporte (Burges 2003), etc. Como exponentes más reconocidos y utilizados entre los que usan modelos homogéneos tenemos "bagging" (Breiman 1996) y "boosting" (Freund &Schapire 1996; Freund &Schapire 1997). Por su parte, en el grupo de métodos que usan modelos heterogéneos en la composición del sistema multiclasificador podemos destacar el método de "generalización apilada" ("stacked generalization") o el de "generalización en cascada" ("cascade generalization").

Dependiendo de como se presenten los ejemplos de entrenamiento, el aprendizaje automático se puede clasificar en dos tipos : aprendizaje por lotes y aprendizaje en linea. En el aprendizaje por lotes, los datos deben ser recolectados y almacenados antes de ser procesados, por lo que su uso no es factible para procesar datos generados constantemente en el tiempo. Sin embargo, el aprendizaje en linea permite procesar flujos de datos de manera secuencial . En las tareas de clasificación, un flujo de datos es comúnmente definido como una secuencia muy grande (potencialmente infinita) de pares que se van adquiriendo a lo largo del tiempo. Estos pares, llamados instancias o ejemplos, están compuestos por un conjunto de atributos y una etiqueta de clase. Debido a la dimensión temporal de los datos y la dinámica de muchas situaciones reales, la distribución de probabilidad que regula a los mismos puede cambiar con el tiempo, un problema conocido comúnmente como cambio de concepto. Consecuentemente, los algoritmos de aprendizaje para la minera de flujos de datos deben ser actualizados con respecto a los conceptos mas recientes. Los métodos de ensambles de clasificadores han recibido en los últimos tiempos gran atención para el modelado y la clasificación de flujos de datos no estacionarios. Los métodos de ensambles combinan las predicciones de los clasificadores base con el objetivo de alcanzar una precision mas alta que la de estos clasicadores base del ensamble de forma individual. Con el fin de manipular cambios de concepto, utilizan medidas de rendimiento para monitorizar la consistencia del ensamble en relacion con los nuevos datos. Variaciones significativas en los valores de rendimiento se interpretan como un cambio de concepto y los métodos de ensamble

eliminan, reactivando o a~naden nuevos clasificadores base dinámicamente en respuesta a estas variaciones.

Leo Breiman propone el algoritmo Bagging como una estrategia de obtención de los clasificadores individuales de un esquema. El término Bagging proviene de la frase en inglés Bootstrap Aggregating, que deja claro que el método combina los resultados de clasificadores entrenados con conjuntos de entrenamientos generados por la técnica de bootstraping, es decir se crean conjuntos de entrenamientos realizando muestreos con reemplazos de un conjunto original. Cada clasificador individual es obtenido a partir de un algoritmo de clasificación que se denotará por A y que es parámetro del método Bagging, luego en la etapa de clasificación se combinan los resultados a través de una votación y siguiendo el consenso de la plurabilidad.

MOA (Massive Online Analysis) (Bifet & Gavalda, 2007) es uno de los más completos entornos de trabajos para minería de datos que está relacionado con el proyecto WEKA (Waikato Environment for Knowledge Analysis) (Hall et al., 2009). MOA incluye una colección de algoritmos para el aprendizaje automático (Clasificación, regresión y agrupamiento) para trabajar sobre flujos de datos entre ellos implementa versiones incrementles de los algoritmos Bagging y Boosting entre ellos el OzaBag y el OzaBoost esarrolladas por Oza & Russell (2001). Además, cuenta con herramientas para evaluar los algoritmos y generar datos (STAGGER, LED, SEA, Rotating Hyperplane, Random RBF entre otras) de forma artificial, incluyendo la posibilidad de simular cambios de conceptos. Este trabajo tiene como objetivo proponer una nueva version del algoritmo Bagging para la herramienta MOA pues las versiones existentes solo tienen en cuenta como metodo de combinacion de los resultados de los clasificadores base al voto mayoritario.

Los algoritmos basados en votación consideran la salida de cada clasificador como un voto, devido a la independencia entre los clasificadores esta no es la mejor situación que se puede tener. En estudios realizados por Kuncheva se ha llegado a la conclucion de que si el voto mayoritario va a ser usado, en lugar de trabajar por obtener clasificadores independientes es más provechoso trabajar por obtener clasificadores que registren el patrón de éxito pero la independencia como el patrón de éxito son condiciones imposibles de alcanzar en la prácticar. Por lo que se hace notable la necesidad del estudio y experimentacion del metodo Bagging con la incorporacion de nuevos metodos de combinacion.

Planteamiento del problema La herramienta MOA es ampliamente utilizada para evaluar multiclasificadores; sin embargo, solo ofrece una forma de combinar las salidas de los clasificadores base para el caso del multiclasificador bagging, lo que limita los resultados de esta clasificación sobre determinados conjuntos de datos.

Por lo que podemos llegar a la siguientes preguntas de investigación

1. ¿Cuáles son las principales formas de combinación de clasificadores base existentes?
2. ¿Cómo lograr la incorporcion de las principales formas de combinar clasificadores base al entorno de la herramienta MOA?
3. ¿Qué medidas de validación se podría usar para el análisis de los resultados obtrenidos a partir del uso de diversas formas de combinación de salidas de clasificadores base?

Objetivo general

1. Evaluar el multiclasificador bagging a partir del estudio de varias formas de combinación de salidas de los clasificadores base.

Objetivos específicos

1. Estudiar las principales formas de combinación de clasificadores base.
2. Incorporar a la herramienta MOA las principales formas de combinar clasificadores base.
3. Validar los resultados obtenidos por el multiclasificador bagging a partir del uso de diversas formas de combinación de salidas de clasificadores base aplicados a colecciones de datos internacionales.

Estructura de la investigación El Capitulo1 abordaremos sobre los diferentes tipos de multiclasificadores y lo s métodos de combinación existentes, que serian los multiclasificadores clásicos, multiclasificadores incrementales y los métodos para combinar diferentes tipos de clasificadores base. El Capitulo2 se centra en el estudio de la herramienta MOA que es utilizada para evaluar los algoritmos que aprenden a partir de flujos de datos, proponiendo un nuevo enfoque para el uso de métodos de clasificadores base para el algoritmo Bagging y el diseño e implementación de los mismos.En el Capitulo3 se validara el resultado de la incorporación de los nuevos métodos de combinación para el clasificador Bagging apoyandonos en medidas de validación ya existentes para estos tipos de multiclasificadores.

CAPÍTULO 1

Acerca de los multiclasificadores y métodos de combinación

1.1. Clasificadores individuales

En esta sección se presentan varios de los modelos individuales de aprendizaje que son utilizados para clasificación de flujos de datos no estacionarios.

1.1.1. Redes neuronales artificiales

Las redes neuronales artificiales han sido exitosamente utilizadas en la minería de datos. En las aplicaciones tradicionales de minería de datos las redes neuronales son entrenadas utilizando un conjunto de entrenamiento fijo. Sin embargo, la mayoría de los modelos de redes neuronales requieren varias pasadas sobre el conjunto de entrenamiento para ajustar los pesos de las neuronas, usualmente se utiliza el algoritmo backpropagation. Sin embargo en la minería de flujos de datos los ejemplos son vistos una sola vez por el modelo y es necesario mantener acostado el costo computacional para actualizar los pesos de las neuronas con una sola pasada sobre los datos. Un ejemplo para adaptar una red para aprender a partir de flujos datos es utilizar un mecanismo de olvido, es decir cuando la red no se entrena con los mismos datos múltiples veces, esta se ajustará dinámicamente a los datos que se van obteniendo en el tiempo, por tanto pueden reaccionar de forma natural al cambio de concepto. Uno de los sistemas más conocidos capaz de manipular cambio es FRANN (*Floating Rough Approximation in Neural Network*) (Kubat y Widmer (1995a)).

1.1.2. Naïve Bayes

Este modelo se basa en el teorema de Bayes y calcula la probabilidad condicional de cada una de las clases para cada ejemplo nuevo de entrenamiento asumiendo que los atributos son independientes dada la clase. El algoritmo Naïve Bayes aprende eficientemente a partir de flujos de datos no estacionarios con complejidad computacional constante

Cada ejemplo de entrenamiento $e_i = (\overrightarrow{x}, y_i)$ está formado por un vector $\overrightarrow{x}$ y un valor discreto y_i, llamado etiqueta y tomado de un conjunto $\mathscr{Y}$ denominado clase. el clasificador Naïve Bayes se basa en encontrar la clase más probable que describa a ese ejemplo. La clase más probable será la que cumpla:

$$y_p = argmax_{y_i \in \mathscr{Y}} P(y_i \mid \overrightarrow{x}) \tag{1.1}$$

es decir dada la probabilidad de los atributos que describen el ejemplo e_i, éste pertenezca a la clase y_i. Por el teorema de Bayes:

$$y_p = argmax_{y_i \in \mathscr{Y}} \frac{P(\overrightarrow{x} \mid y_i) P(y_i)}{P(\overrightarrow{x})} \tag{1.2}$$

$$y_p = argmax_{y_i \in \mathscr{Y}} P(\overrightarrow{x} \mid y_i) P(y_i) \tag{1.3}$$

Se puede estimar $P(y_i)$ contando las veces que aparece y_i en el conjunto de entrenamiento y dividiéndolo por el número total de ejemplos que forman este conjunto. Para estimar el término $P(\overrightarrow{x} \mid y_i)$ se debe recorrer todo el conjunto de entrenamiento, lo que requiere un alto costo computacional. Naïve Bayes simplifica el problema asumiendo que los atributos son independientes dada la clase. Así la probabilidad de que un ejemplo no etiquetado $\overrightarrow{x}$ pertenezca a la clase y_i está dada por:

$$P(\overrightarrow{x} \mid y_i) = \prod_j P(x_j \mid y_i) \tag{1.4}$$

La estimación de los parámetros del modelo (por ejemplo, $P(y_i)$ y $P(\overrightarrow{x})$) se pueden aproximar con frecuencias relativas del conjunto de entrenamiento. Estas son las estimaciones de máxima verosimilitud de las probabilidades. Las probabilidades de las clases se pueden calcular asumiendo clases con igual probabilidad. Para estimar las probabilidades de cada uno de los atributos generalmente se utilizan la distribución de binomial para atributos discretos y la distribución normal cuando se trata con atributos continuos.

1.1.3. Árboles de decisión

La construcción de árboles de decisión ha sido ampliamente estudiada en la comunidad del aprendizaje automático. En la minería de flujos de datos, la desigualdad de probabilidades de Hoeffding (Del Campo Ávila (2007)(Frias-Blanco et al. (2015)(Frías-Blanco, del Campo-Ávila, Ramos-Jiménez, Carvalho, Ortiz-Díaz y Morales-Bueno (2016a)))), de Chernoff (Del Campo Ávila (2007))(Frías-Blanco, del Campo-Ávila, Ramos-Jiménez, Carvalho, Ortiz-Díaz y Morales-Bueno (2016a)) han sido herramientas poderosas para la inducción en línea de árboles de decisión. La cota Hoeffding indica que con probabilidad $1-\delta$, la media real de una variable aleatoria con rango R no difiere de la media estimada después de n observaciones independientes por más de:

$$\varepsilon = \sqrt{\frac{R^2 ln\left(\frac{1}{\delta}\right)}{2n}} \tag{1.5}$$

Utilizando esta cota, Domingos y Hulten propusieron VFDT (*Very Fast Decision Tree*) (Hulten et al. (2001a)). VFDT induce incrementalmente un árbol de decisión a partir de un flujo de datos, sin la necesidad de almacenar ejemplos de entrenamiento después de haber sido utilizados para actualizar el árbol. Este algoritmo es similar a los algoritmos clásicos de inducción de árboles de decisión, solo difiere en la selección del atributo para dividir. En lugar de seleccionar el mejor atributo después de ver todos los ejemplos, VFDT utiliza la cota de Hoeffding para determinar cuántas instancias deben ser muestreadas antes de poder elegir, con probabilidad $1-\delta$, qué atributo debe ser usado para la expansión de cada nodo.

1.2. Multiclasificadores incrementales

Los sistemas multiclasificadores (Dietterich (2000a)) se han revelado como un mecanismo eficaz que combina un conjunto de clasificadores para producir un modelo algo más complejo pero también más preciso. La principal idea que subyace a los sistemas multiclasificadores es la posibilidad de mejorar la precisión de un modelo individual (clasificador básico) combinándolo con otros, sean del mismo tipo o de diferentes tipos, y pretendiendo que, en conjunto, sean capaces de evitar determinados errores en la predicción de observaciones (Hagerup y Rüb (1990)). De forma intuitiva se puede explicar esta mejora diciendo que, para una observación en concreto, es posible que un clasificador individual se equivoque, pero al usar más de un clasificador, es posible que la predicción correcta sea alcanzada por la mayoría de clasificadores, aunque exista una

minoría que se equivoque. Combinando esas predicciones, la ofrecida en común por la mayoría de clasificadores básicos puede evitar errores individuales. El principal objetivo de los sistemas multiclasificadores es aumentar el poder predictivo, pero se consigue a costa de mermar el poder descriptivo. Estos sistemas alcanzan, por lo general, una precisión mayor que la que alcanzan los clasificadores básicos individualmente, pero la interpretabilidad de un clasificador básico es, típicamente, mayor que la que se pueda alcanzar teniendo que combinar múltiples modelos. A pesar de este aumento en la dificultad para interpretar los sistemas multiclasificadores, existe una ventaja y es que se aumenta el poder expresivo, es decir, el modelo multiclasificador puede sintetizar conocimiento que antes no podía ser recogido en un único modelo básico. Algo fundamental para inducir sistemas multiclasificadores de calidad, es decir, para generar modelos más precisos, es inducir clasificadores básicos que sean precisos y diferentes entre ellos (Hagerup y Rüb (1990)). Al decir que deben ser precisos nos referimos a que su tasa de error sea mejor que la propia de hacer predicciones aleatorias y con modelos diferentes nos referimos a que estos modelos deben cometer errores diferentes para diferentes subconjuntos del espacio de búsqueda (errores no correlacionados (Jacobs (1995))). Si todos los modelos en el sistema multiclasificador cometiesen los mismos errores para las mismas experiencias, no tendría sentido disponer de múltiples modelos. El objetivo de tener varios, como ya hemos indicado, es que para los casos en los que un clasificador básico pueda equivocarse, existan otros que puedan darnos alternativas correctas. A la hora de diseñar algoritmos que generen sistemas multiclasificadores hay que considerar dos puntos fundamentales (Gama (1999))(Hernández Orallo et al. (2004)): cómo son generados los distintos clasificadores básicos que componen el sistema multiclasificador y cómo son combinados para realizar una predicción conjunta.

1.2.1. Métodos para la generación de sistemas multiclasificadores

Los trabajos en el ámbito de los sistemas multiclasificadores han sido numerosos y siguen surgiendo debido a su interés y a las ventajas que aportan. Es difícil recoger todos los enfoques propuestos de forma exhaustiva, pero podemos presentar diferentes características propias de los mismos que sirven para categorizarlos. Si atendemos a la homogeneidad de los modelos que componen el sistema multiclasificador (Gama (1999)) podemos diferenciar entre aquellos que combinan clasificadores básicos homogéneos y aquellos que combinan clasificadores heterogéneos.

Existe una gran variedad de modelos que pueden formar parte del sistema multiclasificador siempre que sean precisos y diferentes entre ellos: redes neuronales (Freund y Schapire (1997a)),

clasificadores ingenuos de Bayes (Zheng (1998)), árboles de decisión (Dietterich (2000c)), máquinas de vectores soporte Kim et al. (2003), etc. Como exponentes más reconocidos y utilizados entre los que usan modelos homogéneos tenemos "bagging" (Breiman (1996)) y "boosting" (Freund et al. (1996))(Freund y Schapire (1997a)). Por su parte, en el grupo de métodos que usan modelos heterogéneos en la composición del sistema multiclasificador podemos destacar el método de "generalización apilada" ("stacked generalization") (Wolpert (1992)) o el de "generalización en cascada" ("cascade generalization") (Gama y Brazdil (2000)). Como ya se ha expuesto, la diversidad de los modelos, sean del mismo tipo (homogéneos) o de tipos diferentes (heterogéneos), es un requisito necesario para conseguir sistemas multiclasificadores. Para alcanzar dicha diversidad, los algoritmos de aprendizaje deben lograr, mediante alguna fórmula, que los modelos que se generen no sean idénticos. Basándose en los diferentes elementos que pueden ser manipulados para conseguir que los algoritmos induzcan modelos diferentes, Dietterich propuso otra clasificación con cuatro grupos (Dietterich (2000a)): los métodos que manipulan los conjuntos de datos de entrenamiento, los que manipulan los atributos de entrada, los que manipulan las clases y los que introducen algún tipo de aleatoriedad en el proceso de aprendizaje. Bagging y boosting forman parte del grupo de algoritmos que consiguen la variedad manipulando el conjunto de datos de entrenamiento, aunque lo hacen de forma diferente: bagging no modifica la distribución de los datos de entrenamiento, mientras que boosting sí lo hace. La idea de modificar el conjunto de experiencias hay que combinarla con el uso de algoritmos que sean bastante sensibles a pequeñas variaciones del mismo, con el objetivo de producir modelos bastante diferentes con pequeñas variaciones en el conjunto de entrenamiento. A los algoritmos que presentan esta "inestabilidad" frente a pequeñas variaciones se les conoce como "aprendices débiles". En el grupo de métodos que manipulan los atributos de entrada podemos destacar los "bosques de decisión" (Ho (1998)) que seleccionan un pequeño conjunto de dimensiones. Aunque la clase podría considerarse como atributo de entrada y su manipulación ser englobada en el grupo anterior, ocupa un lugar diferente en esta clasificación puesto que se considera a la clase como el objetivo de salida. Uno de los métodos perteneciente a este grupo sería el propuesto por el propio Dietterich y por Bakiri (Dietterich y Bakiri (1994)) en el que se generan subconjuntos mediante la partición aleatoria de las clases del conjunto original. En el último grupo de esta clasificación se introduce aleatoriedad a la hora de tomar las decisiones en el algoritmo de aprendizaje. En este grupo podemos destacar el algoritmo de "aleatorización" ("randomization"), también propuesto por Dietterich (Dietterich (2000c)), en el que la expansión de una hoja por un atributo se hace de forma aleatoria seleccionando de entre los mejores atributos disponibles. Otro ejemplo en este mismo grupo es la inducción de "multi-árboles de decisión" (Estruch et al. (2003)) donde la

aleatoriedad se introduce a la hora de seleccionar la partición con la que continuar la generación de la estructura.

1.2.1.1. Bagging y Boosting

Puesto que bagging y boosting han sido dos de los mecanismos más extendidos y sobre los que se han desarrollado multitud de variantes, procedemos a dar los detalles más significativos de cada uno de ellos. Además, como se verá en las siguientes secciones, parte de las aportaciones que presentaremos se basan en el esquema de bagging y en los resultados experimentales hemos usado ambos enfoques como modelos de referencia.

El método bagging, propuesto por Breiman (Breiman (1996)), es un método sencillo para generar diversos clasificadores a partir de un conjunto de datos. Su nombre viene de bootstrap aggregation, que resume la filosofía del método: generar diferentes subconjuntos de entrenamiento realizando muestreo con reemplazamiento a partir del original. Sería lógico pensar que si los subconjuntos generados son parecidos, los modelos inducidos usando un mismo algoritmo también fuesen parecidos y no cumpliesen con el requisito de ser diferentes entre ellos. Pero esto no ocurre siempre de este modo, ya que existen algoritmos de inducción que son muy sensibles a pequeñas variaciones en el conjunto de entrenamiento, por ejemplo, algunos algoritmos de inducción de árboles de decisión. Esta característica de inestabilidad es la que da nombre a los algoritmos conocidos como “aprendices débiles” y se ve acentuada cuando el subconjunto de entrenamiento tiene un tamaño limitado. Normalmente, el tipo de combinación de las predicciones que se usa es la votación mayoritaria: se elige la clase que haya sido seleccionada por más clasificadores individuales de entre los que componen el sistema multiclasificador.

El método boosting, propuesto por Freund y Schapire (Freund et al. (1996))(Freund y Schapire (1997a)), asigna pesos a las experiencias en el conjunto de entrenamiento. El objetivo del método boosting es inducir un modelo que minimice la suma de los pesos de las experiencias que no se clasifiquen correctamente. En un principio, los pesos de las diferentes experiencias tendrán el mismo valor y los errores serán igual de importantes. Pero en cada iteración, los pesos de las experiencias que han sido incorrectamente clasificadas se aumentarán para darles más importancia. De esta forma, en cada iteración se le da más importancia a las experiencias que no han sido clasificadas correctamente en pasos anteriores. En este modelo, la combinación de los clasificadores básicos no es tan simple como con bagging. Ahora se tiene en cuenta el error estimado de cada modelo para calcular la clase de la observación que se está intentando predecir. Sumando los pesos de los modelos que eligen la misma clase, se selecciona aquella que acumule mayor valor.

Existen diversos trabajos que comparan los resultados alcanzados por bagging y boosting bajo diferentes circunstancias (Quinlan y others (1996))(Bauer y Kohavi (1999))(Dietterich (2000c)). Como conclusiones generales llegan a decir que boosting alcanza mejores resultados que bagging, pero no en todos los casos. Los casos en los que boosting no alcanza resultados tan buenos son los problemas en los que existe ruido. La razón es sencilla, cuando existe ruido, boosting se centra en intentar aprender experiencias que no siempre están etiquetadas con la misma clase, que son aquellas que han sido mal clasificadas en las iteraciones anteriores. Al centrarse en aprender de estas experiencias, cosa que no es posible puesto que son ruidosas, deja de aprender de otras experiencias que sí eran susceptibles de aportar nuevo conocimiento.

1.3. Métodos para combinar clasificadores base

Dado que una de las tareas fundamentales en el área de la clasificación supervisada es mejorar la eficacia de los métodos existentes, surge la idea de combinar múltiples clasificadores de la misma forma en que intentamos buscar varias opiniones en los problemas que resuelven las personas a diario. De esta forma una combinación de clasificadores es un clasificador que basa su decisión en la decisión de otros clasificadores a los cuales nos referiremos como los clasificadores individuales. Según (Rokach (2009)) en la literatura el término esquema (ensemble) es utilizado cuando los clasificadores individuales forman una familia, cuando los clasificadores no son de una misma familia entonces se suele utilizar el término de sistemas de múltiples clasificadores (multiple classifiers systems).

El objetivo con el desarrollo de combinaciones de clasificadores es que estos tengan un mejor desempeño que los clasificadores individuales, o sea, presentar mejores soluciones a los problemas que no han sido resueltos de manera aceptable por los métodos tradicionales, entendiendo por métodos tradicionales a los clasificadores que no toman sus decisiones basados en otras decisiones anteriores. Por desempeño de los clasificadores nos referimos tanto a la eficacia como a la eficiencia de los mismos. Si los clasificadores individuales que van a ser combinados no cometen errores coincidentes en los puntos del espacio de representación, entonces la eficacia se puede mejorar si los resultados de estos se combinan adecuadamente. Por otra parte si se tiene un problema complejo, la eficiencia se puede mejorar dividiendo el problema en sub-problemas más simples, los cuales serán resueltos por los clasificadores individuales, el problema complejo se resuelve después integrando las soluciones dadas por los clasificadores individuales.

Dietterich en (Dietterich (1997)) propone tres razones por la cuales debe considerarse el uso de los esquemas de clasificadores, estas razones serán retomadas por Dietterich en (Dietterich

(2000b)) y Kuncheva en (Kuncheva (2004)) y en este trabajo se expondrán a continuación.

La primera razón es estadística:

En la práctica, una vez dado el conjunto de entrenamiento, la tarea es encontrar un clasificador lo más eficaz posible. Sin embargo, puede darse el caso de que en lugar de encontrar un solo clasificador que es el más eficaz, se obtengan varios con una eficacia igual de aceptable, pues todos tienen un error estimado muy semejante o igual. Sin embargo, estos clasificadores pueden generalizar de manera diferente, es decir, cuando se clasifican objetos no presentes en la muestra disponible los clasificadores no tienen por qué trabajar de la misma manera, estos pueden cometer errores sobre elementos distintos. Una opción disponible es tomar aleatoriamente uno de estos clasificadores, entonces el clasificador que se tome puede ser la mejor de las opciones, pero también la peor. Una alternativa puede ser tomarlos todos y de alguna manera promediar los resultados de estos para intentar reducir el riesgo de una clasificación incorrecta.

La segunda razón es computacional:

En muchos casos los algoritmos de clasificación realizan una búsqueda en un conjunto de clasificadores posibles a partir del conjunto de entrenamiento en cuestión con el objetivo de encontrar un buen clasificador. En la literatura al conjunto de clasificadores posibles se le llama conjunto de las hipótesis. Realizar esta búsqueda de manera extensiva, o sea, tenerlos en cuenta a todos, puede ser muy costoso. Por ejemplo, el problema de encontrar el árbol de decisión más pequeño que es consistente con un conjunto de entrenamiento es NP-completo (Hyafil y Rivest (1976)), también lo es el problema de encontrar los pesos de la red neuronal más pequeña consistente con un conjunto de entrenamiento (Blum y Rivest (1989)). Por esta razón, algunos algoritmos de clasificación realizan la búsqueda a través de alguna heurística, y esto trae como consecuencia la posibilidad de quedar atrapado en un óptimo local. Esto implica que en muchos casos aunque la información disponible acerca del problema –conjunto de entrenamiento– permita encontrar un clasificador con una alta eficacia, por problemas de complejidad computacional no será posible acceder a este. Entonces, si se toman varios clasificadores, –cada uno pudo haber sido un óptimo local distinto, por lo que no tienen por qué cometer los mismos errores– al combinarlos en búsqueda de consenso se puede lograr una mejor eficacia.

Fig. 1. : La imagen izquierda muestra el límite de decisión diagonal y las aproximaciones realizadas por varios árboles de decisión. La imagen a la derecha muestra el resultado de una aproximación a través de una votación realizada por las aproximaciones mostradas en la imagen de la izquierda.

La tercera razón es representacional:

Como se ha visto anteriormente un algoritmo de clasificación define una familia de clasifica-

dores. Existe la posibilidad de que el clasificador óptimo para un problema dado no se encuentre dentro de la familia de clasificadores que define el algoritmo de clasificación que se está utilizando. Por ejemplo, existen algoritmos de clasificación cuyos clasificadores discriminan entre dos clases a partir de un híper-plano entre estas; pero también existen problemas donde las clases se encuentran distribuidas de una forma más compleja, es decir, que no es posible separar las clases solamente a través de híper-planos. Entonces si se está obligado a considerar clasificadores dentro de la familia anterior solamente –quizás por razones de eficiencia, o simplicidad–, ocurre que el clasificador óptimo no está presente dentro de los posibles resultados del algoritmo de clasificación, pero si combinamos distintos clasificadores de este tipo entonces es posible un acercamiento mucho mayor al clasificador óptimo. Dietterich propone otro ejemplo: algunos de los clasificadores que utilizan un árbol de decisión generan una partición del espacio de representación –se asume R^n- donde las supuestas clases se delimitan a partir de híper-planos paralelos a los ejes de coordenadas, si el borde real que discrimina a dos clases de objetos es diagonal entonces el árbol de decisión aproximará este borde de una forma escalonada. Si combinamos distintos árboles, obtendremos distintas aproximaciones y mediante una votación se puede llegar a obtener una mejor aproximación, véase las figuras tomadas del trabajo de Dietterich (Figura 1). Nótese que este argumento puede ser rebatido, pues, que una familia de clasificadores no contenga a un clasificador óptimo para la solución de un problema dado, no quiere decir que no exista otra familia de clasificadores que sí lo contenga. Sin embargo, es posible salvar el argumento de Dietterich, pues cabe la posibilidad de que obtener un clasificador de la segunda familia sea más complejo que obtener varios de la primera y combinarlos.

Es importante siempre tener en cuenta que una combinación de clasificadores es también un clasificador. En este trabajo la combinación de clasificadores será denotada por *Comb* cada combinación cuenta con l clasificadores individuales los cuales serán denotados por M_k $(1 < k < K)$.

La combinación de las salidas de los clasificadores es otro paso importante a la hora de construir un sistema multiclasificador. Puede dividirse en dos tipos: selección o fusión. La selección es la simple elección del “mejor” clasificador para una instancia determinada. La fusión, por otro lado, se basa en combinar, mediante alguna función, las salidas de los diferentes clasificadores. Esta segunda puede basarse en combinar las salidas de las clases ya etiquetadas o las salidas de valores continuos, correspondientes a la distribución de probabilidad dada por el clasificador.

1.3.1. Funciones de combinación

Si todos los clasificadores individuales tienen el mismo espacio de salida, la salida de todos ellos para un objeto en específico puede ser codificada a través de una matriz DP que tiene k columnas y l filas. A la matriz DP le llamará matriz de perfil de decisión. El valor de la matriz en la fila i y la columna j se denotará por $DP(i,j)$ y representa el valor de la componente número j de la tupla de salida del clasificador D_i para el objeto en cuestión, es decir, la fila i representa la tupla de salida del clasificador $D_i$la columna j representa las salidas de los clasificadores respecto al subconjunto K_j.

1.3.1.1. Combinación de salidas de tipo abstracto

Los algoritmos basados en votación consideran la salida de cada clasificador como un voto. Entre estos se encuentran los denominados: voto mayoritario y voto ponderado.

Votación mayoritaria:

En el caso del voto mayoritario a partir de todos los votos emitidos se trata de llegar a cierto tipo de consenso. Entre las formas de consenso más manejadas se encuentran la unanimidad, la simple mayoría y la pluralidad.

En el consenso por unanimidad se exige que todos los clasificadores estén de acuerdo respecto a una misma clase K_j es decir, todos deben tener la misma salida, en caso contrario la combinación se abstiene de clasificar. En otras palabras, el resultado de la combinación sería Kj si$\sum_{i=1}^{l} DP(i,j) = 1 = l$ abstención en otro caso.

En el consenso por simple mayoría se exige que más de la mitad de los clasificadores concuerden en una clase, o sea, el resultado de la combinación es K_j si $\sum_{i=1}^{l} DP(i,j) = l > \frac{l}{2}$ y una abstención en otro caso.

En el consenso por pluralidad se toma como resultado la clase por la que más se vota. Es decir, se tiene que K_j es el resultado final de la combinación si se verifica que $\sum_{i=1}^{l} P(i,j) = max_{j=1}^{k}\{\sum_{i=1}^{l} DP(i,j)\}$ los casos de empates pueden ser resueltos arbitrariamente o desembocar en una abstención. En todos los casos la salida de la combinación no es el subconjuntoKj ,sino una tupla con ceros en todas las posiciones excepto en la posición número j en la cual va un uno.

El voto mayoritario es el término utilizado en la literatura para referirse a los algoritmos basados en votación más simples, usualmente se define el tipo de consenso que se va a utilizar y después se utiliza solamente dicho término.

Teorema de Condorcet: Entre los resultados teóricos que respaldan al voto mayoritario se encuentra el teorema de Condorcet (De Condorcet y others (2014)), el cual fue publicado en el

año 1785 y asume que se tiene que tomar una decisión que tiene dos alternativas (+/−), una es correcta (+) y la otra es incorrecta (−) y que tenemos l individuos que tomarán la decisión independientemente, y estas serán combinadas usando el voto mayoritario para llegar a una solución final. Entonces...

Teorema 1 (de Condorcet). Si cada individuo hace su voto independientemente de los votos de los demás, y cada voto tiene una probabilidad$p > 0{,}5$ de ser correcto entonces cuando la cantidad de individuos tiende a infinito, la probabilidad de que la decisión del voto mayoritario sea correcta es una función monótona creciente y tiende a 1. En cambio, si $p < 0{,}5$, entonces la probabilidad del consenso correcto, a través del voto mayoritario, es una función monótona decreciente y tiende a 0. Si $p = 0{,}5$ entonces la probabilidad del consenso correcto se mantiene en 0,5.

Este teorema da soporte a la idea intuitiva de que si juntamos muchos individuos que tienen mayor probabilidad de tomar la decisión correcta que de equivocarse, entonces podemos esperar que un consenso entre estos tenga muchas posibilidades de ser correcto. En el contexto de la combinación de clasificadores supervisados, cada individuo representa un clasificador que puede votar por el Ki correcto o equivocarse. Sin embargo, este teorema no ofrece grandes asideros ya que solo es posible estimar la probabilidad de que un clasificador vote correctamente a partir de un conjunto de validación, nótese que puede ocurrir que el conjunto de entrenamiento disponible no sea lo suficientemente representativo como para permitir una buena generalización. Además que l tienda a infinito es imposible de lograr en la práctica. Lo que con un espíritu muy pragmático, tendría que traducirse en que la cantidad de clasificadores a combinar sea muy grande para que el teorema pueda dar algún aval al resultado. Esto conlleva un inconveniente que no podemos olvidar y es el costo computacional de tal solución.

Votación ponderada:

Si los clasificadores en la combinación no tienen la misma eficacia, resulta lógico dar un peso mayor en la votación a aquellos de los cuales se espera un mejor desempeño. Por ejemplo, a partir de la matriz de perfil de decisión es posible definir la siguiente familia de funciones como funciones discriminativas de cada clase:

$g_j = \sum_{i=1}^{l} b_i * DP(i, j)$

En la ecuación anterior el término bi es el peso correspondiente al clasificador Di, y por supuesto que la cuestión está en hallar (ajustar) dichos pesos de manera que la eficacia de la combinación sea lo mayor posible. Existen sobrados ejemplos (Kuncheva (2004)), (Zhou (2012)) de casos en los que la asignación correcta de los pesos a los clasificadores conlleva a una mejora de la eficacia del voto mayoritario tradicional y de la eficacia del mejor clasificador individual.

Enfoque probabilístico:

Sea $d = (d_1, d_2, \ldots, d_l)$ una tupla que codifica las salidas de los l clasificadores que van a ser combinados $d_j = K_i$ si el clasificador D_j asigna el objeto representado por x al subconjunto K_i .Entonces en el enfoque probabilístico se estiman las probabilidades de los subconjuntos K_i una vez conocidas las salidas de los clasificadores individuales –o sea la tupla d–, esto es $P(K_i|d)$.

Naïve Bayes: Aplicando la regla de Bayes y asumiendo que los clasificadores son condicionalmente independientes respecto a los subconjuntos Ki se obtiene la igualdad siguiente:

$P(K_i|d) = P(K_i)\prod_{j=1}^{l} P(d_j|K_i)$

En la ecuación anterior el término $P(K_i)$ representa la probabilidad a priori del subconjunto K_i de ocurrir, es decir la probabilidad de que un objeto cualquiera pertenezca al subconjunto K_i.Y los términos $P(d_j|K_i)$ codifican la probabilidad de que el clasificador D_j dé el resultado d_j cuando el objeto pertenece al subconjuntoK_i. En (Titterington et al. (1981)) existen propuestas parea realizar la estimación de $P(d_j|K_i)$ y $P(K_i)$.

BKS (Behavior Knowledge Space)

El BKS (Huang y Suen (1995))(Huang y Suen (1993)) forma parte de un grupo de métodos de combinación llamados multinomiales, dichos métodos estiman la mejor salida de la combinación para cada una de todas las posibles tuplas d. El espacio de conocimiento del comportamiento (BKS por sus siglas en inglés) no es más que un espacio l-dimensional donde cada dimensión se corresponde con la decisión de un clasificador.

1.3.1.2. Combinación de salidas de tipo rango

En (Ho et al. (1994)) Ho hace notar que las salidas de tipo rango contienen más información que las de tipo abstracto, sobre todo cuando el número de subconjuntos K_i aumenta, pues en estos casos las segundas y terceras opciones aportan información que no debe ser pasada por alto. Por otra parte, cuando las salidas de los clasificadores son del tipo de medida, puede ocurrir que la diferencia de escala, o incompatibilidades entre las medidas de los clasificadores haga muy complejo sino imposible su combinación. En la Biometría por ejemplo, es usual que el número de clases sea muy grande, además casi siempre es deseable que el clasificador dé varios candidatos. Por ejemplo, si se detecta una huella en la escena de un crimen –que por otra parte puede que no sea muy buena– es deseable buscar en una base de datos de personas –que suele ser muy grande– los mejores candidatos. Ho propone dos enfoques para la combinación de clasificadores con salida de tipo rango: la reducción del conjunto de clases disponibles y el reordenamiento de los subconjuntos K_i .En el primero, el objetivo es extraer un subconjunto de tan pequeño como sea posible y que todavía contenga el K_i que contenga el objeto en cuestión. En el segundo, como

su nombre lo indica el objetivo es dar un nuevo ordenamiento de que exprese un consenso entre los clasificadores individuales.

Métodos de reducción del conjunto de las clases:

Ho (Ho et al. (1994)) propone dos métodos para la reducción del conjunto de clases, y ambos requieren una fase de entrenamiento. El primero se denomina método de intersección de grandes vecindades, pues computa la intersección de conjuntos que contienen los K_i obtenidos de la salida de cada clasificador, y se espera que estos conjuntos tengan un tamaño grande. Esto se debe a que el conjunto es determinado a partir del rango más bajo que se le dio a una clase correcta en un conjunto de entrenamiento. Es decir, que un clasificador dé un posicionamiento lejano a la clase correcta para un objeto en el conjunto de entrenamiento, implicará que los conjuntos que este aporte al método de combinación sean grandes. Por otra parte, clasificadores eficaces sobre todos los objetos en el conjunto de entrenamiento siempre darán conjuntos pequeños.

El segundo método se denomina método de unión de vecindades pequeñas, pues se calcula la unión de conjuntos seleccionados de la salida de cada clasificador, el tamaño de cada conjunto se define a partir del conjunto de entrenamiento de la siguiente manera: por cada clasificador D_i, cada vez que este asigne el mejor rango a la clase correcta, se almacenará en una lista. Entonces el mayor de estos rangos por cada clasificador determinará el tamaño del conjunto de ese clasificador.

En (Ho et al. (1994)) se explica que como el método de intersección se basa en el caso peor, resulta beneficioso usarlo cuando los clasificadores tienen un desempeño aceptable considerando el caso peor, pues en caso contrario el tamaño de los conjuntos será demasiado grande. Sin embargo, se conoce que este no es el caso para un conjunto de clasificadores especializados solo en un conjunto de las características de los objetos, o simplemente especializados en cierta parte del espacio de representación pues estos clasificadores tienden a clasificar muy bien a unos objetos y muy mal a otros. De esta forma el método de la unión es preferible cuando los clasificadores están especializados en ciertos tipos de entrada.

Métodos de reordenamiento (re-ranking) de las clases:

Los métodos de reordenamiento de las clases intentan mejorar la posición en la que se encuentra la clase correcta. Ho propone en (Ho et al. (1994)) tres soluciones de este tipo y a continuación se exponen las dos más conocidas.

Método del rango mayor:

Cada objeto recibe l posicionamientos no necesariamente diferentes, pues se tienen l clasificadores. El método del rango mayor, en el reordenamiento resultante, coloca cada clase en la mejor de las l posiciones asignadas por cada clasificador, los empates lamentablemente deben ser

resueltos de manera arbitraria. Nótese que aunque se aplique una técnica como darle prioridad de la clase que más veces fue posicionada en la respectiva mejor posición, aún pueden seguir ocurriendo empates. Este método aprovecha la fortaleza de cada clasificador, y esto puede ser beneficioso pero también indeseable.

Método del conteo de Borda:

El método del conteo de Borda es un método de decisión en elecciones donde los votantes se expresan a través de un ordenamiento de sus opciones, fue desarrollado independientemente por varios autores pero debe su nombre al matemático francés Jean-Charles de Borda que le dio su forma definitiva en 1770. Sea $b_{i,j}$ el número de clases que se encuentran en una posición peor que la clase K_i en el resultado obtenido del clasificadorD_j al clasificar un objeto. Sea $B_i = \sum_{j=1}^{l} b_{i,j}$, es decir, B_i expresa cuán bien está ubicado K_ia través de todos los clasificadores. Entonces el ordenamiento final viene determinado por el ordenamiento de forma descendente de todos los B_j .Propuestas para enfrentar este problema pueden ser encontradas en (Black et al. (1958)).

Combinación de salidas de tipo medida:

Cuando la salida de un clasificador es del tipo medida, este devuelve una tupla al clasificar un objeto la cual contiene los grados de soportes que dicho clasificador da a la hipótesis de que el objeto pertenezca a cada una de las clases. Estos grados de soporte pueden ser interpretados de distintas maneras, por ejemplo: como estimaciones de la probabilidad a posteriori del objeto perteneciendo a cada clase, o sea $P(K_i|x)$, como la similitud con los prototipos de cada una de las clases, como grados de pertenencia difusa, etc. Es importante notar que las salidas de medida no pueden ser interpretadas de manera arbitraria, pues eso depende del clasificador en cuestión, por ejemplo, las salidas de los clasificadores de discriminante lineal y cuadrático no pueden ser interpretadas directamente como las probabilidades a posteriori de cada clase (Kuncheva (2004)), tampoco es correcto interpretar los soportes como distancias o similitudes a un prototipo de la clase sin ningún fundamento; por otra parte, a veces se plantean teorías acerca de las propiedades que cumplen las salidas de ciertos clasificadores realizando asunciones que no se verifican en la realidad.

1.4. Consideraciones finales del capítulo

En este capítulo se realizó un análisis del marco teórico relacionado con los Multiclasificadores y los métodos para combinar las salidas de los mismos, se profundizó en el estudio de los clasificadores individuales y los clasificadores incrementales haciendo énfasis en los Métodos Bagging y Boosting. La combinación de las salidas de estos multiclasificadores es actualmente

una de las tareas fundamentales en el área de la clasificación supervisada, a grandes rasgos se estudio 3 razones por las cuales es considerable el uso de esquemas de clasificadores, estas razones son desde el punto de vista estadístico, computacional y representacional describiendo después varias funciones de combinación que pueden o no mejorar el desempeño de los clasificadores referente a la eficacia y la eficiencia de los mismos. Luego de realizado este análisis se describe la evaluación de los métodos en el aprendizaje de maquina, así como las métricas que han sido usadas en estudios anteriores y la metodología a seguir referente a este campo. Se ha hecho evidente con el estudio realizado en este capitulo la necesidad de la evaluación y estudio de los métodos de combinación de las salidas de los clasificadores para distintos tipos de clasificadores. En particular en este trabajo se evaluaran distintos tipos de combinación para el multiclasificador incremental Bagging en la versión en linea existente en la herramienta Análisis Masivo Online (MOA) en la cual se profundizara en el capitulo siguiente.

CAPÍTULO 2

Herramienta MOA: Bagging y combinación de clasificadores base

2.1. Descripción de la herramienta MOA

Análisis Masivo Online (MOA) es un entorno de software para la implementación de algoritmos permitiéndonos llevar a cabo experimentos para el aprendizaje en línea con flujos de datos. MOA incluye una colección de métodos tradicionales y online, así como herramientas para la evaluación. En particular, se implementa boosting, bagging y Hoeffding tree, todos con y sin clasificadores bayesianos en las hojas. MOA está relacionado con WEKA que es otra herramienta de Minería de Datos. ambos están implementados en el lenguaje de programación java. Los principales beneficios son la potabilidad de Java, donde las aplicaciones se pueden ejecutar en cualquier plataforma con una adecuada máquina virtual.

El funcionamiento del análisis en MOA se basa en flujos de datos continuos. Un entorno de análisis que trabaje con stream de datos debe tener requisitos diferentes de la configuración batch tradicional (la soportada por WEKA). Los más significativos son los siguientes:

C:/Users/Rosita/Desktop/Plantilla cubana/fotos/Sin título.png

C:/Users/Rosita/Desktop/Plantilla cubana/fotos/ciclo.png

Figura 2.1: Ciclo de clasificación de flujo de datos

1. Procesar un ejemplo por vez y en una sola iteración.
2. Utilizar una cantidad limitada de memoria.
3. Trabajar en un periodo de tiempo limitado.
4. Estar preparado para predecir en cualquier momento.

En la figura 2.1

En (1), se pasa al clasificador el ejemplo siguiente disponible del stream (requisito 1).

En (2), el clasificador procesa el ejemplo y actualiza sus estructuras de datos. Lo hace sin rebasar los límites de memoria establecidos en el mismo (requisito 2), y lo más rápidamente posible (requisito 3).

En (3), el algoritmo está listo para aceptar el siguiente ejemplo. A petición, es capaz de predecir la clase de los ejemplos que no constan en la entrada (requisito 4).

En el aprendizaje batch tradicional, el problema de la escasez de datos es superado analizando y promediando el resultado de distintos conjuntos de datos de train y test. Por otro lado, en el aprendizaje incremental, el problema de hacer preciso un análisis continuo, plantea retos distintos. Una solución consiste en hacer 'capturas' en diferentes momentos para ver hasta qué punto mejora el modelo. El procedimiento de evaluación de un algoritmo de aprendizaje determina qué ejemplos son usados en el entrenamiento del clasificador, y cuáles son utilizados en las pruebas. El procedimiento utilizado históricamente en el aprendizaje batch, en parte ha dependido del tamaño de los datos de entrada. Con grandes sets de datos suele ser necesario reducir el número de repeticiones del algoritmo, a fin de completar los experimentos en un plazo razonable. Sin embargo, al intentar deducir qué procedimiento de evaluación utilizar sobre un entorno de aprendizaje incremental, uno de los problemas que surgen es cómo construir una buena medida de precisión en el tiempo. Surgen dos enfoques:

• Holdout (retención): Cuando el aprendizaje batch alcanza un tamaño en el que la validación cruzada consume demasiado tiempo, a menudo se retiene un conjunto de entrenamiento para medir su rendimiento. Esto es muy útil cuando la división entre los sets de entrenamiento y de

prueba han sido predefinidas, de forma que los resultados de los diferentes estudios se pueden comparar directamente.

• Interleaved Test Then Train o Prequential: Cada ejemplo se puede utilizar individualmente para probar el modelo antes de ser usado en el entrenamiento. Desde aquí, la precisión se puede actualizar de forma incremental. Este esquema tiene la ventaja de no necesitar la retención de ejemplos, haciendo el máximo uso de los datos disponibles. También garantiza un gráfico temporal de precisión liso, ya que cada ejemplo será cada vez menos significativo en la media global. Como la clasificación stream de datos es un campo relativamente nuevo, estas prácticas de evaluación no están tan bien investigadas y establecidas como lo están en la configuración batch. La mayoría de las evaluaciones experimentales utilizan menos de un millón de ejemplos de entrenamiento. En el contexto de los flujos de datos esto es decepcionante ya que, para ser verdaderamente útil, los algoritmos tienen que ser capaces de manejar flujos de ejemplos potencialmente infinitos. MOA permite evaluar adecuadamente algoritmos de clasificación de stream de datos en flujos de grandes dimensiones, del orden de decenas de millones de ejemplos siempre que sea posible, y bajo los límites de memoria explícita.

2.1.1. Creación de un nuevo clasificador en MOA.

La implementación de un nuevo clasificador en MOA es posible ya que es una plataforma de código abierto y además extensible. MOA tiene una estructura de paquetes bien organizada y facilita a otros programadores la tarea de agregar modelos y algoritmos, o modificar los ya existentes de forma ordenada.

Siguiendo la sintaxis de java, cada clasificador se encuentra ubicado en una clase separada.Cada clasificador en MOA debe contar con la siguiente serie de elementos y métodos:

Elementos Option: Representan los datos de entrada al análisis, que son introducidos por interfaz o terminal. Para protección del sistema Option handling es manejado automáticamente. Los elementos Option necesitan ser miembros públicos de la clase extendiendo a su vez de moa.options.Option

```
1   public IntOption gracePeriodOption = new IntOption
2           ("gracePeriod", "g",
3                   "The number of instance" +
4                           " to observe between model changes"
                                ,
5                   1000,0, Integer.MAX_VALUE);
```

```
6
7     public FloatOption binarySplitsOption = new FloatOption
8             ("binarySplits","b","only␣allow␣binary␣splits");
9
10    public ClassOption splitCriterionOption = new ClassOption
11            ("splitCriterion","e",
12                    "Split␣criterion␣to␣use",SplitCriterion.
                        class,
13                    "InfoGainSplitCriterion");
```

Método resetLearningImpl:

Prepara el aprendizaje restaurando las variables globales a su valor por defecto.

```
1   public void resetLearningImpl() {
2           this.ensemble = new Classifier[this.ensembleSizeOption.
              getValue()];
3           Classifier baseLearner = (Classifier)
              getPreparedClassOption(this.baseLearnerOption);
4           baseLearner.resetLearning();
5           for (int i = 0; i < this.ensemble.length; i++) {
6               this.ensemble[i] = baseLearner.copy();
7           }
8   }
```

Método trainOnInstanceImpl:

Entrena los ejemplos uno por uno. No obstante, también se puede forzar un entrenamiento conjunto mediante la declaración de un buffer de instancias (como se realiza en la clase WEKA-Classifier).

```
1   public void trainOnInstanceImpl(Instance inst) {
2           for (int i = 0; i < this.ensemble.length; i++) {
3               int k = MiscUtils.poisson(1.0, this.
                  classifierRandom);
4               if (k > 0) {
5                   Instance weightedInst = (Instance) inst.copy();
6                   weightedInst.setWeight(inst.weight() * k);
7                   this.ensemble[i].trainOnInstance(weightedInst);
```

```
8                }
9            }
10       }
```

Método getVotesForInstance:

Dicho método se encarga de clasificar una instancia, recibida como parámetro, en la configuración de ese instante de tiempo.

```
1   public double[] getVotesForInstance(Instance inst) {
2           DoubleVector combinedVote = new DoubleVector();
3           for (int i = 0; i < this.ensemble.length; i++) {
4               DoubleVector vote = new DoubleVector(this.ensemble[
                    i].getVotesForInstance(inst));
5               if (vote.sumOfValues() > 0.0) {
6                   vote.normalize();
7                   combinedVote.addValues(vote);
8               }
9           }
10          return combinedVote.getArrayRef();
11      }
```

En particular la implementación de este clasificador dentro de la plataforma MOA se ubicó en el subpaquete moa.classifiers.meta, donde la clase principal que la representa se denomina AbstractClassifier .

2.1.2. MOA a partir de WEKA

Una de las estructuras de datos clave utilizadas en MOA es la descripción de una instancia de un flujo de datos (esto vendría a ser el ejemplo de entrenamiento, en el estado del arte). Dicha estructura se toma prestada de WEKA, donde cada instancia es representada por una matriz de valores de coma flotante de doble precisión. Esto proporciona libertad para almacenar todos los tipos necesarios de valores numéricos. Los valores en coma flotante de doble precisión requieren un espacio de almacenamiento de 64 bits (8 bytes). Este detalle puede tener implicaciones en el uso de memoria.

Como se ha descrito, MOA utiliza las clases y el código de WEKA. MOA varía de WEKA en cuanto a su aplicación en streaming de datos, y su condición de clasificación sin necesidad de entrenamiento previo, gracias a las propiedades de la clasificación incremental (ver estado

del arte). WEKA se encuentra más orientado a pequeños conjuntos de datos, cuyos valores no evolucionan con el tiempo. Sin embargo, el funcionamiento y la formación de sus clases internas siguen, los mismos patrones. Por un lado, el uso de WEKA se encuentra muy ligado al de MOA.

2.2. Nuevo enfoque para el uso de métodos de combinación de clasificadores base en bagging.

La manera en que se combinan los resultados de los clasificadores individuales es una cuestión fundamental en el estudio de los ensambles (combinaciones) de clasificadores. La idea aquí es optimizar la decisión que se va a tomar a partir de las decisiones individuales. Múltiples métodos de combinación de clasificadores han sido utilizados con éxito en el aprendizaje por lotes tradicional. En este trabajo se seleccionaron tres de los métodos más conocidos para la combinación de clasificadores base en el aprendizaje a partir de flujos de datos no estacionarios. Además, se propone el uso del método de control estadístico de procesos EWMA *(Exponentialy Weighted Moving Average*) ya que permite asignar más o menos importancia a la precisión actual y al peso anterior del clasificador. A continuación se describen los métodos de combinación empleados.

2.2.1. Naıve Bayes (NB)

La regla de Bayes ha tenido múltiples aplicaciones en la teoría de las probabilidades y en el aprendizaje automático. Usando la regla de Bayes, se puede extender la idea de Naive Bayes para combinar varios clasificadores (Rokach (2005)) de la siguiente manera:

$$class\,(x) = \underset{\substack{c_j \in dom(y) \\ P\left(y,=c_j\right)>0}}{\operatorname{argmax}} \hat{P}\left(y = c_j\right) \times \prod_{k=1} \frac{\hat{P}_{M_k}\left(y = c_j|x\right)}{\hat{P}\left(y = c_j\right)} \tag{2.1}$$

donde M_k denota al clasificador k y $\hat{P}_{M_k}\left(y = c_j|x\right)$ denota la probabilidad de que y obtenga el valor c_j dada una instancia x.

2.2.2. EWMA

EWMA (*Exponentialy Weighted Moving Average*) es un método de control estadístico de procesos. La característica esencial de este método es que otorga menos peso a las observaciones

cuanto más alejadas están en el tiempo. El estadístico que se representa en el gráfico es:

$$Ewma_{M_k} \rightarrow W_{M_k} = \beta x_{M_k} + (1-\beta) W_{M_k} \tag{2.2}$$

donde x_{M_k} es una variable aleatoria , β es una constante para determinar el peso de las observaciones, $(0 < \beta < 1)$.

En este trabajo, la ecuación 2.2 se utiliza para determinar el peso de los clasificadores base. En este caso, W_{M_k} sería el peso de cada clasificador y x_{M_k} la precisión actual. Como cada clasificador utiliza un detector de cambio, se puede estimar en cualquier momento el error ε_m de cada clasificador base, por lo que la precisión sería $1-\varepsilon_m$. La constante β preestablecida representa el nivel de importancia que se les otorga al funcionamiento de los clasificadores base frente a los datos antiguos y frente a los datos actuales respectivamente. Un valor elevado de β significa que se le dará más importancia al funcionamiento histórico del clasificador que al funcionamiento frente a los datos actuales; la adaptación al cambio será un poco más lenta pero el proceso será más robusto frente a datos ruidosos.

2.2.3. Entropy Weighting (EW)

La idea de este método de combinación es asociarle a cada clasificador un peso que es inversamente proporcional a la entropía de su vector de clasificación (Rokach (2005)) :

$$class(x) = \underset{c_i \in dom(y)}{\operatorname{argmax}} \sum_{k:c_i = \underset{c_j \in dom(y)}{\operatorname{argmax}} \hat{P}_{M_k}(y=c_j|x)} Ent(M_k, x) \tag{2.3}$$

donde:

$$Ent(M_k, x) = -\sum_{c_j \in dom(y)} \hat{P}_{M_k}(y=c_j|x) \times \log\left(\hat{P}_{M_k}(y=c_j|x)\right) \tag{2.4}$$

permite calcular la entropía para el clasificador M_k y la instancia x.

2.2.4. Dempster–Shafer (DS)

La idea de usar la teoría de la evidencia de Dempster–Shafer para combinar modelos fue propuesta por (Shlien (1990)) . Este método utiliza la noción de asignación de probabilidades

básica definida para una cierta clase c_i dada la instancia x:

$$bpa\left(c_i,x\right) = 1 - \prod_k \left(1 - \hat{P}_{M_k}\left(y = c_i|x\right)\right) \tag{2.5}$$

En consecuencia, la clase seleccionada es la que maximiza el valor de la función:

$$Bel\left(c_i,x\right) = \frac{1}{A} \times \frac{bpa\left(c_i,x\right)}{1 - bpa\left(c_i,x\right)} \tag{2.6}$$

donde A es un factor de normalización definido como:

$$A = \sum_{\forall c_i \in dom(y)} \frac{bpa\left(c_i,x\right)}{1 - bpa\left(c_i,x\right)} + 1 \tag{2.7}$$

2.2.5. Algoritmo propuesto

Entrada:
instance: ejemplos de entrenamiento con etiquetas de clase $y_i \in Y = 1,\ldots,L$
K: tamaño del ensamble
Comb: método de combinación de clasificadores
Salida :
$\mathfrak{C}(x) =$ combinar los clasficadores base M_k utilizando *Comb* for all $k \in \{1,2,\ldots,K\}$

```
1 begin
2   Inicializar los clasficadores base M_k, detectores D_k for all k ∈ {1,2,...,K}
3   foreach ejemplo de entrenamiento do
4     for k ← 1 to K do
5       K ← P(λ)   // P(λ) es la distribución de Poisson con media λ
6       for j ← 1 to K do
7         Actualizar M_k and D_k con el ejemplo actual
8       if D_k estima un cambio then
9         reiniciar M_k y D_k
```

Algoritmo 1: ADCE

El algoritmo propuesto llamado ADCE utiliza la versión en línea del algoritmo bagging para entrenar los clasificadores bases y uno de los métodos de combinación de clasificadores descritos en la sección anterior para determinar los pesos de los clasificadores. ADCE manipula cambios de concepto de una manera simple y eficiente (ver algoritmo 1). Cada clasificador base

M_k $(1 < k < K)$ utiliza un detector de cambios $\mathfrak{D}_k$ para estimar la tasa de error ε_k de cada clasificador. Cuando $\mathfrak{D}_k$ estima un cambio, el clasificador M_k es remplazado por uno nuevo. Para combinar la salida de los clasificadores base se puede utilizar cualquiera de los métodos mencionados anteriormente. El algoritmo solo recibe como parámetros el detector de cambios y el número de clasificadores base.

La tasa de error de los clasificadores base se monitoriza constantemente a medida que llega cada ejemplo de entrenamiento. Por lo tanto, este seguimiento también debe realizarse con recursos computacionales controlados. En los últimos años se han propuesto varios métodos en la comunidad estadística para detectar cambios en línea (Montgomery (2009)). Sin embargo, estos suponen que los datos de entrada están regulados por una distribución de probabilidad conocida. ADCE utiliza HDDM (*Hoeffding Drift Detection Method*) (Frias-Blanco et al., 2015) como detector de cambios de concepto y estimador de error. HDDM procesa cada valor entrante con una complejidad computacional constante y proporciona garantías matemáticas para las tasas de falsos positivos y falsos negativos.

2.3. Diseño e implementación de los métodos de combinación de clasificadores base.

La Interfaz Option le permite a todas las clases añadir una opción o parámetro para el clasificador a implementar.

Para la creación del algoritmo fue necesario la implementación de ClassOption, IntOption, FloatOption, ademas de MultiChoiceOption permitiendo al usuario escojer entre los 4 métodos de combinación de clasificadores implementados para el algoritmo propuesto, facilitando el uso de la interfaz y la evaluación del algoritmo ADCE con respecto a estos métodos.

```
1 public MultiChoiceOption combinationOption =
2                 new MultiChoiceOption(
3                "leafprediction", 'c', "Leaf␣prediction␣to␣use.",
4                 new String[]{
5                "WMV", "NB", "DS", "EW"},
6                 new String[]{
7                "Weighted␣Majority␣Vote",
8                "Naive␣Bayes",
9                "Dempster-Shafer",
```

```
10              "Entropy-Weighting"}, 0);
```

Para entrenar el clasificador es utilizado el método trainOnInstanceImpl se utiliza una matriz de confucion de tres dimensiones que contiene los clasificadores con los que se realiza el entrenamiento, las clases a predecir y la predicción de un clasificador con respecto a una clase. El entrenamiento es realizado al igual que en el algoritmo en linea del método Bagging, solo se añadió la utilización de la matriz de confusión tratada anteriormente. Luego se comprueba que la clasificación sea correcta para cada uno de los clasificadores si se comprueba la existencia de algún cambio se reinicia el entrenamiento.

```
1       public void trainOnInstanceImpl(Instance inst) {
2
3           //confusio matrix
4           if (this.confusionmatrix == null) {
5               this.confusionmatrix = new int[this.
                    ensembleSizeOption.getValue()]
6                       [inst.numClasses()][inst.numClasses()];
7           }
8
9           boolean change = false;
10          for (int i = 0; i < this.ensemble.length; i++) {
11              //nuevo
12              double[] prediction = ensemble[i].
                    getVotesForInstance(inst);
13              int predictedClass = Utils.maxIndex(prediction);
14              if (this.combinationOption.getChosenIndex() == 1) {
15                  this.confusionmatrix[i][(int) inst.classValue()
                        ][predictedClass]++;
16              }
17
18              int k = MiscUtils.poisson(6,this.classifierRandom);
19              if (k > 0) {
20                  Instance weightedInst = (Instance) inst.copy();
21
22                  weightedInst.setWeight(inst.weight() * k);
23                  // main
```

```
24
25                  this.ensemble[i].trainOnInstance(weightedInst);
26              }
27              // main
28              boolean correctlyClassifies = this.ensemble[i].
                    correctlyClassifies(inst);
29              this.estimator[i].input(correctlyClassifies ? 0 :
                    1);
30              this.weight[i] = this.weight[i] * (1 - this.lambda)
                     + (1 - this.estimator[i].getEstimation()) * this
                    .lambda;
31              if (this.estimator[i].getChange() == true) {
32                  this.ensemble[i].resetLearning();
33                  this.estimator[i] = this.estimator[i] =
34                          ((ChangeDetector)
                                getPreparedClassOption(this.
                                driftDetectionMethodOption)).copy();
35                  this.weight[i] = 1.0;
36              }
37          }
38
39      }
```

para reiniciar el proceso de aprendizaje fue necesario reiniciar el clasificador ensemble obteniendo los valores desde ensembleSizeOption, se reinician los detectores de cambio preparando al clasificador para un nuevo aprendizaje.

```
1   public void resetLearningImpl() {
2           this.ensemble = new Classifier[this.ensembleSizeOption.
                getValue()];
3           this.estimator = new ChangeDetector[this.ensemble.
                length];
4           this.weight = new double[this.ensembleSizeOption.
                getValue()];
5           this.lambda = this.lambdaOption.getValue();
6           Classifier baseLearner = (Classifier)
```

```
            getPreparedClassOption(this.baseLearnerOption);
7           baseLearner.resetLearning();
8           for (int i = 0; i < this.ensembleSizeOption.getValue();
                i++) {
9               this.ensemble[i] = baseLearner.copy();
10              this.estimator[i] = ((ChangeDetector)
                    getPreparedClassOption(this.
                    driftDetectionMethodOption)).copy();
11              this.weight[i] = 1.0;
12          }
13
14          this.confusionmatrix = null;
15      }
```

En este método se realiza la predicción de la clase a la que pertenece una instancia dada, dicha predicción se realizara mediante el uso de los métodos de combinación descritos en la sección anterior.

```
1   public double[] getVotesForInstance(Instance inst) {
2           int combineMethodIndex = this.combinationOption.
                getChosenIndex();
3
4           switch (combineMethodIndex) {
5               case 0:
6                   return weightedMajority(inst);
7               case 1:
8                   return naiveBayesVotes(inst);
9               case 2:
10                  return dempsterShafer(inst);
11              case  3:
12                  return entropyWeighting(inst);
13          }
14          return null;
15      }
```

2.4. Conclusiones finales del capítulo

En este capitulo se describe la herramienta MOA y quedan descritos los pasos necesarios para la realización de un nuevo clasificador, creándose así una metodología que servirá como guía para la creación de los mismos. Ademas se propuso un nuevo enfoque para la realización de la combinación de los clasificadores, describiendo seguidamente el diseño e implementación de estos métodos para la versión existente en MOA del multiclasificador Bagging, este nuevo método queda implementado en el subpaquete moa.classifiers.meta, donde la clase principal que la representa se denomina AbstractClassifier y fue compilado con la versión 1.8 de la máquina virtual de Java.

CAPÍTULO 3

Validación de bagging con la incorporación de varias formas de combinación de salidas de clasificadores

3.1. Evaluación de los algoritmos incrementales

Un aspecto muy importante en el aprendizaje de máquina es la evaluación de los métodos y algoritmos con el objetivo de validar su funcionamiento. Al mismo tiempo, el proceso de evaluación es útil para evaluar la eficacia de los métodos, y descubrir los puntos que pueden ser mejorados o igualados para comparar las diferentes alternativas. Por las mismas razones, estas metodologías también son necesarias cuando el proceso de aprendizaje toma lugar y hay tendencia al cambio de concepto. Además, debido a las propiedades especiales de este contexto y el la novedad relativa de este campo, las estrategias tradicionales (Bifet (2010a) no son válidas y se necesitan algunas modificaciones y mejoras. Aunque algunos métodos incluyen garantías matemáticas y límites (Helmbold y Long (1994))(), ellos normalmente se refieren al funcionamiento del peor de los caso (Bifet y Gavalda (2007)), (Freund y Schapire (1997b)), (Žliobait (2009)) o incluso ellos calculan en relación al funcionamiento actual de la basen de aprendizaje (Kolter y Maloof (2005)). Ademas los cálculos de estos limites son muy restrictivos y poco realistas y es muy difícil determinar si ellos están presentes en el problema estudiado. Debido a estos inconvenientes, es difícil de sacar conclusiones significativas sobre el funcionamiento en condiciones de limites, pero las técnicas de evaluación ofrecen una aproximación diferente que puede ser aplicada a una variedad amplia de casos, independientemente de la naturaleza especifica del

problema.

3.1.1. Métricas

Para analizar el funcionamiento de algoritmos de clasificación en un contexto donde los conceptos pueden cambiar, los principales aspectos a ser evaluados son aquéllos relacionados al poder de generalización del modelo inducido y los requisitos del algoritmo para ser ejecutado (tiempo y espacio) (Bifet (2010a)).

El poder de generalización, se mide calculando la exactitud con la que se produce la predicción (Harries y Wales (1999)) (o error (Gama et al. (2004))) , aunque la precisión e invocación este consiguiendo más presencia (Klinkenberg (2004)). Incluso algunas medidas están basadas en las últimas medidas, como el área bajo la curva de la precision-recall (AUC-PR) (Dries y Rückert (2009)), el área bajo la curva de las características del receptor(AUC) (Zhao et al. (2011)), o la F-score (Angiulli y Fassetti (2010)), todas estas medidas han sido usadas para evaluar el funcionamiento. Actualmente, es común encontrar estudios experimentales que analice exactitud y precisión al ser ejecutados (Katakis et al. (2010)).

Por otro lado, el costo de actualizar el modelo y los requisitos computacionales de los métodos, medido en tiempo y espacio, es muy importante en este contexto pues el problema esencial del funcionamiento del problema es el uso limitado de tiempo y espacio. La relevancia de estos aspectos es considerar inicialmente el aprendizaje (Schlimmer y Fisher (1986a)), dónde los requisitos de memoria son moderados por el uso de un numero de elementos necesarios para definir el concepto. Del punto de vista de los requisitos de memoria, es calculada como el número de componentes que forman el modelo y describe el modelo de aprendizaje (el número de reglas en el modelo (Schlimmer y Fisher (1986a)), clasificadores básicos en el conjunto (Katakis et al. (2010)), etc.). Con respecto al tiempo de ejecución, diferentes niveles pueden ser monitoreados. Así,podemos considerarlos como espacios dónde sólo tiempo global se ofrece (entrenando la fase (Wang et al. (2003)), o uniendo el el entrenamiento con las pruebas de las fases (Katakis et al. (2010))), Ademas, podemos encontrar algunos trabajos dónde en el tiempo de ejecución solo se entrena un ejemplo a la ves (Angiulli y Fassetti (2010)).

Con el objetivo de complementar las métricas anteriores son a menudo incluidas medidas adicionales . Algunas son estudiados en otros contextos de aprendizaje de máquina. Así, aunque pudiera haber algunas variaciones debido a la presencia de cambios de concepto se pueden considerar las pruebas de resistencia al ruido (Street y Kim (2001a))(Wang et al. (2003)), la robustez en la presencia de atributos irrelevantes (Kubat y Widmer (1995b)), la sensibilidad al orden de

presentación de los ejemplos (Schlimmer y Fisher (1986b)), la complejidad del modelo inducido (Ferrer Troyano et al. (2005a)) o, normalmente, una combinación de muchos de ellos (Kubat y Widmer (1995c)), (Minku et al. (2010)), (Widmer y Kubat (1996)). Algunas otras características están relacionadas a la detección de cambios de concepto. Específicamente, podemos estudiar el funcionamiento de los métodos para te asunciones para las que se han diseñado los algoritmos. En este grupo, se pueden considerar las métricas:

1. la cantidad de ejemplos necesarios para estabilizar el proceso de aprendizaje o para recuperarse después de una fase de empeoramiento.

2. evaluar la respuesta de los métodos atendiendo a diferentes tipos de cambios (abrupto o graduales) con diferentes niveles de magnitud en el cambio (Wang et al. (2003)), (Widmer y Kubat (1993))

3.la facilidad de adaptación del modelo cuando un concepto se repite (Ramamurthy y Bhatnagar (2007))

4. el tiempo necesario para detectar un cambio, complementándolo con la medida de falsos positivos y los errores negativos, en función de evaluar la sensibilidad y especificidad del descubridor de cambio de concepto (Baena-García et al. (2006)), (Bifet (2010b)), (Gama (2010a)).

3.1.2. Metodología

En general todas las métricas expuestas anteriormente pueden ser calculadas utilizando diferentes procedimientos. hasta ahora hemos mostrado los mas extendidos, desde las primeras ideas usadas hasta los mecanismos mas avanzados propuestos recientemente.

Una de las maneras de calcular las métricas considera solo un punto de medición , al final del proceso de aprendizaje. Este enfoque es mas frecuente cuando se aprende de conjuntos de datos reales donde la información puede ser insuficiente (Gama y Castillo (2006a)). Sin embargo, como puede ser deducido, no muestra como el proceso de aprendizaje evoluciona, lo cual es un aspecto mas relevante que el comportamiento puntual (y final).Por tanto, es mucho mas informativo presentar curvas de aprendizaje que expresen el rendimiento de diferentes métricas en tiempo real(Black y Hickey (1999)) (Black y Hickey (1999)) mientras los ejemplos están siendo procesados. Un punto interesante es como las métricas individuales que forman las diferentes curvas de aprendizaje son calculadas desde una perspectiva secuencial, esto es, que configuración es usada en cada paso del aprendizaje para calcular los valores deseados(precisión, memoria usada, etc.). De hecho, existen dos enfoques holdout (Bifet (2010c)), (Gama y Castillo (2006b)) y interleaved test-then-train.

El enfoque holduot fija, en un paso preliminar, uno o mas conjuntos de prueba (holdout test) que contienen ejemplos que se usaran solo para pruebas. Es posible usar solo un conjunto de pruebas (Syed et al. (1999)) , pero no parece ser la configuración mas apropiada porque con solo una prueba no podemos saber cual es el concepto evaluado. Es preferible separa los diferentes conjuntos que serán usados en los diferentes periodos(Street y Kim (2001b)),(Tsymbal et al. (2008)),(Widmer y Kubat (1996)). Si los conceptos o periodos de cambio son desconocidos, la selección de la prueba no puede ser guiada de manera precisa, pero la granularidad del proceso sera mayor que si usamos solo un conjunto de prueba. Por otra parte si los conceptos o periodos de cambio son conocidos, la creación de los conjuntos de prueba es inmediato, y se sabrá ademas que los conjuntos de datos usados se corresponden con el concepto actual.

El enfoque prueba-entonces-entrena intercalado, también conocido como presecuencial, deriva del error predictivo secuencial(Dawid (1984)). Básicamente consiste en calcular la medida de interés (generalmente la precisión) a medida que cada ejemplo llega (paso de prueba); y entonces, el ejemplo se hace disponible al método el cuál continua con el entrenamiento (paso de entrenamiento)(Hulten et al. (2001b)). Esta metodología es basada en la suma acumulativa de los valores de una función dada, de tal manera que el error presecuencial se vuelve más pesimista a medida que se ven más ejemplos, porque no existe un mecanismo de olvido y el rendimiento actual es diluido en toda la historia. La primera variación propuesta para eliminar esta desventaja fuel el uso de ventanas deslizantes, por lo que el error presecuencial es calculado considerando solo los últimos ejemplos (Gama, Rodrigues y Sebastião (2009)), olvidando el contexto pasado y brindando una estimación que refleja mejor el estado actual. Una desventaja al usar ventanas deslizantes es el de determinar el tamaño de la ventana, por lo que otra aproximación fue propuesta para eliminar el problema de olvido: el factor de desvanecimiento. Con este método, un factor de deterioro (decay factor)(Gama, Sebastião y Rodrigues (2009)) es aplicado y la importancia de los ejemplos previos disminuye, dándole más importancia a los más recientes. Adicionalmente, el uso del factor de desvanecimiento es más eficiente ya que no requiere almacenar ningún ejemplo.

Considerando solo algunas métricas, se ha demostrado que la precisión presecuencial (o error) (Gama, Sebastião y Rodrigues (2009))calculado con mecanismo de olvido (ventana deslizante o factor de desvanecimiento), converge a las mediciones calcadas con el enfoque holdout y el tiempo necesario para detectar cambios es menor cuando se usa el factor de desvanecimiento en pruebas Page-Hinkley.

El método presecuencial considera un paso de evaluación cada vez que un ejemplo está disponible, y acumula el resultado (con o sin mecanismo de olvido), tomando en cuenta el último ejemplo. Existe una variación del método presecuencial que usa bloques de ejemplos para rea-

lizar la prueba(Brzeziński (2010)). Para conectar estas dos alternativas, podemos ver el método presecuencial como un caso particular de la variación mencionado anteriormente, en donde los bloques están limitados a un ejemplo. Esta generalización no debe ser confundida con el método holdout, donde los conjuntos de pruebas son usados en diferentes intervalos (Ferrer Troyano et al. (2005b)). Esto se debe al hecho que en este último los conjuntos de pruebas son fijados con anterioridad, y, más importante, no son usados luego en la fase de entrenamiento.

Otro punto interesante que diferencia las metodologías de evaluación en este contexto con respecto a las tradicionales es el soporte estadístico que se puede conseguir. En el tradicional aprendizaje en lote (batch learning), en donde los conjuntos de datos son finitos y los ejemplos se sumen son independientes, varias técnicas (validación cruzada, bootstraping, etc.) pueden ser usadas para obtener soporte estadístico de los resultados. Pero cuando el contexto puede cambiar y los ejemplos pueden presentar correlación con respecto al tiempo, no es posible usar las técnicas previas para obtener el mismo resultado(Gama (2010b)). Algunos diseños experimentales usan enfoques inspirados en repetición(Kubat y Widmer (1995d)),(Widmer y Kubat (1996)) y validación cruzada(Syed et al. (1999)) pero no es posible realizar ninguna prueba estadística que asegure las diferencias significativas deseadas. Por tanto nuevas estrategias deben ser desarrolladas.

3.2. Diseño experimental

En el estudio experimental descrito en esta sección se evalúan los algoritmos frente a conjuntos de datos artificiales y reales. Además, los algoritmos son evaluados sobre los diferentes tipos de cambio de concepto (abruptos y graduales).

Todos los experimentos fueron ejecutados en MOA (Bifet, Holmes, Kirkby y Pfahringer, 2010), una herramienta para evaluar los algoritmos que aprenden a partir de flujos de datos. MOA provee una gran colección de herramientas de evaluación, varios algoritmos de aprendizaje y métodos para generar flujos de datos artificiales con la posibilidad de incluir cambios de concepto.

La medida de rendimiento utilizada para evaluar los algoritmos fue la precisión de la clasificación. Esta medida se calculó en línea con el objetivo de medir cómo evoluciona el proceso de aprendizaje en el tiempo. En este artículo se utilizó para evaluar los algoritmos el enfoque *test-then-train*, también conocido como *prequential* (*predictive sequential*) que se deriva del error predictivo secuencial (Dawid, 1984). Este enfoque consiste básicamente en calcular las medidas de interés (usualmente la precisión) para cada instancia nueva (etapa de prueba) y después utilizar

el ejemplo para seguir con el entrenamiento del algoritmo (etapa de entrenamiento) (Domingos y Hulten (2000)). Por lo tanto, en cada ejemplo nuevo, el clasificador primero se probó y luego se entrenó. Durante el proceso de aprendizaje, la precisión se calculó con respecto a una ventana deslizante de tamaño 100 (Bifet, Holmes, Kirkby y Pfahringer, 2010). La precisión de los clasificadores se calculó cada 100 ejemplos procesados por medio de la fracción entre el número de los ejemplos bien clasificados en esta ventana deslizante y el tamaño de la ventana.

3.3. Características principales de los conjuntos de datos a utilizar en la experimentación

En los experimentos se consideraron conjuntos de datos artificiales (LED, SEA, RBF, WAV, AGR, STA, HYP) y conjuntos de datos reales (ver Tabla 3.1). Los conjuntos de datos artificiales permitieron evaluar los métodos frente a conceptos estables, cambios abruptos y graduales. Los conjuntos de datos artificiales fueron generados por el software MOA (Bifet, Holmes, Kirkby y Pfahringer (2010)). En los conjuntos de datos reales, no hay una afirmación sólida sobre la presencia o el tipo de cambio de concepto (Bifet, Holmes y Pfahringer (2010))(Bifet et al. (2009))(Frías-Blanco, del Campo-Ávila, Ramos-Jiménez, Carvalho, Ortiz-Díaz y Morales-Bueno (2016b))(Frías-Blanco, Verdecia-Cabrera, Ortiz-Díaz y Carvalho (2016))

En los conjuntos de datos artificiales, se generaron 10 cambios de concepto por cada generador. Los cambios se generan cada 25,000 instancias. Para evaluar el algoritmo propuesto frente a cambios graduales se realizaron varios experimentos cambiando el número de instancias en el período de transición entre conceptos consecutivos. Los valores utilizados fueron 100, 500, y 100.

3.3.1. Configuración de los algoritmos

El algoritmo propuesto ADCE utiliza la versión en línea del algoritmo bagging para entrenar los clasificadores base. Primeramente se evaluó el algoritmo propuesto en combinación con diferentes métodos para combinar clasificadores: EWMA, NB, EW, DS.

Las combinaciones de ADCE con los diferentes métodos de combinación de clasificadores se evaluaron sobre todos los conjuntos de datos descritos en la sección anterior. Los resultados obtenidos se analizaron utilizando el test de Friedman y el procedimiento de Holm para el análisis post hoc, con un valor de significación de $0,05$. La Figura 3.1 muestra los rankings de los algoritmos con respecto a la precisión de la clasificación, en la cual las combinaciones entre las

Tabla 3.1: Principales características de los conjuntos de datos utilizados en los experimentos

Conjunto de Datos	Acrónimo	Eejemplos	Nominal	Numérico	Valores perdidos	Clases
LED display	LED	1,000,000	24	0	no	10
SEA	SEA	1,000,000	0	3	no	2
Radial base functions	RBF	1,000,000	0	10	no	2
Waveform	WAV	1,000,000	0	40	no	3
Agrawal	AGR	1,000,000	3	6	no	2
Stagger	STA	1,000,000	3	0	no	2
Hyperplane	HYP	1,000,000	0	10	no	2
Usenet 1	USE1	1,500	100	0	no	2
Usenet 2	USE2	1,500	100	0	no	2
Segment	SEG	2,310	0	19	no	7
Mushroom	MUS	8,124	22	0	yes	2
Spam	SPA1	4,601	1	57	mo	2
Spam corpus 2	SPA2	9,323	500	0	no	2
Nursery	NUR	12,960	8	0	no	5
EEG Eye State	EYE	14,980	0	14	no	2
Weather	WEA	18,159	0	8	no	2
Bank marketing	BAN	41,188	9	7	no	2
Electricity	ELE	45,312	1	7	yes	2
Connect-4	CON	67,557	21	0	no	3
KDD Cup 10%	KDD	494,021	7	34	no	2
Forest Cover	COV	581,012	44	10	no	7
Poker Hand	POK	1,000,000	10	0	no	10

que no existen diferencias significativas están conectadas por una línea horizontal. En la misma se muestra que el algoritmo propuesto en combinación con NB obtiene los mejores resultados aunque sin superar significativamente a la combinación con EWMA. Por este motivo se utilizó la combinación con NB para comparar a ADCE con algunos de los algoritmos más conocidos basados en bagging. Específicamente se incluyeron OzaBag (versión en línea del algoritmo bagging) (Oza (2005)), OzaBagAdwin (Bifet et al., 2009) y LeveragingBag (Bifet, Holmes y Pfahringer, 2010). Los algoritmos fueron utilizados con la configuración por defecto en MOA. En el detector de cambio HDDM, el nivel de significación para el cambio fue de $0,001$ y para la alerta de $0,005$. La cantidad de clasificadores fue 10 en todos los algoritmos.

C:/Users/Rosita/Desktop/Plantilla cubana/fotos/rank-adce.eps

Figura 3.1: Ranking de los algoritmos. Los algoritmos que no son significativamente diferente están conectados

Algoritmo	ADCE+NB	LeveragingBag	OzaBagAdwin	OzaBag
AGR	**83,99 ±12,49**	82,98 ±13,29	80,57 ±17,28	63,38 ±15,64
HYP	**92,98 ±02,79**	85,45 ±13,87	85,41 ±15,59	65,77 ±23,49
LED	**73,53 ±02,90**	70,16 ±08,86	71,34 ±06,46	46,87 ±20,45
RBF	71,96 ±01,54	**72,00 ±01,48**	71,98 ±01,50	71,95 ±01,51
SEA	**87,76 ±01,63**	87,69 ±01,56	87,04 ±02,03	84,42 ±03,65
STA	**99,95 ±00,28**	87,10 ±19,18	87,28 ±18,98	65,78 ±20,05
WAV	**80,52 ±01,28**	80,49 ±01,28	80,48 ±01,30	80,48 ±01,30

Tabla 3.2: Resultados los algoritmos frente a cambios abruptos

Algoritmo	ADCE+NB	LeveragingBag	OzaBagAdwin	OzaBag
AGR	**83,96 ±12,38**	81,89 ±15,24	82,20 ±14,94	63,38 ±15,63
HYP	**91,92 ±03,06**	87,12 ±14,18	90,40 ±07,44	62,12 ±20,64
LED	**73,47 ±03,06**	73,24 ±03,61	73,19 ±03,72	47,95 ±19,20
RBF	71,96 ±01,54	72,00 ±01,48	**72,02 ±01,49**	72,00 ±01,50
SEA	**87,76 ±01,64**	87,47 ±01,86	87,28 ±01,94	84,43 ±03,65
STA	**99,87 ±00,66**	91,79 ±12,37	91,48 ±12,52	65,75 ±19,99
WAV	**80,53 ±01,29**	80,50 ±01,28	80,50 ±01,30	80,50 ±01,30

Tabla 3.3: Resultados de los algoritmos frente a cambios graduales. El número de instancias en el período de transición fue 100

Algoritmo	ADCE+NB	LeveragingBag	OzaBagAdwin	OzaBag
AGR	**83,76 ±12,42**	83,72 ±12,35	83,72 ±12,30	63,38 ±15,57
HYP	**92,42 ±2,57**	91,72 ±3,94	91,71 ±4,02	63,21 ±18,53
LED	73,32 ±3,46	**73,50 ±2,87**	73,44 ±3,14	46,06 ±19,36
RBF	71,91 ±1,53	**71,97 ±1,48**	71,92 ±1,50	71,91 ±1,50
SEA	**87,73 ±1,64**	87,63 ±1,61	87,01 ±1,91	84,44 ±3,62
STA	**99,56 ±2,08**	99,32 ±3,19	99,11 ±4,17	65,82 ±19,95
WAV	**80,54 ±1,29**	80,50 ±1,28	80,50 ±1,30	80,50 ±1,30

Tabla 3.4: Resultados de los algoritmos frente a cambios graduales. El número de instancias en el período de transición fue 500

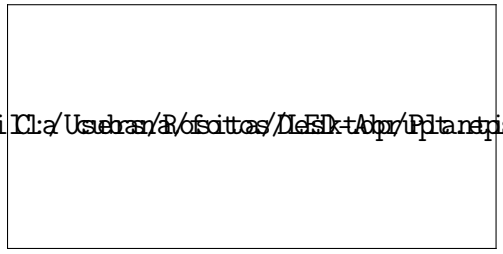
Todos los algoritmos contendientes usaron Naive Bayes como clasificador base. Naive Bayes fue elegido porque es uno de los algoritmos más exitosos para aprender de los flujos de datos (Langley et al. (1992))(Cestnik y others (1990))(Frías-Blanco, Verdecia-Cabrera, Ortiz-Díaz y Carvalho (2016)): tiene un costo computacional bajo, tiene una semántica clara y funciona bien con atributos continuos y valores perdidos.

3.3.2. Experimentos frente a cambios abruptos y graduales

En este experimento los algoritmos se ejecutaron frente a cambios abruptos y graduales utilizando los conjuntos de datos artificiales descritos en la Tabla 3.1. Para simular cambios graduales utilizamos la función sigmoide que está implementada en MOA Bifet, Holmes, Kirkby y Pfahringer (2010), aumentando la probabilidad de que las nuevas instancias pertenecen al nuevo concepto. Se generaron 10 cambios cada 25,000. Para evaluar el algoritmo frente a cambios graduales el período de transición entre conceptos consecutivos fue de 100, 500, 1000. Las Tabla 3.2 resume el rendimiento de los algoritmos sobre cambios abruptos en términos de la media y la desviación estándar para la precisión de la clasificación. Los niveles más altos de precisión están resaltados en negrita. Como se puede observar en la Tabla 3.2 el algoritmos propuesto obtiene los mejores resultados en cuanto a precisión.

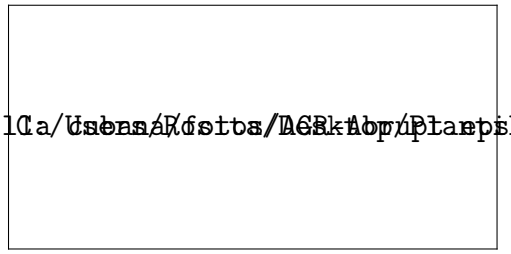
En las Tablas 3.3, 3.4 y 3.5 se muestran los resultados de los algoritmos frente a cambios de conceptos graduales. Las Tablas 3.3, 3.4 y 3.5 muestran que el algoritmo también obtiene buenos resultados frente a cambios graduales. Adicionalmente la Figura 3.2 muestra la precisión de los algoritmos sobre los conjuntos de datos utilizados. Podemos ver que la precisión del algoritmo propuesto no cae drásticamente cuando ocurre el cambio de concepto.

(a) HYP



(b) HYP

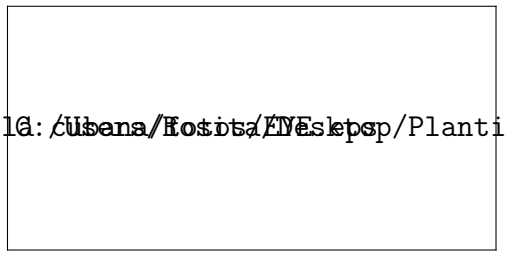
(c) SEA



(d) STA

Figura 3.2: Precisión de los algoritmos frente a cambios abruptos. Se generaron 10 cambios cada 25,000 instancias.

(a) EYE



(b) NUR

(c) USE1

(d) USE2

Figura 3.3: Precisión de los algoritmos en conjuntos de datos reales.

Algoritmo	ADCE+NB	LeveragingBag	OzaBagAdwin	OzaBag
AGR	83,46 ±12,48	83,47 ±12,38	**83,49 ±12,35**	63,36 ±15,47
HYP	**92,75 ±03,14**	92,52 ±03,48	92,35 ±03,87	63,84 ±34,24
LED	73,20 ±03,90	73,30 ±03,60	**73,22 ±03,77**	44,66 ±19,23
RBF	**72,01 ±01,55**	71,99 ±01,47	72,00 ±01,49	71,96 ±01,51
SEA	**87,67 ±01,72**	87,41 ±01,68	87,00 ±01,80	84,46 ±03,54
STA	**99,08 ±03,70**	99,01 ±04,30	98,86 ±04,89	65,86 ±19,88
WAV	**80,54 ±01,30**	80,51 ±01,29	80,51 ±01,31	80,51 ±01,31

Tabla 3.5: Resultados de los algoritmos frente a cambios graduales. El número de instancias en el período de transición fue 1000

C:/Users/Rosita/Desktop/Plantilla cubana/fotos/rank-all.eps

Figura 3.4: Ranking de los algoritmos. Los algoritmos que no son significativamente diferente están conectados. El ranking se calculó con respecto a las Tablas 3.2, 3.3, 3.4, 3.5 y 3.6.

3.3.3. Experimentos con conjuntos de datos reales

En muchos escenarios reales se ha detectado la presencia de cambios de cambios de concepto. En estos escenarios no se sabe que tipos de cambios están presentes por lo que es importante realizar experimentos con datos de distintos escenarios. Los conjuntos de datos reales seleccionados (ver Tabla 3.1) se han usado en diferentes estudios sobre aprendizaje a partir de flujos de datos no estacionarios (Frías-Blanco, del Campo-Ávila, Ramos-Jiménez, Carvalho, Ortiz-Díaz y Morales-Bueno (2016c))(Frías-Blanco, Verdecia-Cabrera, Ortiz-Díaz y Carvalho (2016)).

La Tabla 3.6 muestra el rendimiento de los algoritmos en estos conjuntos de datos reales. Los niveles más altos de precisión están resaltados en negrita. Como se puede ver en la Tabla 3.6 el algoritmo propuesto obtiene los mejores resultados. Por lo que reemplazar el clasificador en el que se detectó el cambio por uno nuevo es más eficiente que eliminar siempre el peor clasifi-

Algoritmo	ADCE+NB	LeveragingBag	OzaBagAdwin	OzaBag
ADU	**83,49 ±03,80**	**83,49 ±03,84**	83,29 ±03,89	83,29 ±03,89
BAN	88,74 ±11,74	**89,94 ±11,17**	89,79 ±11,39	89,15 ±12,58
CAR	**85,56 ±10,11**	82,56 ±10,42	80,67 ±10,78	80,56 ±11,71
CON	74,83 ±12,77	**75,07 ±13,72**	74,87 ±13,99	69,17 ±17,33
COV	**87,36 ±07,70**	83,20 ±11,92	83,07 ±11,96	60,55 ±21,76
ELE	**84,75 ±06,54**	78,84 ±11,97	78,91 ±12,13	74,26 ±14,63
EYE	**98,17 ±03,71**	90,61 ±17,88	90,91 ±18,26	47,31 ±46,41
KDD	**99,80 ±01,29**	99,62 ±03,14	99,66 ±02,88	97,88 ±11,01
MUS	98,82 ±01,75	**99,28 ±01,45**	98,65 ±02,17	98,41 ±02,29
NUR	**93,34 ±05,75**	91,19 ±08,13	90,30 ±09,27	84,11 ±14,18
OUT	**63,20 ±06,69**	60,83 ±10,53	58,33 ±11,80	55,95 ±15,50
POK	**75,69 ±09,71**	73,08 ±12,58	73,48 ±12,35	59,55 ±21,95
SEG	**78,00 ±07,21**	77,83 ±06,88	77,58 ±07,19	77,58 ±07,19
SPA1	**99,38 ±04,04**	97,64 ±05,75	98,00 ±05,19	83,55 ±14,46
SPA2	**93,05 ±05,95**	91,67 ±10,08	90,53 ±10,84	90,44 ±10,97
USE1	**79,20 ±09,24**	65,67 ±20,78	64,07 ±20,50	62,87 ±23,92
USE2	**76,67 ±08,78**	73,27 ±10,40	72,33 ±11,38	71,93 ±11,43
WEA	**72,90 ±09,81**	71,20 ±11,73	72,26 ±11,06	69,95 ±11,90

Tabla 3.6: Resultados de los algoritmos frente a conjuntos de datos reales.

cador. Al eliminar el peor clasificador puede que no se elimine el clasificador donde se detectó el cambio, por lo que el ensamble se adapta más lento a los cambios. La Figura 3.3 muestra la precisión de los algoritmos frente a los conjuntos de datos EYE, NUR, USE1 y USE2. En la Figura 3.3 se puede ver que la precisión del algoritmo propuesta no disminuye significativamente cuando ocurre un posible cambio de concepto.

Para comprobar diferencias significativas entre los algoritmos se utilizó el test de Friedman y el procedimiento de Holm para el análisis post hoc, con un valor de significación de $0,05$. El ranking se calculó con respecto a todos los conjuntos de datos (artificiales y reales). La Figura 3.4 muestra que el algoritmo propuesto es significativamente mejor que el resto de los algoritmos.

3.4. Conclusiones finales del capítulo

Se presentó un nuevo algoritmo capaz de aprender de flujos de datos no estacionarios. El nuevo algoritmo combina la simplicidad de bagging para entrenar clasificadores base con los métodos de combinación de clasificadores: EWMA, NB, DS y EW. Los resultados experimen-

tales muestran los mejores resultados utilizando NB como método de combinación. El nuevo algoritmo ADCE procesa los datos de entrada con complejidad temporal y espacial constante, y solo procesa cada ejemplo de entrenamiento una vez. Para manipular cambios de concepto, ADCE utiliza un detector de cambios en cada clasificador base para estimar su tasa de error. Cuando se estima un cambio en un clasificador base, este es remplazado por uno nuevo.

ADCE fue comparado empíricamente con varios algoritmos de ensamble de la familia bagging, utilizando Naive Bayes como clasificador base. Los experimentos incluyeron conjuntos de datos artificiales y reales. Todos los algoritmos fueron probados frente a los tipos de cambios comunes (abruptos y graduales). Los experimentos mostraron que el nuevo algoritmo es una buena opción para el aprendizaje a partir de flujos de datos con cambios de concepto.

CAPÍTULO 4

Concluciones

En este trabajo se realizó el estudio de los multiclasificadores incrementales y los métodos de combinación de los mismos, debido a que la combinación de clasificadores es una de las tareas fundamentales en el área de la clasificación supervisada se concluyo la necesidad de la experimentación y estudio del algoritmo bagging para nuevas formas de combinación de las salidas de sus clasificadores en la herramienta Análisis Masivo Online (MOA) pues esta herramienta solo cuenta con versiones del método Bagging (OzaBag, OzaBagAdwin, OzaBagASHT) que realizan la combinación utilizando el voto mayoritario. Bagging en la etapa de generación de clasificadores obtiene clasificadores independientes por la técnica de bootstrap aggregation, y se ha llegado a la conclusión de que el método del voto mayoritario no es recomendable para la combinación de dichos clasificadores.

Se describe la herramienta MOA , herramienta utilizada para la clasificación de datos tradicionales y online la cual se relaciona bidereccionalmente con la herramienta Weka utilizada para la Minería e Datos. Quedan descritos ademas los pasos necesarios para la realización de un nuevo clasificador, creándose una metodología que servirá como guia para la creación del algoritmo propuesto ADCE.

Oza y Russel muestran como el proceso de probar las replicas bootstrap para el entrenamiento puede ser simulado en un contexto de arroyo de datos, concluyendo que la probabilidad de que cualquier ejemplo individual sea escogido tiende a una distribución de Poisson(1) con este enfoque realizan versiones en linea para el método Bagging.

ADCE entrena los clasificadores al igual que la versiones en linea del método Bagging utilizando un detector de cambios en cada uno de ellos para estimar la tasa de error, remplazando un clasificador por uno nuevo cuando se estima un cambio, combinando la salida de los clasificadores obtenidos mediante los métodos Naıve Bayes (NB), EWMA, Entropy Weighting (EW)

y Dempster–Shafer (DS) Como era esperado los resultados experimentales muestran mejores resultados para el método NB como método de combinación pues este método calcula la probabilidad condicional de cada una de las clases para cada ejemplo nuevo de entrenamiento asumiendo que los atributos son independientes dada la clase. Cada ejemplo es procesado solo una vez obteniendo los datos de entrada con complejidad temporal y espacial constante. Este método fue evaluado con las versiones en linea del algoritmo Bagging, utilizando Naive Bayes como clasificador base. Los experimentos realizados incluyeron datos artificiales y reales para comprobar el funcionamiento del algoritmo frente a cambios abruptos y graduales quedando demostrado que el algoritmo propuesto ADCE es una opción para el aprendizaje a partir de flujos de datos.

REFERENCIAS

Angiulli, F. y Fassetti, F. (2010). Distance-based outlier queries in data streams: the novel task and algorithms, *Data Mining and Knowledge Discovery* **20**(2): 290–324.

Baena-García, M., del Campo-Ávila, J., Fidalgo, R., Bifet, A., Gavaldà, R. y Morales-Bueno, R. (2006). Early drift detection method.

Bauer, E. y Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants, *Machine learning* **36**(1-2): 105–139.

Bifet, A. (2010a). Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams, *Proceedings of the 2010 Conference on Adaptive Stream Mining: Pattern Learning and Mining from Evolving Data Streams*, IOS Press, Amsterdam, The Netherlands, The Netherlands, pp. 1–212.
URL: *http://dl.acm.org/citation.cfm?id=1735125.1735127*

Bifet, A. (2010b). Adaptive stream mining: Pattern learning and mining from evolving data streams, *Proceedings of the 2010 conference on adaptive stream mining: Pattern learning and mining from evolving data streams*, Ios Press, pp. 1–212.

Bifet, A. (2010c). Adaptive stream mining: Pattern learning and mining from evolving data streams, *Proceedings of the 2010 conference on adaptive stream mining: Pattern learning and mining from evolving data streams*, Ios Press, pp. 1–212.

Bifet, A. y Gavalda, R. (2007). Learning from time-changing data with adaptive windowing, *In SIAM International Conference on Data Mining*.

Bifet, A., Holmes, G., Kirkby, R. y Pfahringer, B. (2010). Moa: Massive online analysis, *Journal of Machine Learning Research* **11**(May): 1601–1604.

Bifet, A., Holmes, G. y Pfahringer, B. (2010). Leveraging bagging for evolving data streams, *Joint European conference on machine learning and knowledge discovery in databases*, Springer, pp. 135–150.

Bifet, A., Holmes, G., Pfahringer, B., Kirkby, R. y Gavaldà, R. (2009). New ensemble methods for evolving data streams, *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 139–148.

Black, D., Newing, R. A., McLean, I., McMillan, A. y Monroe, B. L. (1958). The theory of committees and elections.

Black, M. y Hickey, R. J. (1999). Maintaining the performance of a learned classifier under concept drift, *Intelligent Data Analysis* **3**(6): 453–474.

Blum, A. y Rivest, R. L. (1989). Training a 3-node neural network is NP-complete, *Advances in neural information processing systems*, pp. 494–501.

Breiman, L. (1996). Bagging predictors, *Machine learning* **24**(2): 123–140.
URL: *http://link.springer.com/article/10.1007/BF00058655*

Brzeziński, D. (2010). *Mining data streams with concept drift*, PhD Thesis, PhD thesis, MS thesis, Dept. of Computing Science and Management, Poznan University of Technology, Poznan Google Scholar.

Cestnik, B. y others (1990). Estimating probabilities: a crucial task in machine learning., *ECAI*, Vol. 90, pp. 147–149.

Dawid, A. P. (1984). Present position and potential developments: Some personal views: Statistical theory: The prequential approach, *Journal of the Royal Statistical Society. Series A (General)* pp. 278–292.

De Condorcet, N. y others (2014). *Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix*, Cambridge University Press.

Del Campo Ávila, J. (2007). Nuevos enfoques en aprendizaje incremental.

Dietterich, T. G. (1997). Machine-learning research, *AI magazine* **18**(4): 97.

Dietterich, T. G. (2000a). Ensemble Methods in Machine Learning, *SpringerLink*, Springer, Berlin, Heidelberg, pp. 1–15.
URL: *https://link.springer.com/chapter/10.1007/3-540-45014-9*$_1$

Dietterich, T. G. (2000b). Ensemble methods in machine learning, *International workshop on multiple classifier systems*, Springer, pp. 1–15.

Dietterich, T. G. (2000c). An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization, *Machine learning* **40**(2): 139–157.

Dietterich, T. G. y Bakiri, G. (1994). Solving multiclass learning problems via error-correcting output codes, *Journal of artificial intelligence research* **2**: 263–286.

Domingos, P. y Hulten, G. (2000). Mining high-speed data streams, *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 71–80.

Dries, A. y Rückert, U. (2009). Adaptive concept drift detection, *Statistical Analysis and Data Mining: The ASA Data Science Journal* **2**(5-6): 311–327.

Estruch, V., Ferri, C., Hernández-Orallo, J. y Ramírez-Quintana, M. J. (2003). Beam search extraction and forgetting strategies on shared ensembles, *International Workshop on Multiple Classifier Systems*, Springer, pp. 206–216.

Ferrer Troyano, F. J., Aguilar Ruiz, J. S. y Riquelme Santos, J. C. (2005a). Incremental rule learning and border examples selection from numerical data streams, *Journal of Universal Computer Science* **11**(8): 1426–1439.

Ferrer Troyano, F. J., Aguilar Ruiz, J. S. y Riquelme Santos, J. C. (2005b). Incremental rule learning and border examples selection from numerical data streams, *Journal of Universal Computer Science* **11**(8): 1426–1439.

Freund, Y. y Schapire, R. E. (1997a). A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of computer and system sciences* **55**(1): 119–139.

Freund, Y. y Schapire, R. E. (1997b). A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of computer and system sciences* **55**(1): 119–139.

Freund, Y., Schapire, R. E. y others (1996). Experiments with a new boosting algorithm, *ICML*, Vol. 96, pp. 148–156.
URL: *http://www.public.asu.edu/ jye02/CLASSES/Fall-2005/PAPERS/boosting-icml.pdf*

Frias-Blanco, I., Campo-Avila, J. d., Ramos-Jimenez, G., Morales-Bueno, R., Ortiz-Diaz, A. y Caballero-Mota, Y. (2015). Online and Non-Parametric Drift Detection Methods Based on Hoeffding Bounds, *IEEE Transactions on Knowledge and Data Engineering* **27**(3): 810–823.
URL: *http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6871418*

Frías-Blanco, I., del Campo-Ávila, J., Ramos-Jiménez, G., Carvalho, A. C., Ortiz-Díaz, A. y Morales-Bueno, R. (2016a). Online adaptive decision trees based on concentration inequalities, *Knowledge-Based Systems* **104**: 179–194.

Frías-Blanco, I., del Campo-Ávila, J., Ramos-Jiménez, G., Carvalho, A. C., Ortiz-Díaz, A. y Morales-Bueno, R. (2016b). Online adaptive decision trees based on concentration inequalities, *Knowledge-Based Systems* **104**: 179–194.

Frías-Blanco, I., del Campo-Ávila, J., Ramos-Jiménez, G., Carvalho, A. C., Ortiz-Díaz, A. y Morales-Bueno, R. (2016c). Online adaptive decision trees based on concentration inequalities, *Knowledge-Based Systems* **104**: 179–194.

Frías-Blanco, I., Verdecia-Cabrera, A., Ortiz-Díaz, A. y Carvalho, A. (2016). Fast adaptive stacking of ensembles, *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, ACM, pp. 929–934.

Gama, J. (2010a). *Knowledge discovery from data streams*, CRC Press.

Gama, J. (2010b). *Knowledge discovery from data streams*, CRC Press.

Gama, J. y Brazdil, P. (2000). Cascade Generalization, *Machine Learning* **41**(3): 315–343.
URL: *http://dx.doi.org/10.1023/A:1007652114878*

Gama, J. y Castillo, G. (2006a). Learning with local drift detection, *International Conference on Advanced Data Mining and Applications*, Springer, pp. 42–55.

Gama, J. y Castillo, G. (2006b). Learning with local drift detection, *International Conference on Advanced Data Mining and Applications*, Springer, pp. 42–55.

Gama, J. M. P. d. (1999). Combining classification algorithms.

Gama, J., Medas, P., Castillo, G. y Rodrigues, P. (2004). Learning with drift detection, *Brazilian symposium on artificial intelligence*, Springer, pp. 286–295.

Gama, J., Rodrigues, P. P. y Sebastião, R. (2009). Evaluating algorithms that learn from data streams, *Proceedings of the 2009 ACM symposium on Applied Computing*, ACM, pp. 1496–1500.

Gama, J., Sebastião, R. y Rodrigues, P. P. (2009). Issues in evaluation of stream learning algorithms, *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 329–338.

Hagerup, T. y Rüb, C. (1990). A guided tour of Chernoff bounds, *Information processing letters* **33**(6): 305–308.

Harries, M. y Wales, N. S. (1999). Splice-2 comparative evaluation: Electricity pricing.

Helmbold, D. P. y Long, P. M. (1994). Tracking Drifting Concepts By Minimizing Disagreements, *Machine Learning* **14**(1): 27–45.
URL: *http://dx.doi.org/10.1007/BF00993161*

Hernández Orallo, J., FERRI RAMIREZ, C. y RAMIREZ QUINTANA, M. J. (2004). *Introducción a la Minería de Datos*, Pearson Prentice Hall,.

Ho, T. K. (1998). C4. 5 decision forests, *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, Vol. 1, IEEE, pp. 545–549.

Ho, T. K., Hull, J. J. y Srihari, S. N. (1994). Decision combination in multiple classifier systems, *IEEE transactions on pattern analysis and machine intelligence* **16**(1): 66–75.

Huang, Y. S. y Suen, C. Y. (1993). The behavior-knowledge space method for combination of multiple classifiers, *IEEE computer society conference on computer vision and pattern recognition*, Institute of Electrical Engineers Inc (IEEE), pp. 347–347.

Huang, Y. S. y Suen, C. Y. (1995). A method of combining multiple experts for the recognition of unconstrained handwritten numerals, *IEEE transactions on pattern analysis and machine intelligence* **17**(1): 90–94.

Hulten, G., Spencer, L. y Domingos, P. (2001a). Mining Time-changing Data Streams, *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and*

Data Mining, KDD '01, ACM, New York, NY, USA, pp. 97–106.
URL: *http://doi.acm.org/10.1145/502512.502529*

Hulten, G., Spencer, L. y Domingos, P. (2001b). Mining time-changing data streams, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 97–106.

Hyafil, L. y Rivest, R. L. (1976). Constructing optimal binary decision trees is NP-complete, *Information processing letters* **5**(1): 15–17.

Jacobs, R. A. (1995). Methods for combining experts' probability assessments, *Neural computation* **7**(5): 867–888.

Katakis, I., Tsoumakas, G. y Vlahavas, I. (2010). Tracking recurring contexts using ensemble classifiers: an application to email filtering, *Knowledge and Information Systems* **22**(3): 371–391.

Kim, H.-C., Pang, S., Je, H.-M., Kim, D. y Bang, S. Y. (2003). Constructing support vector machine ensemble, *Pattern recognition* **36**(12): 2757–2767.

Klinkenberg, R. (2004). Learning drifting concepts: Example selection vs. example weighting, *Intelligent data analysis* **8**(3): 281–300.

Kolter, J. Z. y Maloof, M. A. (2005). Using additive expert ensembles to cope with concept drift, *Proceedings of the 22nd international conference on Machine learning*, ACM, pp. 449–456.
URL: *http://dl.acm.org/citation.cfm?id=1102408*

Kubat, M. y Widmer, G. (1995a). Adapting to drift in continuous domains, *European Conference on Machine Learning*, Springer, pp. 307–310.

Kubat, M. y Widmer, G. (1995b). Adapting to drift in continuous domains, *European Conference on Machine Learning*, Springer, pp. 307–310.

Kubat, M. y Widmer, G. (1995c). Adapting to drift in continuous domains, *European Conference on Machine Learning*, Springer, pp. 307–310.

Kubat, M. y Widmer, G. (1995d). Adapting to drift in continuous domains, *European Conference on Machine Learning*, Springer, pp. 307–310.

Kuncheva, L. I. (2004). *Combining pattern classifiers: methods and algorithms*, John Wiley & Sons.

Langley, P., Iba, W., Thompson, K. y others (1992). An analysis of Bayesian classifiers, *Aaai*, Vol. 90, pp. 223–228.

Minku, L. L., White, A. P. y Yao, X. (2010). The impact of diversity on online ensemble learning in the presence of concept drift, *IEEE Transactions on knowledge and Data Engineering* **22**(5): 730–742.

Montgomery, D. C. (2009). *Introduction to statistical quality control*, John Wiley & Sons (New York).

Oza, N. C. (2005). Online bagging and boosting, *Systems, man and cybernetics, 2005 IEEE international conference on*, Vol. 3, Ieee, pp. 2340–2345.

Quinlan, J. R. y others (1996). Bagging, boosting, and C4. 5, *AAAI/IAAI, Vol. 1*, pp. 725–730.

Ramamurthy, S. y Bhatnagar, R. (2007). Tracking recurrent concept drift in streaming data using ensemble classifiers, *Machine Learning and Applications, 2007. ICMLA 2007. Sixth International Conference on*, IEEE, pp. 404–409.

Rokach, L. (2005). Ensemble methods for classifiers, *Data mining and knowledge discovery handbook*, Springer, pp. 957–980.

Rokach, L. (2009). Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography, *Computational Statistics & Data Analysis* **53**(12): 4046–4072.

Schlimmer, J. C. y Fisher, D. (1986a). A case study of incremental concept induction, *AAAI*, Vol. 86, pp. 496–501.

Schlimmer, J. C. y Fisher, D. (1986b). A case study of incremental concept induction, *AAAI*, Vol. 86, pp. 496–501.

Shlien, S. (1990). Multiple binary decision tree classifiers, *Pattern Recognition* **23**(7): 757–763.

Street, W. N. y Kim, Y. (2001a). A streaming ensemble algorithm (SEA) for large-scale classification, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 377–382.

Street, W. N. y Kim, Y. (2001b). A streaming ensemble algorithm (SEA) for large-scale classification, *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 377–382.

Syed, N. A., Liu, H. y Sung, K. K. (1999). Handling concept drifts in incremental learning with support vector machines, *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 317–321.

Titterington, D., Murray, G., Murray, L., Spiegelhalter, D., Skene, A., Habbema, J. y Gelpke, G. (1981). Comparison of discrimination techniques applied to a complex data set of head injured patients, *Journal of the Royal Statistical Society. Series A (General)* pp. 145–175.

Tsymbal, A., Pechenizkiy, M., Cunningham, P. y Puuronen, S. (2008). Dynamic integration of classifiers for handling concept drift, *Information fusion* **9**(1): 56–68.

Wang, H., Fan, W., Yu, P. S. y Han, J. (2003). Mining concept-drifting data streams using ensemble classifiers, *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, AcM, pp. 226–235.

Widmer, G. y Kubat, M. (1993). Effective learning in dynamic environments by explicit context tracking, *European Conference on Machine Learning*, Springer, pp. 227–243.

Widmer, G. y Kubat, M. (1996). Learning in the presence of concept drift and hidden contexts, *Machine learning* **23**(1): 69–101.

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

Zhao, P., Hoi, S. C., Jin, R. y YANG, T. (2011). Online AUC maximization.

Zheng, Z. (1998). Naive Bayesian classifier committees, *European Conference on Machine Learning*, Springer, pp. 196–207.

Zhou, Z.-H. (2012). *Ensemble methods: foundations and algorithms*, CRC press.

Žliobaitė, I. (2009). Combining time and space similarity for small size learning under concept drift, *International Symposium on Methodologies for Intelligent Systems*, Springer, pp. 412–421.