Universidad de Huelva

Departamento de Tecnologías de la Información

Mejorando las técnicas de verificación de wrappers web mediante técnicas bioinspiradas y de clasificación

Memoria para optar al grado de doctor
presentada por:

Iñaki Josep Fernández de Viana y González

Fecha de lectura: 22 de enero de 2016

Bajo la dirección de los doctores:

Pedro José Abad Herrera

José Luis Álvarez Macías

José Luis Arjona Fernández

Huelva, 2016

MEJORANDO LAS TÉCNICAS DE VERIFICACIÓN DE WRAPPERS WEB MEDIANTE TÉCNICAS BIOINSPIRADAS Y DE CLASIFICACIÓN

IÑAKI FERNÁNDEZ DE VIANA Y GONZÁLEZ

UNIVERSIDAD DE HUELVA, ESPAÑA

TESIS DOCTORAL
DIRIGIDA POR DR. PEDRO J. ABAD, DR. JOSÉ L. ÁLVAREZ Y DR. JOSÉ L. ARJONA

PROGRAMA DE DOCTORADO: TECNOLOGÍAS INFORMÁTICAS AVANZADAS

OCTUBRE, 2015

Publicado por primera vez en octubre de 2015 por
The Distributed Group Onuba
Escuela Técnica Superior de Ingeniería
Campus Universitario de la Rábida
La Rábida (Huelva), 21071. España

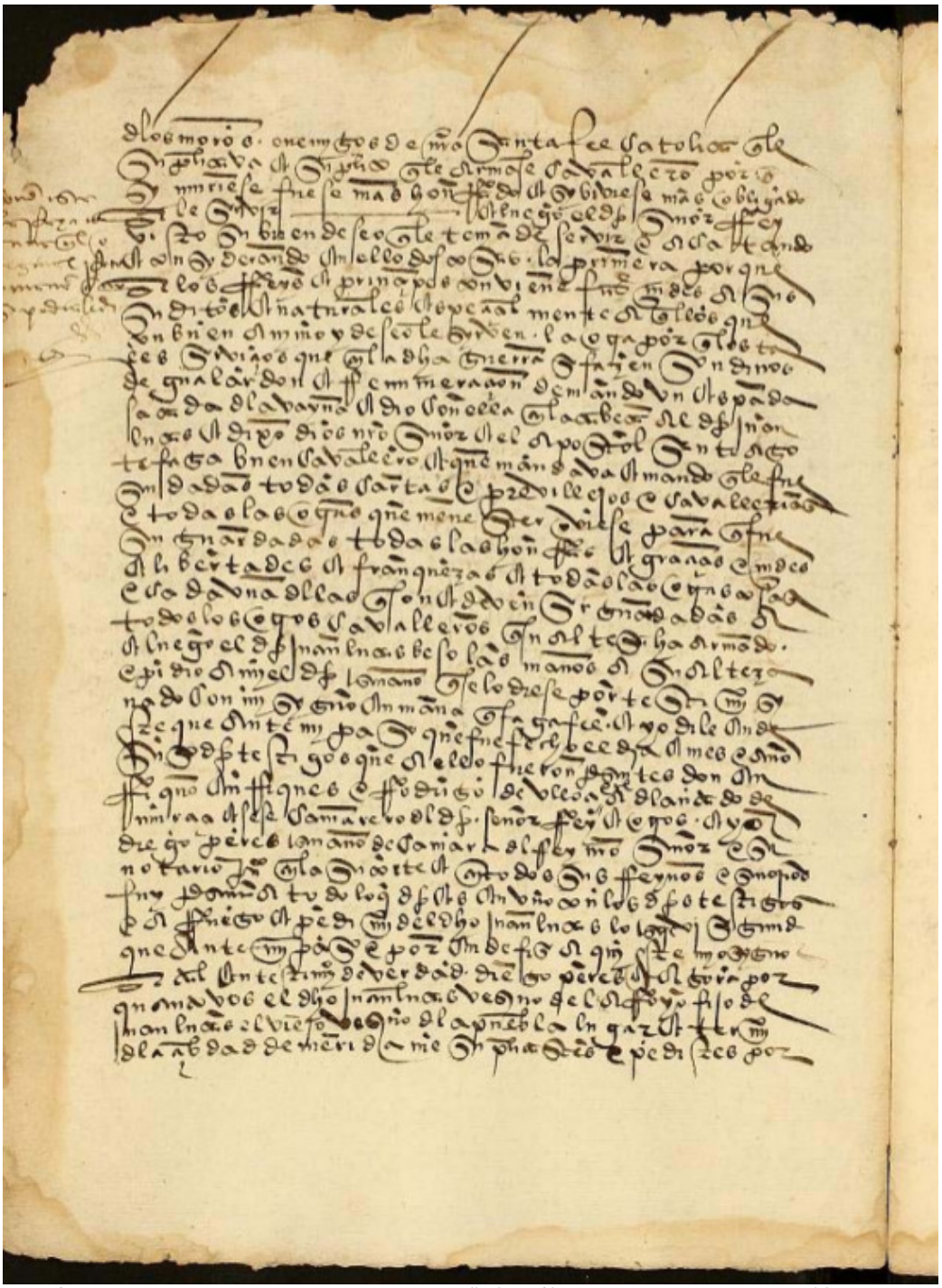
Financiación: Financiado por la European Commission (FEDER) y por proyectos I+D+I Españoles y Andaluces (TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, y TIN2010-09988-E).

Universidad de Huelva, España

El tribunal encargado de la evaluación de la tesis doctoral presentada por D. Iñaki Fernández de Viana y González en cumplimiento de los requisitos legales para la obtención del título de Doctor en Ingeniería del Software, RECOMIENDA _____ de esta tesis doctoral y otorgar al autor la calificación de _____.

_____

Y para que conste donde sea necesario, lo firmamos en _____, _____.



La verificación vista por un escribano que detalla las falsedades aportadas por el litigante para obtener los privilegios de caballería

Non permitas que aquí morra,
airiños da miña terra,
que aínda penso que de mort[o]
hei de suspirar por ela.

A mi padre

Índice general

Índice de figuras

Índice de tablas

Índice de programas

Agradecimientos

La gratitud es como aquel licor de Oriente que sólo se conserva en jarros de oro: perfuma las almas grandes y se agria en las pequeñas.

Jules Sandeau (1811-1883), Novelista francés

En especial a mis directores de Tesis Don Pedro J. Abad Herrera, Don José Luis Alvarez Macías y Don José Luis Arjona Fernández quienes junto a Don Rafael Corchuelo Gil dieron origen a esta investigación en el campo de la verificación de información.

Muchas gracias a Francisco Herrera Triguero y Óscar Cordón García que me dieron la oportunidad de formar parte del grupo de investigación Soft Computing and Intelligent Information Systems donde continué la labor de investigación que comenzó en el proyecto fin de carrera del cual fueron mis directores.

A todos los miembros del The Distributed Group, con quienes hemos compartido mucho momento de trabajo y ocio.

A los amigos de la Universidad de Castilla-La Mancha, la Universidade do Porto y el Instituto Superior de Engenharia do Porto con lo que descubrí el mundo de la calidad de los datos.

A mi familia y amigos, para los que están, han dejado de estar y lo que acaban de llegar. Durante todo este periodo de investigación me habéis sabido ayudar y aconsejar.

A todos mis compañeros de promoción, en especial la legión extrajera que llegó en los últimos cursos. Gracias a vosotros los malos momentos se hacen menos lentos.

En fin, a todas las personas que han estado conmigo desde antes de este recorrido y también a quienes conocí durante el mismo.

Abstract

Las palabras están ahí para explicar el significado de las cosas, de manera que el que las escucha, entienda dicho significado.

Aldous Huxley (1894-1963), Novelista, ensayista y poeta inglés

Many Enterprise Applications require wrappers to deal with information from the deep web. Wrappers are automated systems that allow you to navigate, extract, reveal structures and verify information from the web. One of its elements, the information extractor, is formed by extraction rules series that are usually based on HTML tags. Therefore, if you change sources, the wrapper, in some cases, may return unwanted information by the company and cause, at the best, delays in their decision-making process. Some wrappers verification systems have been developed to automatically detect when a wrapper is taking out incorrect data. These systems have a number of shortcomings whose origin lies in assuming that the data to verify follow a series of pre statistics. This dissertation analyzes these systems, a framework is designed to develop verifiers and the verification problem is approached from two different points of view. Initially, we place it within the branch of computational optimization and solve it applying bio-inspired metaheuristic as it is found in ant colonies, specifically we will apply the BWAS algorithm. Subsequently we will formulate and solve as if it were a unsupervised classification problem. The result of this second approach is MAVE, a multilevel verifier whose main base are the unique class classifiers.

Resumen

Las palabras están ahí para explicar el significado de las cosas, de manera que el que las escucha, entienda dicho significado.

Aldous Huxley (1894-1963), Novelista, ensayista y poeta inglés

Muchas Aplicaciones Empresariales necesitan de los *wrappers* para poder tratar con información proveniente de la web profunda. Los *wrappers* son sistemas automáticos que permiten navegar, extraer, estructurar y verificar información relevante proveniente de la web. Uno de sus elementos, el extractor de información, está formado por una serie de reglas de extracción que suelen estar basadas en etiquetas HTML. Por tanto, si las fuentes cambian, el *wrapper*, en algunos casos, puede devolver información no deseada por la empresa y provocar, en el mejor de los casos, retrasos en sus tomas de decisión. Diversos sistemas de verificación de *wrappers* se han desarrollado con el objetivo de detectar automáticamente cuándo un *wrapper* está extrayendo datos incorrectos. Estos sistemas presentan una serie de carencias que hace que su uso práctico sea limitado. En esta Tesis se analizan estos sistemas, se diseña un marco de trabajo para desarrollar verificadores y se aborda el problema de la verificación desde dos puntos de vista distintos. Inicialmente lo ubicaremos dentro de la rama de la optimización computacional y lo resolveremos aplicando metaheúristicas bioinspiradas como es la basada en colonias en hormigas, en concreto aplicaremos el algoritmo BWAS; con posterioridad, lo formularemos y resolveremos como si de un problema de clasificación no supervisada se tratara. Fruto de este segundo enfoque surge MAVE, un verificador multinivel cuya base principal son los clasificadores de una única clase.

Parte I

Prefacio

Capítulo 1

Introducción

Imagínese a un hombre sentado en el sofá favorito de su casa. Debajo tiene una bomba a punto de estallar. El lo ignora, pero el público lo sabe. Esto es el suspense.

Alfred Hitchcock, Director de cine inglés (1899-1980)

El objetivo de la presente Tesis es presentar los resultados de la investigación llevada a cabo en el campo de la verificación automática de la información. Ésta ha consistido, de forma resumida, en proponer un marco de trabajo para crear verificadores y tratar la verificación automática como un problema, primero de optimización computacional y luego de clasificación no supervisada. Este capítulo está organizado de la siguiente manera: la Sección §1.1, introduce el contexto de trabajo dentro del cual se ha realizado esta investigación; la Sección §1.2 presenta la hipótesis de la que se ha partido para realizar esta disertación; la Sección §1.3 resume las principales contribuciones propuestas dentro del campo de la verificación; la Sección §1.4 enumera las colaboraciones recibidas a lo largo de este trabajo; por último, se describe la estructura de esta disertación en la Sección §1.5.

1.1. Contexto de la investigación

La integración de los sistemas de información (IS, *Information Services*) es un grave problema para muchas empresas. Éstas disponen de una serie de aplicaciones heterogéneas, autónomas e independientes que no han sido diseñadas para coordinarse y colaborar entre ellas. Estas deficiencias y la complejidad intrínseca de los IS, puede causar graves problemas cuando la empresa define sus arquitecturas, estructuras y protocolos empresariales [157]. En consecuencia, la integración de este tipo de sistemas es una labor complicada para la que se han propuesto distintas soluciones [99, 105, 115].

La Integración de Aplicaciones Empresariales (EAI, *Enterprise Application Integration*) [71] es una nueva generación de software de integración que trata de una forma más eficiente la necesidad de integrar sistemas tanto dentro de una organización como entre organizaciones. A las técnicas tradicionales ya usadas [107], se les añade un conjunto de nuevas técnicas [70] con el objetivo de facilitar la integración entre sistemas.

La necesidad de aplicar estas técnicas la encontramos en el coste que supone la integración de aplicaciones y de información. Según un informe de IBM, por cada dolar gastado en desarrollar una aplicación, el costo de mantenerla es de 5 a 20 veces mayor [155]. Además, dentro del presupuesto invertido en tecnologías de la información, el 40 % se destina a la integración de información [7]. Por tanto, las necesidades de desarrollar soluciones de ingeniería que afronten el problema de la integración de información queda así justificada.

En esta Tesis, nos centraremos en solucionar el problema de la integración cuando las empresas están particularmente interesadas en alimentar sus IS con información proveniente de la web profunda (*Deep Web*) -todo el contenido de la Web que no forma parte de su "superficie", entendiendo por superficie todo aquello que es visible por los motores de búsqueda- [110]. En estos casos, la solución pasa por usar *wrappers* [111].

En el mundo de la integración de información, un *wrapper* es una pieza de software que recibe como entrada una consulta, localiza los formularios de búsqueda adecuados, los rellena, navega por las páginas resultantes con el objetivo de localizar aquello que resulte relevante para responder a la consulta formulada, extrae los atributos de interés de dichas páginas y devuelve estos atributos como un conjunto de datos. Estos conjuntos pueden estar formados por un conjunto de atributos o una jerarquía de ellos [50].

El uso de los *wrappers* es una práctica común [68, 104, 120, 138, 149, 160] en áreas como la inteligencia competitiva, comparativa de productos, subastas o integración negocio a negocio (B2B, *Business-to-business*). Esta, todavía mayor, dependencia de los datos albergados en la Web, han motivado a muchos autores a proponer distintos sistemas que tienen como objetivo reducir los costes de mantenimiento de los *wrappers* [104].

Este proceso de mantenimiento [105] se divide en dos fases: en la primera debemos ser capaces de detectar automáticamente si el *wrapper* no está extrayendo los atributos que debía, mientras que la segunda consiste en reconstruir el *wrapper* automáticamente para que éste vuelva a extraer los atributos para los que fue diseñado. En esta Tesis nos centraremos en estudiar y proponer nuevas técnicas para la primera parte, la verificación de la información.

A la hora de extraer atributos, los *wrappers* emplean reglas de extracción que se inducen a partir de una serie de conjuntos de entrenamiento, es decir, elementos que representan el dominio del discurso. Cuando dichas reglas se basan en etiquetas HTML, los *wrappers* sólo podrán extraer información de interés de las fuentes que fueron usadas para obtener dichas reglas. Así pues, si dicha web sufre modificaciones en los formularios de búsqueda, en el tipo de navegación o en la forma en la que se representa la información, el *wrapper* puede devolver información que no es de nuestro interés, esto es, información para la cual no fue diseñado. Pero no sólo modificaciones en las web pueden provocar este mal funcionamiento, el uso de reglas de extracción imprecisas o de conjuntos de entrenamiento mal etiquetados también suelen tener implicaciones negativas sobre el *wrapper*.

A menos que la información extraída por un *wrapper* sea verificada automáticamente, esta información, que puede ser errónea, podría alimentar los IS de las empresas, lo que implicaría, en el mejor de los casos, retrasos en la toma de decisiones.

En la literatura encontramos distintos sistemas de verificación [98, 99, 105, 115, 150] que presentan diversas carencias debido a que, entre otras cosas, asumen que los datos usados para construir los verificadores son homogéneos y que éstos se pueden mapear en un espacio n-dimensional en el que cada una de estas dimensiones es independiente al resto.

1.2. Investigación relacionada

En esta sección, exponemos la motivación que nos ha llevado a realizar este trabajo de investigación en el contexto de la verificación de la información y presentamos la tesis que justificaremos a lo largo de esta disertación.

1.2.1. Hipótesis

Hoy en día existe un creciente interés, por parte de las empresas, en usar la información proveniente de la Web dentro de sus IS [7]. Si bien los servicios web nos permiten acceder a esta información estructurada, no es menos cierto que su uso no está generalizado en la Web y, en otros casos, las implantaciones que se hacen de estos servicios no permiten acceder a la información de nuestro interés [2]. Para estos casos se usan los *wrappers*, cuya función es proporcionar información estructurada partiendo de fuentes de información semi-estructuradas [51]. Los continuos cambios en estas fuentes de información hacen necesario que la información verificada por los *wrappers* sea continuamente comprobada antes de poder ser usada dentro del IS [98].

Así pues, nuestra hipótesis de partida es la siguiente:

> *Las empresas están especialmente interesadas en usar wrappers para alimentar sus IS con información proveniente de la Web. Esta información es cada vez mayor y está sujeta a diversos tipos de perturbaciones, lo que hace necesario que sea verificada de forma automática antes de que ésta forme parte de sus IS.*

1.2.2. Tesis

En el contexto de la verificación de información, existen una serie de técnicas que se han venido usando en los últimos años [98, 99, 105, 115, 150]. Estas técnicas han demostrado ser soluciones válidas a la hora de verificar la información y, por tanto, reducir los costes de integración.

Desafortunadamente, estos sistemas presentan una serie de problemas que repercuten negativamente en su rendimiento y aplicabilidad en determinados contextos. Como consecuencia de esto, podemos formular la siguiente tesis:

> *En el contexto de la verificación de información es posible obtener nuevos sistemas de verificación, que mejoren el rendimiento de las técnicas actuales, si logramos solucionar las deficiencias que éstas presentan.*

1.3. Resumen de contribuciones

Para demostrar nuestra tesis, hemos desarrollado dos sistemas de verificación distintos que son la base de las contribuciones expuestas en esta disertación. Las principales contribuciones son:

- **Primera**: Proponemos el primer marco de trabajo para construir verificadores. Gracias a él podemos clarificarlos, compararlos, detectar sus deficiencias y aportar mejoras. Su diseño se basa en un profundo estado del arte de las distintas propuestas realizadas hasta la fecha.

  En cuanto a esta contribución, tenemos las siguientes publicaciones aceptadas: [47–49]. Reseñar que el trabajo presentado en las Jornadas de Ingeniería del software y Base de Datos del año 2009 fue seleccionado entre los mejores artículos.

- **Segunda**: Hemos sido innovadores a la hora de abordar, por primera vez, el problema de verificar la información extraída por un *wrapper* dentro del campo de la optimización computacional. Este nuevo enfoque nos permite aplicar un gran abanico de heurísticas y metaheurísticas y, como caso particular, los algoritmos basados en colonias de hormigas. Estos han demostrado mejorar los resultados obtenidos por las técnicas de verificación clásicas usando un modelo muy sencillo pero a la vez extensible y potente. De todos los algoritmos de hormigas existentes, se optó por usar el BWAS [33] que ha sido usado satisfactoriamente para resolver diversos problemas de optimización.

  En cuanto a esta contribución, tenemos las siguientes publicaciones aceptadas: [23, 31, 33, 46]. Además el trabajo presentado en las Jornadas de Ingeniería del software y Base de Datos del año 2012 fue premiado como mejor artículo de la sesión "Ingeniería Web, Interfaces de Usuario, Sistemas Colaborativos, Computación Ubicua", y hemos mandado una versión extendida a la revista Applied Soft Computing. Además, una extensión del uso de BWAS aplicado al problema de la verificación está en segundo ronda de revisiones en la revista Expert Systems with Applications.

- **Tercera**: También hemos sido los primeros en abordar la verificación de la información de *warppers* web desde el punto de vista de la clasificación. Para eso se ha reformulado el problema de la verificación

como un problema de clasificación. Ahora bien, durante esta disertación veremos que las peculiaridades de la verificación automática de información nos han hecho usar un tipo determinado de clasificador denominado de una única clase. Este tipo de clasificador usa un proceso de aprendizaje no supervisado en el que no es necesario conocer de antemano todas las clases que podrán aparecer durante la fase de test. Además, hemos tenido que adaptar este tipo de clasificadores al problema de la verificación debido al tipo de características que se usan a la hora de caracterizar la información extraída por los *wrappers*. Toda esta investigación nos ha llevado a proponer un nuevo sistema de verificación multinivel, denominado MAVE, que usa clasificadores de una clase y que trata de forma independiente las características que se usan durante el proceso de verificación. Este nuevo verificador no sólo es capaz de comprobar si un verificador está fallando, sino que también permite detectar exactamente qué datos se están extrayendo mal, lo que permitirá simplificar el proceso de reinducción de *wrapper*. Por último, para poder comparar este nuevo verificador con el resto de verificadores propuestos, se ha diseñado un marco de trabajo para realizar las experimentaciones. También estamos en segunda fase de revisión en la revista IEEE Transactions on Knowledge and Data Engineering del artículo denominado "MAVE: Multilevel wrApper Verification systEm".

En cuanto a esta contribución, tenemos las siguientes publicaciones aceptadas: [43–45].

1.4. Colaboraciones

Durante el desarrollo del trabajo expuesto en esta disertación, se han recibido diversas colaboraciones que nos han permitido mejorar y perfilar los distintos objetivos de esta investigación. Las principales colaboraciones han sido las siguientes:

Universidad de Granada (España) . Una visita de investigación en el Departamento de Arquitectura y Tecnologías de Computadores de la Universidad de Granada invitado por la Dra. María Isabel García Arenas. El objetivo de esta visita fue perfilar el uso de sistemas bioinspirados a la verificación de información y sembrar las bases de futuros sistemas de verificación altamente distribuidos y paralelizables.

USC Information Sciences Institute (Estados Unidos) . La Dra. Kristina Lerman aportó todo el material del que disponía sobre su investigación en

verificación de *wrappers*. Gracias a ella, pudimos entender mejor su sistema de verificación y detectar aquellos elementos que podíamos mejorar.

University College Dublin (Irlanda) . El Dr. Nicholas Kushmerick nos aportó todo el material que disponía sobre verificación de *wrappers*. Gracias a su colaboración obtuvimos una base de datos completa y etiquetada que nos permitió hacer la experimentación de los distintos sistemas de verificación que estudiaremos a lo largo de esta disertación.

University of Illinois (Estados Unidos) . El Dr. Robert McCann y su equipo nos ayudaron en la comprensión de su sistema de verificación. Sus aclaraciones nos permitieron adaptarlo al marco de trabajo que se ha desarrollado.

Universidad de Sevilla (España) : El Dr. Rafael Corchuelo fue quien nos introdujo en el mundo de la verificación de *wrappers*, motivándonos su necesidad e indicándonos las principales propuestas existentes en este campo. Muchas de las ideas expuestas en este trabajo han surgido de las innumerables, e interminables, reuniones que hemos mantenido con él.

Universidad de Granada (España) . El Dr. Óscar Cordón y Dr. Francisco Herrera fueron los encargados de detallarnos todo lo relacionado con los algoritmos bioinspirados. De esta colaboración surgió el algoritmo BWAS.

Además de estas colaboraciones, esperamos trabajar en un futuro cercano con: Dr. Paulo Oliveira (Instituto Superior de Engenharia do Porto), Dr. Ismael Caballero (Universidad de Castilla-La Mancha) y el Dr. Alberto Freitas (Universidade do Porto).

1.5. Estructura de esta Tesis

Esta Tesis está organizada de la siguiente manera:

Parte I: Prefacio. Incluye esta introducción y el Capítulo §2, en el que se motiva este trabajo de investigación y se indican los problemas a los que se tienen que enfrentar los sistemas de verificación y se describirán las propuestas existentes carencias existentes en los sistemas actuales de verificación.

Parte II: Antecedentes. Proporciona información sobre los sistemas bioinspirados basados en colonias de hormigas y los clasificadores no supervisados de una clase. El Capítulo §3 trata la teoría básica que hay tras los algoritmos de colonias de hormigas que han servido de base para resolver el problema de la verificación desde el punto de vista de la optimización computacional. El Capítulo §4 presenta la clasificación no supervisada de una clase y la compara con la clasificación multiclase.

Parte III: Propuesta. Desarrolla las principales contribuciones contenidas en esta disertación. El Capítulo §5 presenta el marco de trabajo desarrollado y que se ha tomado como referencia a lo largo de toda esta investigación. El Capítulo §6 trata la propuesta del sistema de verificación de información basado en colonias de hormigas. El Capítulo §7, describe MAVE, la propuesta del sistema de verificación de información multinivel que se cimienta en formular el problema de la verificación de información como un problema de clasificación no supervisado. En el Capítulo §8 describiremos la base de datos usada para hacer la experimentación, detallaremos los experimentos realizados y expondremos los resultados obtenidos.

Part IV: Comentarios finales. Finaliza esta disertación y enfatiza algunas de las líneas de investigación futuras en el Capítulo §9.

Capítulo 2

Motivación

Si usted puede soñar eso, usted puede hacerlo.

Walt Disney (1901-1966), Productor, director, guionista y animador estadounidense.

Aunque la verificación de wrappers ha sido estudiada en profundidad por distintos investigadores, es necesario resolver los problemas que presentan las propuestas actuales. El objetivo de este capítulo es presentar estos problemas y ver las posibles vías de mejora. Este capítulo se estructura de la siguiente manera: en la Sección §2.1 estudiaremos la necesidad de crear verificadores; la Sección §2.2 presenta las principales carencias de las propuestas actuales; en la Sección §2.4 discutiremos sobre los problemas de las propuestas actuales y expondremos diversas soluciones; en la Sección §2.5 introduciremos nuestras propuestas que se desarrollarán a lo largo de toda esta disertación; finalmente, resumiremos el capítulo en la Sección §2.6.

2.1. Introducción

En la actualidad, la mayoría de la información que muestran los sitios web no es más que el resultado de aplicar una serie de consultas sobre diversos tipos de fuentes de datos. Si una aplicación de terceros quisiera acceder a dicha información sólo tendría que usar estas fuentes. Desafortunadamente, el acceso a ellas no suele estar disponible por diversos motivos (económicos, técnicos, inherente a las reglas de negocio, etc). Por tanto, el único recurso que nos queda es acceder a ellas a través de las webs a las que proveen información. Esta tarea es bastante compleja ya que los sitios web no muestran la información estructurada y su diseño está orientado al usuario final lo que imposibilita su procesamiento automático. Para afrontar este tipo de problemas se diseñan *wrappers*, sistemas que proporcionan una interfaz de programación para sitios web emulando la interacción que un usuario tendría con ellos.

Para ilustrar este problema, supongamos un sistema de integración como el que se muestra en la figura Figura §2.1 donde se esboza el funcionamiento de un metabuscador de referencias bibliográficas (WEB1) que usa como fuentes de información otros dos sitios web (WEB2 y WEB3). Tanto Web2 como Web3 tienen fuentes de información propias, grandes bases de datos de referencias bibliográficas, que sólo pueden ser consultadas mediante el interfaz web que éstas proporcionan. Cuando un usuario hace una consulta en WEB1, éste ejecuta una serie de consultas tanto en WEB2 como en WEB3 para obtener la información de interés para el usuario. El resultado de estas consultas son páginas HTML donde la información de interés, que para nuestro ejemplo es el título del artículo, la revista y el año de publicación, está formateada de forma tabular. A partir de estas páginas el *wrapper*, usando una serie de reglas de extracción basadas en la disposición tabular de los datos, es capaz de obtener los datos de interés y darle un formato estructurado. Suponga que, con el paso del tiempo, los diseñadores de WEB2 y WEB3 deciden, por ejemplo, intercambiar las columnas donde aparece el título del artículo y el nombre de la revista. Este pequeño cambio implicaría que, muy probablemente, WEB1 confundiría la información y la representaría erróneamente.

Para evitar este tipo de problemas, los *wrappers* incluyen un sistema de verificación cuyo objetivo es detectar automáticamente incorrecciones en los datos obtenidos por el extractor de información. Al problema de verificar la información extraída por un *wrapper* lo denominaremos *Wrapper Verification Problem (WVP)*.

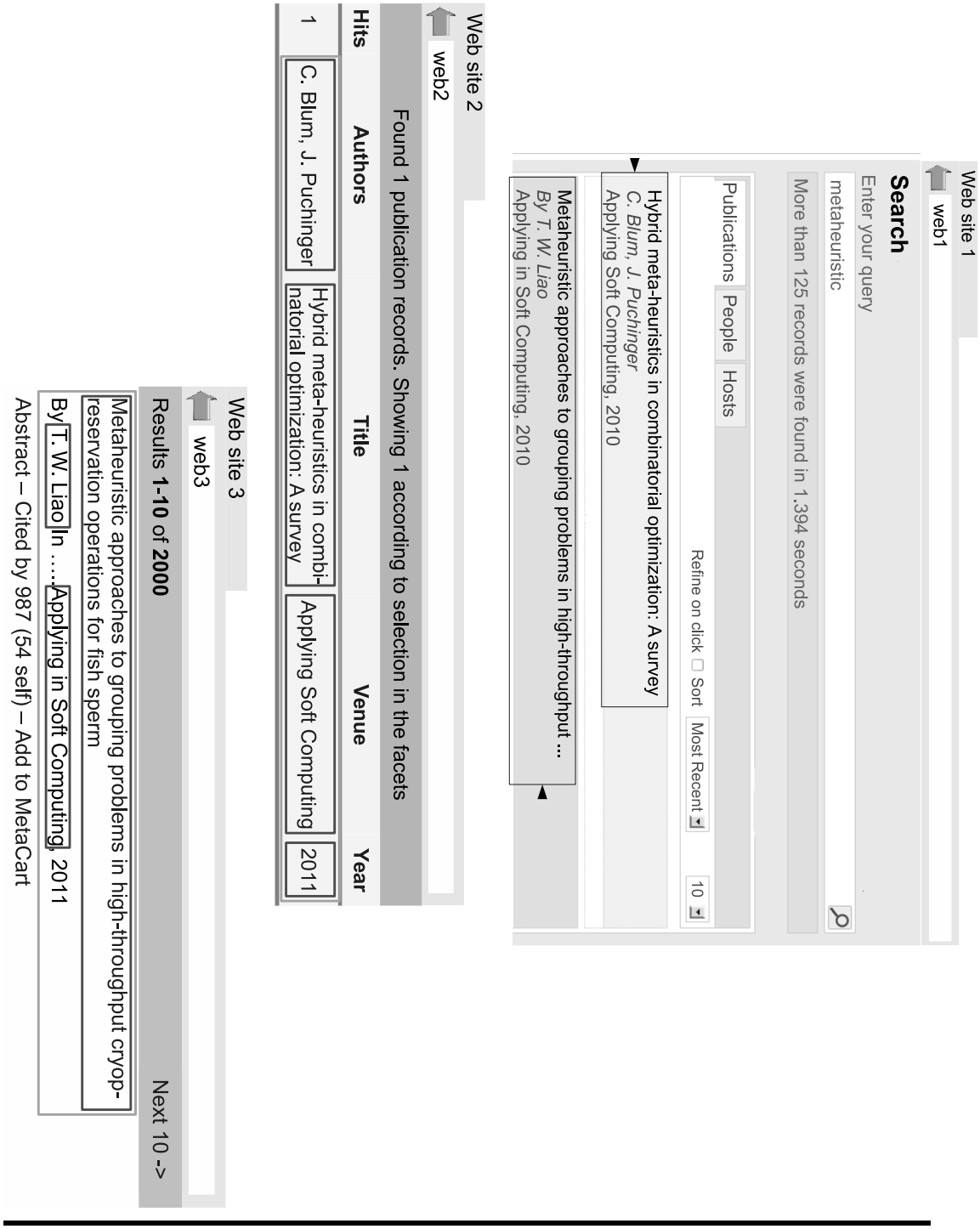


Figura 2.1: *Ejemplo del problema de la verificación*

2.2. Principales problemas

La verificación automática de información no es una tarea trivial y, si no se realiza adecuadamente, puede incrementar exponencialmente los costes de integración [155]. Los principales problemas a los que se tienen que enfrentar los sistemas de verificación son:

1. **(P1) Sobreaprendizaje**: Durante la fase de construcción del sistema de verificación sólo disponemos de ejemplos positivos (Sección §5.4), esto es, sólo tenemos representado una parte del dominio de discurso. Esto acarrea graves problemas de sobreaprendizaje ya que no estamos aprendiendo con todo lo que nos podemos encontrar en un futuro y nos ceñimos a aprender lo conocido (no generalizamos).

2. **(P2) Aplicación de características**: Los sistemas de verificación, reciben cadenas que representan los elementos de las web que contienen la información de interés. Para poder tratar estas cadenas, se aplican una serie de funciones matemáticas, denominadas **características**, que convierten cada cadena en un vector de valores. Atendiendo al dominio del sitio web, se aplicarán unas características u otras.

3. **(P3) Características independientes**: Los valores devueltos tras la aplicación de las características se pueden ver como variables aleatorias para las que no se puede asumir dependencia o independecia entre ellas.

4. **(P4) Conjuntos de datos heterogéneos**: A la hora de verificar información hay que tener en cuenta que, en muchas ocasiones, los conjuntos de datos a verificar no son homogéneos. Esto es, cadenas como `"Don Quijote de la Mancha"` y `"1492"` representan lo mismo, títulos de libros. En cambio, si los representamos mediante características pueden dar lugar a vectores de valores muy distantes en el espacio. En otras palabras, existe una dependencia entre la consulta, los datos que se obtuvieron y las características usadas para representarlos.

5. **(P5) Perfilado de las características**: Una vez que los datos de interés se caracterizan, se busca un modelo matemático que los describa. A priori este modelo no es conocido y los sistemas de verificación lo deben calcular.

6. **(P6) Poder de verificación**: A la hora de comprobar si un nuevo conjunto de datos es válido o no, hay que verificar si éste se ajusta al perfilado

previamente calculado. Al trabajar en un espacio n-dimensional, el grado de ajuste será mejor en algunas dimensiones que en otras por lo que debemos de disponer de mecanismos eficientes que combinen estos valores en uno único.

7. **(P7) Granularidad**: Cuando se detecta que un conjunto de datos es inválido, es especialmente importante que el sistema de verificación identifique qué elementos del conjunto son inválidos ya que la actuación del experto estaría más dirigida y el tiempo de reinducción del *wrapper* sería mucho menor.

2.3. Análisis de las soluciones actuales

Del estudio realizado sobre la literatura existen en el campo de la verificación de información [26, 69, 104], hemos identificado distintas propuestas. Muchas de ellas se centran en seleccionar un número predefinido de características y combinan los perfiles como suma ponderada, para luego ser comparado con un umbral predefinido que determinará si la alarma debe ser o no enviada [42, 133]; otras propuestas como las descritas en [65] o [27] presuponen que cambios en el árbol DOM de una página web pueden provocar una extracción incorrecta de los datos. Este tipo de sistema de verificación proponen características encaminadas a detectar estos cambios en vez de determinar si los datos extraídos siguen siendo correctos.

El sistema de verificación propuesto por Lerman y otros [105] parte de un conjunto de entrenamiento formado únicamente por conjuntos de datos válidos. Las características que utiliza para crear su modelo de verificación son: el número medio de registros por conjunto de datos, número medio de tokens, longitud media de los tokens, la densidad de letras, números, etiquetas HTML y signos de puntuación para los atributos de las clases. Cuando hablamos de tokens nos referimos a cadenas que contienen distintos tipos de caracteres alfabéticos, numéricos y de puntuación.

Lerman y otros [105] usan un algoritmo propio, denominado DATAPROG, para inferir los patrones de comienzo y fin y calcula el número de atributos de una clase que cumplen alguno de los patrones aprendidos para la clase.

Tanto el conjunto de entrenamiento, como cada uno de los conjuntos que se deseen verificar, están representados por un vector que contiene los valores medios de cada una de las características.

Para verificar un conjunto de datos no verificado simplemente comprueba si su vector y el del conjunto de entrenamiento son estadísticamente iguales para lo cual usa el estadístico de Pearson):

$$p == \sum_{f \in f_s} \frac{(r(f) - k(f))^2}{k(f)}$$

donde f_s es el conjunto de características consideradas, k es un mapa que asocia cada característica con valor medio en el conjunto de entrenamiento, y r es el mapa que asocia las características con sus valores en el conjunto de datos no verificado. Si p está por debajo de un nivel de confianza entonces se dispara una alarma.

Tsourakakis y Paliourast [149] modifican el estadístico de Pearson usado por Lerman y otros [105], para adaptarlo al conjunto de características usado. Si bien los autores indican que esto implica una mejora sustancial sobre la propuesta inicial, la poco concreción sobre las propuestas presentadas en este trabajo hace bastante difícil reproducir los resultados logrados.

Por otro lado Kushmerick [98, 100] presenta dos sistemas de verificación prácticamente idénticos. El conjunto de entrenamiento que usa para crear el modelo de verificación sólo tiene conjuntos de datos válidos. En su propuesta inicial trabajaba con las siguientes características: número de registros por conjunto de datos, número de palabras, longitud, fracción de dígitos, letras, letras mayúsculas, letras minúsculas, símbolos de puntuación y símbolos html. Posteriormente también incluyo características relacionadas con la estructura de los registros y el número de registros de cada una de las clases que hay en cada registro.

Estas características son modeladas como si de variables aleatorias que siguen una distribución de probabilidad se trataran. Para las variables numéricas sugiere usar una distribución normal mientras que para las categóricas usa una empírica que la crea basándose en los datos que estas variables toman en el conjunto de entrenamiento.

A la hora de verificar un conjunto de datos, primero calcula los valores que toman las distintas características en dicho conjunto. Luego obtiene la probabilidad con la que estos valores siguen las distribuciones antes calculadas. Para combinar estas probabilidades, hay una por característica, se sugieren tres alternativas:

- **Independencia.** Supone que las variables son independientes y calcula la probabilidad combinada como la multiplicación de todas ellas.

- **Vinculación.** Presume que hay una variable que vincula a otras y, por tanto, la probabilidad combinada es igual al menor valor.
- **Equivalencia.** Todas las variables tienen la misma importancia así que la probabilidad combinada es igual a la media geométrica de sus valores.

El valor que se obtiene de aplicar cualquiera de estos métodos es considerado una muestra de una distribución de probabilidad V. Dicha distribución se calculó previamente tomando como muestras las probabilidades con las que los datos del conjunto de entrenamiento siguen las distribuciones normales. Solo se lanza la alarma si la probabilidad con la que este valor sigue la distribución V es inferior a un límite. Pek y otros [126] introducen una nueva característica basada en el arbol DOM (caminos de extracción, número de hijos por nodo, número de nodos por datos extraibles) que mejora el proceso de verificación del sistema propuesto por Kushmerick [98, 100] ya que evalúa modificaciones en la estructura de la página antes de evaluar los valores extraidos.

Otro sistema muy conocido en el presentado por McCann y otros [115] que considera las siguientes características a nivel de conjunto de datos como: número de atributos de cada clase, longitud media, número medio de tokens, longitud media de los tokens, densidad de dígitos, densidad de tokens alfabéticos, densidad de tokens numéricos, densidad de tokens alfanuméricos, orden de tuplas atributos, una serie de características booleanas que indican el cumplimiento de restricciones semánticas definidas por el usuario. Además, para cada una de estas características, calcula su tendencia con el objetivo de detectar fluctuaciones en sus valores.

Cada uno de estas características es perfilada como si fuera una variables aleatoria que sigue una distribución normal. El proceso de verificación propuesto se basa en cuatro características fundamentales:

1. Las estimaciones de las características se calculan mediante densidades normalizas. Esto es, calcula la probabilidad de que una característica tenga un valor con más probabilidad que otro. Para una estimación gausiana, el valor de una variable aleatoria S se calcula como:

   $$1 - 2\Pr\left[\nu' \leq \mu - |\mu - \nu|\right]$$

   donde ν y ν' son valores de S.

Propuestas	P1	P2	P3	P4	P5	P6	P7
Kushmerick [98, 100]	×	×	×	✓	×	×	×
Lerman y otros [105]	×	✓	×	×	×	×	×
McCann y otros [115]	✓	×	×	×	×	×	×

P1 = Sobreaprendizaje; P2 = Aplicación de características; P3 = Características independientes; P4 = Conjuntos de datos heterogéneos; P5 = Perfilado de las características; P6 = Poder de verificación; P7 = Granularidad;

Tabla 2.1: *Comparación de propuestas actuales de sistemas de verificación*

2. Usa tanto conjuntos de valores válidos como inválidos. Cada característica se perfila mediante dos distribuciones normalizadas, una para los conjuntos de valores válidos y la otra para los inválidos. Aunque los autores proponen dos formas de combinar dichas distribuciones ninguna de ellas parece satisfactoria.

3. Calcula el valor combinado de las distintas características teniendo en cuenta el peso discriminatorio de cada una de ellas. Para los conjuntos de datos válidos los perfiles combinados se calculan como $p^+ = \sum_{p \in ps} wm(p)p(rs)$ y para los inválidos como $p^- = \sum_{p \in ps} wm(p)(1 - p(rs))$ donde *wm* es una función que mapea pesos con estimaciones, *rs* es un conjunto de datos y *ps* es el conjunto de características.

4. El usuario define un límite fijo que se usará para rechazar o no el conjunto a verificar.

2.4. Discusión

Las propuestas indicadas con anterioridad presentan una serie de problemas que hacen que su rendimiento en la práctica no sea óptimo (§2.1). Así, los sistemas actuales sufren de sobreaprendizaje (P1) ya que, salvo McCann y otros [115], sólo tratan con datos válidos. En [115] se propone el uso de ejemplos inválidos y se dan distintos métodos para generarlos pero las experimentaciones demuestran que estas técnicas dan lugar a ejemplos demasiado sintéticos y que no influyen positivamente en el proceso de construcción.

Respecto a la aplicación y dependencia de características (P2 y P3), todas las propuestas usan un número limitado de características independientemente del dominio del sitio web para el que se creo el wrapper. La propuesta de Lerman y otros [105] esta menos expuesta a este problema ya que la característica basada en DATAPROG parte de un esquema de etiquetas predefinido y adaptado al sitio web. Además, todos los autores presuponen la independencia de las características usadas cosa que, en la mayoría de los casos, no es cierta. Esto hace que, por ejemplo, el rendimiento del test χ^2 de bondad de ajuste usado en [105], la técnica *Naive Bayes* en la que se basa la técnica propuesta en [98], e incluyo el método *Winnow* usado en la propuesta descrita en [115] puedan dar unos resultados muy pobres.

Tanto Kushmerick [98] como McCann y otros [115] proponen modelar las características como distribuciones de probabilidad gausianas (P5), incluso en aquellos casos donde éstas no sigan este tipo de distribuciones. Además, la metodología que siguen para combinar las funciones de pertenencia de nuevos valores a dichas distribuciones se reduce a una única función matemática (P6). Todo esto se añade a que a la hora de crear el modelo de verificación Lerman y otros [105] y McCann y otros [115] implícitamente asumen que los datos a verificar son homogéneos (P4).

Por último, ninguna de las propuestas actuales es capaz de indicar qué elemento de un conjunto de datos marcado como inválido (P7), ha provocado la generación de la alarma. Esto implica que un experto será el que revise todo el conjunto de datos hasta detectar los datos potencialmente inválidos.

2.5. Propuesta

Para afrontar adecuadamente el problema de la verificación de información extraída por *wrappers* web, proponemos el desarrollo de un marco de trabajo en el que englobaremos los sistemas de verificación actuales con el objetivo de:

1. Homogeneizar el vocabulario usado en el campo de la verificación.
2. Identificar los diferentes elementos que forman parte del proceso de verificación.
3. Facilitar la comparación funcional entre sistemas de verificación.
4. Detectar los principales inconvenientes presentes en los sistemas de verificación.

En el Capítulo §5 se tratan todos estos aspectos. Se presentan diversos diagramas y algoritmos de los distintos bloques funcionales, donde se establecen las relaciones con los sistemas de verificación actuales y se justifica la necesidad de aplicar otro tipo de técnicas no usadas hasta la fecha en este campo. Así, por ejemplo, como consecuencia de proponer una base de datos de características común para todos sistemas de verificación es necesario aplicar algoritmos de selección de características (Sección §4.6) que permitan seleccionar las características que mejor se adecuen tanto al sitio web como a los tipos de datos.

Además, en el Capítulo §8 se presenta la idea de diseñar una metodología de experimentación que permita comparar empíricamente los distintos sistemas de verificación bajo las mismas condiciones, tanto de datos como de medidas de rendimiento.

Una vez establecidas las bases para diseñar y comparar sistemas de verificación, se proponen dos nuevos enfoques que tratan de paliar las carencias de los actuales. El primero de ellos (Capítulo §6) surge de la idea de tratar la verificación de la información como un problema de optimización computacional. En concreto, se establece un paralelismo con el problema de la asignación cuadrática de tareas (QAP) y se llega a la conclusión de que cualquier técnica aplicada a este campo es susceptible de ser usada en la verificación. De todas estas técnicas, usaremos las basadas en las colonias de hormigas para desarrollar un nuevo sistema de verificación. El algoritmo elegido es el de la mejor-peor hormiga (BWAS), también diseñado por el autor de esta Tesis.

El segundo de ellos, tratado en el Capítulo §7, se denomina MAVE (Multilevel wrApper Verification systEm) y es un sistema de verificación multinivel que tiene en cuenta la naturaleza de las características aplicadas. Previo al desarrollo de MAVE, el problema de la verificación se enfoca como un problema de clasificación de una única clase (Capítulo §7). Esta propuesta implicará una reformulación del problema y el estudio de una serie de técnicas (Sección §4.5).

Finalmente, se presenta un estudio comparativo basado en técnicas estadísticas no paramétricas (Capítulo §8) en el que estarán incluidas los sistemas de verificación tradicionales y todas las nuevas propuestas presentes en esta Tesis.

2.6. Resumen

En este capítulo hemos motivado las razones que nos han llevado a afrontar el problema de la verificación de *wrappers*. Hemos analizado los

problemas que se presentan a la hora de verificar información y las propuestas actuales y hemos llegado a la conclusión de que ninguna de las técnicas que se han venido usando solucionan todos los problemas planteados.

Parte II

Antecedentes

Capítulo 3

Metaheurísticas basadas en colonias de hormigas

No sigas órdenes toda tu vida. Piensa por ti mismo.

Chris Miller, Antz, Película 1998

La metaheurística de Optimización basada en Colonias de Hormigas (ACO) (Ant Colony Optimization) se inspira en el comportamiento seguido por las hormigas de diversas especies para encontrar los caminos más cortos entre las fuentes de comida y el hormiguero. En este capítulo repasaremos las bases de esta metaheurística y profundizaremos en el estudio de sus algoritmos más conocidos. La estructura del capítulo es la siguiente: En la Sección §3.1 hablaremos del concepto de metaheurística y justificaremos su necesidad de aplicación; en la Sección §3.2 se presentará el comportamiento real de las colonias de hormigas, a partir del cual se inspira toda la metaheurística ACO; a continuación, en la Sección §3.3 se describirá la transición desde las hormigas naturales hasta las hormigas artificiales, se resumirán las diferencias y similitudes entre ambas; en la Sección §3.4 se expondrá el modo de trabajo general de un algoritmo ACO y se discutirán, los tipos de problemas resolubles mediante ACO; en la Sección §3.5 se mostrarán algunos de los algoritmos ACO existentes. En la Sección §3.6 pasaremos a analizar de manera mas concreta y exhaustiva el Sistema de la Mejor-Peor Hormiga (BWAS). Finalmente, en la Sección §3.7 concluiremos el trabajo con unos comentarios finales.

3.1. Introducción

Existen problemas de optimización combinatoria complejos en diversos campos como la economía, el comercio, la ingeniería, la industria o la medicina. Sin embargo, a menudo estos problemas son muy difíciles de resolver en la práctica. El estudio de esta dificultad inherente para resolver dichos problemas tiene cabida en el campo de la teoría de las Ciencias de la Computación, ya que muchos de ellos pertenecen a la clase de problemas NP-duros, lo que significa que no existe un algoritmo conocido que los resuelva en un tiempo polinomial [75].

Día tras día, siguen apareciendo nuevos problemas de este tipo, lo que ha dado lugar a que se hayan realizado muchas propuestas de algoritmos para tratar de solucionarlos. Para permitir una mejora adicional en la calidad de las soluciones logradas por los algoritmos iterativos, la investigación en este campo en las últimas dos décadas ha centrado su atención en el diseño de técnicas de propósito general para guiar la construcción de soluciones o la búsqueda local en las distintas heurísticas. Estas técnicas se llaman comúnmente metaheurísticas [76, 122, 154] y son conceptos generales empleados para definir, a su vez, métodos heurísticos.

Una metaheurística relativamente reciente es la Optimización basada en Colonias de Hormigas (ACO) (*Ant Colony Optimization*), la cual se inspira en el comportamiento que rige a las hormigas de diversas especies para encontrar los caminos más cortos entre las fuentes de comida y el hormiguero. De hecho, desde el trabajo inicial de Dorigo y otros [61] en el Sistema de Hormigas (AS) (*Ant System*), la ACO se está convirtiendo en un campo de investigación importante: un gran número de autores han desarrollado modelos cada vez más sofisticados para solucionar de manera satisfactoria un gran número de problemas de optimización combinatoria. Igualmente se ha incrementado el número de desarrollos teóricos sobre los algoritmos que se proponen.

3.2. Las colonias de hormigas naturales

Las hormigas son insectos sociales que viven en colonias y que son capaces de mostrar comportamientos complejos y realizar tareas difíciles desde el punto de vista de una hormiga individual. Un aspecto interesante del comportamiento de muchas especies de hormigas es su habilidad para encontrar los caminos más cortos entre su hormiguero y las fuentes de alimento.

Mientras que se mueven entre el hormiguero y la fuente de alimento, algunas especies de hormigas depositan una sustancia química denominada feromona. Si no se encuentra ningún rastro de feromona, las hormigas se mueven de manera básicamente aleatoria, pero cuando existe feromona depositada, tienen mayor tendencia a seguir el rastro. De hecho, los experimentos realizados por biólogos han demostrado que las hormigas prefieren de manera probabilística los caminos marcados con una concentración superior de feromona [78, 125]. En la práctica, la elección entre distintos caminos tiene lugar cuando varios caminos se cruzan. Entonces, las hormigas eligen el camino a seguir con una decisión probabilística sesgada por la cantidad de feromona: cuanto más fuerte es el rastro de feromona, mayor es la probabilidad de elegirlo. Puesto que las hormigas depositan feromona en el camino que siguen, este comportamiento lleva a un proceso de auto-refuerzo que concluye con la formación de rastros señalados por una concentración de feromona elevada. Este comportamiento permite además a las hormigas encontrar los caminos más cortos entre su hormiguero y la fuente del alimento [78].

En la Figura §3.1 se ilustra cómo este mecanismo permite a las hormigas encontrar el camino más corto. Inicialmente no existe ningún rastro de feromona en el medio y, cuando una hormiga llega a una intersección, elige de manera aleatoria una de las bifurcaciones posibles. Según transcurre el tiempo y mientras que las hormigas están recorriendo los caminos más prometedores, estos van recibiendo una cantidad superior de feromona. Esto ocurre gracias a que al ser los caminos más cortos, las hormigas que los siguen consiguen encontrar la comida más rápidamente, por lo que comienzan su viaje de retorno antes. Entonces, en el camino más corto habrá un rastro de feromona ligeramente superior y, por lo tanto, las decisiones de las siguientes hormigas estarán dirigidas en mayor medida a dicho camino. Además, éste recibirá una proporción mayor de feromona por las hormigas que vuelven por él, que por las que vuelven por el camino más largo. Este proceso finaliza haciendo que la probabilidad de que una hormiga escoja el camino más corto aumente progresivamente y que al final el recorrido de la colonia converja al más corto de todos los caminos posibles.

Esta convergencia se complementa con la acción del entorno natural que provoca que la feromona se evapore transcurrido un cierto tiempo. Así, los caminos menos prometedores pierden progresivamente feromona porque son visitados cada vez por menos hormigas. En [14], numerosos experimentos muestran que, debido a la gran persistencia de feromona, es difícil que las hormigas olviden un camino que tiene un alto nivel de feromona aunque hayan encontrado un camino aún más corto..

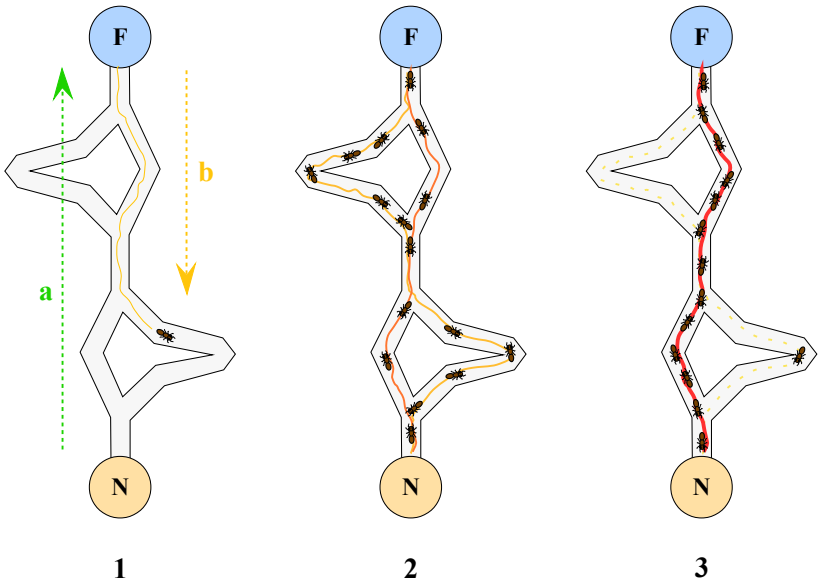


Figura 3.1: *Hormigas buscando el camino más corto*

3.3. De las hormigas naturales a la metaheurística ACO

Los algoritmos de ACO se inspiran directamente en el comportamiento de las colonias reales de hormigas para solucionar problemas de optimización combinatoria. Se basan en una colonia de hormigas artificiales, esto es, unos agentes computacionales simples que trabajan de manera cooperativa y se comunican mediante rastros de feromona artificiales. Los algoritmos de ACO son esencialmente algoritmos constructivos: en cada iteración del algoritmo, cada hormiga construye una solución al problema recorriendo un grafo de construcción. Cada arista del grafo, que representa los posibles pasos que la hormiga puede dar, tiene asociada dos tipos de información que guían el movimiento de la hormiga:

- **Información heurística:** Mide la preferencia heurística de moverse des-

de el nodo r hasta el nodo s. Se nota por η_{rs}. Las hormigas no modifican esta información durante la ejecución del algoritmo.

- **Información de los rastros de feromona artificiales:** mide la deseabilidad aprendida del movimiento de r a s. Se nota por τ_{rs}. Imita a la feromona real que depositan las hormigas naturales. Esta información se modifica durante la ejecución del algoritmo dependiendo de las soluciones encontradas por las hormigas.

En esta sección se presentan los pasos que llevan desde las hormigas reales a la ACO. A partir de ahora debe tenerse en cuenta que los algoritmos ACO presentan una doble perspectiva:

- Por un lado, son una abstracción de algunos patrones de comportamiento naturales relacionados con el comportamiento que permite encontrar el camino más corto.
- Por otro lado, incluyen algunas características que no tienen una contrapartida natural, pero que permiten que se desarrollen algoritmos para obtener buenas soluciones al problema que se pretende resolver (por ejemplo, el uso de información heurística que guíe el movimiento de las hormigas).

3.3.1. Tipos de problemas resolubles por la ACO

El tipo de problemas que se pueden resolver por medio de hormigas artificiales pertenece al grupo (restringido) de problemas de camino mínimo que se pueden caracterizar por los siguientes aspectos [55, 57]:

- Existe un conjunto de restricciones Ω definido por el problema a solucionar.
- Existe un conjunto finito de componentes $N = n_1, n_2, ..., n_l$.
- El problema presenta diversos estados que se definen según secuencias ordenadas de componentes $\delta =< n_r, n_s, ..., n_u, ... >$, que simplificaremos denotándola como $< r, s, ..., u, ... >$, sobre los elementos de N. Si Δ es el conjunto de todas las secuencias posibles, llamaremos $\tilde{\Delta}$ al conjunto de posibles (sub)secuencias que respetan las restricciones Ω. Los elementos en $\tilde{\Delta}$ definen los estados posibles. $|\delta|$ es la longitud de una secuencia δ, esto es, el número de componentes en la secuencia.

- Existe una estructura de vecindario definida como sigue: δ_2 es un vecino de δ_1 si:
  - $\delta_1 \wedge \delta_2 \in \Delta$,
  - el estado δ_2 puede alcanzarse desde δ_1 en un paso lógico, es decir, si r es la última componente de la secuencia δ_1, debe existir una componente $s \in N$ tal que $\delta_2 =< \delta_1, s >$, esto es, debe existir una transición válida entre r y s. El vecindario alcanzable de δ_1 es el conjunto que contiene todas las secuencias $\delta_2 \in \tilde{\Delta}$. Si $\delta_2 \notin \tilde{\Delta}$, diremos que δ_2 está en el vecindario no alcanzable de δ_1.
- Una solución S es un elemento de $\tilde{\Delta}$ que verifica todos los requisitos del problema.
- Existe un costo $C(S)$ asociado a cada solución S. A esta función también se le suele llamar *fitness*.
- En algunos casos, se puede asociar un costo, o una estimación del mismo, a los estados.

Como hemos comentado todas las características anteriores se dan en problemas de optimización combinatoria que pueden representarse en forma de grafo ponderado $G = (N, A)$, donde A es el conjunto de aristas que conectan el conjunto de componentes N. El grafo G se denomina grafo de construcción G. Por tanto, tenemos que:

- Las componentes n_r son los nodos del grafo.
- Los estados δ (y por tanto las soluciones S) se corresponden con caminos en el grafo, esto es, secuencias de nodos o aristas.
- Las aristas del grafo, a_{rs} , son conexiones/transiciones que definen la estructura del vecindario. $\delta_2 =< \delta_1, s >$ será vecino de δ_1 si el nodo r es la última componente de δ_1 y la arista a_{rs} existe en el grafo.
- Deben existir los costos explícitos c_{rs} asociados con cada arista.
- Las componentes o conexiones deben tener asociados rastros de feromona τ, que representan un tipo de memoria indirecta y a largo plazo del proceso de búsqueda, como valores heurísticos η, que representan la información heurística disponible en el problema a resolver.

3.3.2. La hormiga artificial

La hormiga artificial es un agente computacional simple que intenta construir soluciones posibles al problema explotando los rastros de feromona disponibles y la información heurística. Sin embargo, en algunos problemas, puede también construir soluciones no válidas que podrán ser penalizadas dependiendo de lo inadecuado de la solución. La hormiga artificial tiene las siguientes propiedades [55, 57]:

- Busca soluciones válidas de costo mínimo para el problema a solucionar.
- Tiene una memoria L que almacena información sobre el camino seguido hasta el momento, esto es, L almacena la secuencia generada. Esta memoria puede usarse para:
  - construir soluciones válidas,
  - evaluar la solución generada, y
  - reconstruir el camino que ha seguido la hormiga.
- Tiene un estado inicial $\delta_{inicial}$, que normalmente se corresponde con una secuencia unitaria y una o más condiciones t de parada asociadas.
- Comienza en el estado inicial y se mueve siguiendo estados válidos, construyendo la solución asociada incrementalmente.
- Cuando está en un estado $\delta_r =< \delta_{r-1}, r >$, es decir, actualmente está localizada en el nodo r y ha seguido previamente la secuencia δ_{r-1}, puede moverse a cualquier nodo s de su vecindario posible $N(r)$, definido como $N(r) = s|(a_{rs} \in A)\ y\ (< \delta_r, s > \in \tilde{\Delta})$.
- El movimiento se lleva a cabo aplicando una regla de transición, que es función de los rastros de feromona que están disponibles localmente, de los valores heurísticos de la memoria privada de la hormiga y de las restricciones del problema.
- Cuando durante el procedimiento de construcción una hormiga se mueve desde el nodo r hasta el s, puede actualizar el rastro de feromona τ_{rs} asociado al arco a rs . Este proceso se llama actualización en línea de los rastros de feromona paso a paso.

- El procedimiento de construcción acaba cuando se satisface alguna condición de parada, normalmente cuando se alcanza un estado objetivo.
- Una vez que la hormiga ha construido la solución puede reconstruir el camino recorrido y actualizar los rastros de feromona de los arcos/componentes visitados utilizando un proceso llamado actualización en línea a posteriori. Este es el único mecanismo de comunicación entre las hormigas, utilizando la estructura de datos que almacena los niveles de cada arco/componente (memoria compartida).

3.4. Modo de funcionamiento y estructura de un algoritmo de ACO

Como se ha visto en las secciones anteriores, el modo de operación básico de un algoritmo de ACO es como sigue: las m hormigas (artificiales) de la colonia se mueven, concurrentemente y de manera asíncrona, a través de los estados adyacentes del problema (que puede representarse en forma de grafo con pesos). Este movimiento se realiza siguiendo una regla de transición que está basada en la información local disponible en las componentes (nodos). Esta información local incluye la información heurística y memorística (rastros de feromona) para guiar la búsqueda. Al moverse por el grafo de construcción, las hormigas construyen incrementalmente soluciones. Opcionalmente, las hormigas pueden depositar feromona cada vez que crucen un arco (conexión) mientras que construyen la solución (actualización en línea paso a paso de los rastros de feromona). Una vez que cada hormiga ha generado una solución se evalúa ésta y puede depositar una cantidad de feromona que es función de la calidad de su solución (actualización en línea a de los rastros de feromona). Esta información guiará la búsqueda de las otras hormigas de la colonia en el futuro.

Además, el modo de operación genérico de un algoritmo de ACO incluye dos procedimientos adicionales, la evaporación de los rastros de feromona y las acciones del demonio. La evaporación de feromona la lleva a cabo el entorno y se usa como un mecanismo que evita el estancamiento en la búsqueda y permite que las hormigas busquen y exploren nuevas regiones del espacio. Las acciones del demonio son acciones opcionales (que no tienen un contrapunto natural) para implementar tareas desde una perspectiva global que no pueden llevar a cabo las hormigas por la perspectiva local que ofrecen.

La estructura de un algoritmo de ACO genérico [56, 58] se indica en el Algoritmo §3.1.

```
MATAHEURISTICA_ACO()
1  INICIALIZACIÓN_DE_PARÁMETROS()
2  // t= criterio de finalización
3  while not t
4       PROGRAMACIÓN_DE_ACTIVIDADES()
PROGRAMACIOŃ_DE_ACTIVIDADES()
1  GENERACIÓN_DE_HORMIGAS_Y_ACTIVIDAD()
2  EVAPORACIÓN_DE_FEROMONA()
3  ACCIONES_DEL_DEMONIO()
GENERACIÓN_DE_HORMIGAS_Y_ACTIVIDAD()
1  // Creación en paralelo de las hormigas
2  for i = 1 to m
3       NUEVA_HORMIGA(i)
NUEVA_HORMIGA(k)
 1  INICIALIZA_HORMIGA(k)
 2  r = INICIALIZA_ESTADO_HORMIGA(k)
 3  L = INICIALIZA_MEMORIA_HORMIGA(k, r)
 4  δ = r
 5  while (δ ≠ estado_objetivo)
 6       P_k = CALCULAR_PROBABILIDADES_DE_TRANSICIÓN(k, L, N_k(r))
 7       s = APLICAR_POLÍTICA_DECISIÓN(P, N_k(r))
 8       L_k = ACTUALIZAR_MEMORIA_HORMIGA(k, s)
 9       δ = ACTUALIZA_ESTADO_HORMIGA(k, s)
10       if (ACTUALIZACION_FEROMONA_EN_LINEA_PASO_A_PASO())
11            DEPOSITAR_FEROMONA_EN_EL_ARCO_VISITADO(k, r, s)
12       L_k = L_k s
13       S_k =< S_k, s >
14       r = s
ACTIVIDADES_DEL_DEMONIO()
1  if (ACTUALIZACION_FEROMONA_EN_LINEA_A_POSTERIORI())
2       for k = 1 to m
3            for each < r, s >∈ L_k          // <r,s>=vértices visitados
4                 DEPOSITAR_FEROMONA_EN_EL_ARCO(k, r, s)
```

Programa 3.1: *Algoritmo ACO genérico*

El primer paso incluye la inicialización de los valores de los parámetros que se tienen en consideración en el algoritmo. Entre otros, se deben fijar el rastro inicial de feromona asociado a cada transición, τ_o, que es un valor positivo pequeño, normalmente el mismo para todas las componentes/conexiones, el número de hormigas en la colonia, m, y los pesos que definen la proporción en la que afectarán la información heurística y memorística en la regla de transición probabilística. El procedimiento principal de la metaheurística ACO, METAHEURISTICA_ACO() controla la planificación de las tres componentes mencionadas en esta sección:

1. la generación y puesta en funcionamiento de las hormigas artificiales,
2. la evaporación de feromona, y
3. las acciones del demonio.

La implementación de este constructor determinará la sincronía existente entre cada una de las tres componentes. Mientras que la aplicación a problemas clásicos NP-duros, normalmente usa una planificación secuencial. En problemas distribuidos como el enrutamiento en redes, el paralelismo puede ser explotado de manera sencilla y eficiente. Como se ha comentado antes, varias componentes son o bien opcionales, como las acciones del demonio, o bien dependientes estrictamente del algoritmo de ACO especifico, por ejemplo cuándo y cómo se deposita la feromona. Generalmente, la actualización en línea paso a paso de los rastros de feromona y la actualización en línea a posteriori de los rastros de feromona son mutuamente excluyentes y no suelen estar presentes a la vez ni faltar ambas al mismo tiempo (si las dos faltan, el demonio suele actualizar los rastros de feromona).

Por otro lado, hay que remarcar que el procedimiento denominado ACTUALIZA_MEMORIA_HORMIGA() se encarga de especificar el estado inicial desde el que la hormiga comienza su camino y, además almacenar la componente correspondiente en la memoria de la hormiga L. La decisión sobre cuál será dicho nodo depende del algoritmo específico (puede ser una elección aleatoria o una fija para toda la colonia, o una elección aleatoria o fija para cada hormiga, etc.). Por último, comentar que los procedimientos CALCULAR_PROBABILIDADES_DE_TRANSICIÓN() y APLICAR_POLÍTICA_DECISIÓN() tienen en consideración el estado actual de la hormiga, los valores actuales de la feromona visibles en dicho nodo y las restricciones del problema Ω para establecer el proceso de transición probabilístico hacia otros estados válidos.

3.5. Modelos de optimización basada en colonias de hormigas

En la literatura se han propuesto diversos algoritmos que siguen la metaheurística ACO. Entre los algoritmos de ACO disponibles para problemas de optimización combinatoria NP-duros, se encuentran el Sistema de Hormigas (AS) [61], el Sistema de Colonia de Hormigas (ACS) [59], el Sistema de Hormigas Max-Min (MMAS) [142], el AS con ordenación (RAS) [17] y el Sistema de la Mejor-Peor Hormiga (BWAS) [37, 38]. En las siguientes secciones, presentaremos una pequeña descripción de los algoritmos AS (Sección §3.5.1) y ACS (Sección §3.5.2), que nos servirán de base para entender el comportamiento del algoritmo BWAS que se detallará en la Sección §3.6.

3.5.1. El sistema de hormigas

El AS [61] fue el primer algoritmo de ACO. Inicialmente, se presentaron 3 variantes distintas: AS-densidad, AS-cantidad y AS-ciclo, que se diferencian en la manera en que se actualizan los rastros de feromona. En los dos primeros, las hormigas depositan feromona mientras que construyen sus soluciones (esto es, aplicaban una actualización en-línea paso a paso de feromona), con la diferencia de que la cantidad de feromona depositada en el AS-densidad es constante, mientras que la depositada en AS-cantidad depende directamente de la deseabilidad heurística de la transición η_{rs}. Por último, en AS-ciclo la deposición de feromona se lleva a cabo una vez que la solución está completa (actualización en línea a posteriori de feromona). Esta última variante es la que obtiene unos mejores resultados y es, por tanto, la que se conoce como AS en la literatura.

En el AS la actualización de feromona se realiza una vez que todas las hormigas han completado sus soluciones, y se lleva a cabo como sigue: primero, todos los rastros de feromona se reducen en un factor constante, implementándose de esta manera la evaporación de feromona:

$$\tau_{rs} \leftarrow (1 - \rho) \bullet \tau_{rs} \tag{3.1}$$

donde $\rho \in (0, 1]$ es la tasa de evaporación. A continuación cada hormiga recorre de nuevo el camino que ha seguido (el camino esta almacenado en su memoria local L_k) y deposita una cantidad de feromona $\Delta\tau^k_{rs}$, que es función de la calidad de su solución, en cada conexión por la que ha viajado:

$$\tau_{rs} \leftarrow \tau_{rs} + \Delta\tau^k_{rs}, \ \forall a_{rs} \in S_k \tag{3.2}$$

donde $\Delta\tau_{rs}^{k} = f(C(S_k))$.

En cada paso de construcción, una hormiga k escoge ir al siguiente nodo con una probabilidad que se calcula como:

$$p_{r,s}^{k} = \begin{cases} \frac{\tau_{rs}^{\alpha} \bullet \eta_{rs}^{\beta}}{\sum_{u \in N_r^k} \tau_{rs}^{\alpha} \bullet \eta_{rs}^{\beta}} & , \text{ si } \quad s \epsilon N_k(r) \\ 0 & \text{en otro caso} \end{cases} \tag{3.3}$$

donde $N_k(r)$ es el vecindario alcanzable por la hormiga k cuando se encuentra en el nodo r, y $\alpha,\ \beta \in \mathbb{R}$, son dos parámetros que ponderan la importancia relativa de los rastros de feromona y la información heurística. Cada hormiga k almacena la secuencia que ha seguido hasta el momento y su memoria L_k; esta información se utiliza,como se mostró anteriormente, para determinar $N_k(r)$ en cada paso de construcción. La función de los parámetros α y β es la que sigue: si $\alpha = 0$, aquellos nodos con una preferencia heurística mejor tienen una mayor probabilidad de ser escogidos, haciendo el algoritmo muy similar a un algoritmo voraz probabilístico clásico (con múltiples puntos de partida en caso de que las hormigas estén situadas en nodos distintos al comienzo de cada iteración). Sin embargo, si $\beta = 0$, sólo se tienen en cuenta los rastros de feromona para guiar el proceso constructivo, lo que puede causar un rápido estancamiento. Por tanto es preciso establecer una adecuada proporción entre la información heurística y la información de los rastros de feromona.

Para resumir la descripción del AS, en el algoritmo §3.2 se detalla el comportamiento de este algoritmo.

3.5.2. El sistema de colonias de hormigas

El ACS [59] es uno de los primeros sucesores del AS que introduce tres modificaciones importantes con respecto a dicho algoritmo de ACO:

1. El ACS usa una regla de transición distinta, denominada regla proporcional pseudo-aleatoria. Sea k una hormiga situada en el nodo r, $q_0 \in [0, 1]$ un parámetro y q un valor aleatorio en $[0, 1]$, el siguiente nodo s se elige aleatoriamente mediante la siguiente distribución de probabilidad:

   si $q < q_0$

$$p_{rs}^{k} = \begin{cases} 1, & \text{si } s = \underset{u \in N_k(r)}{arg\ max}\{\tau_{ru}^{\alpha} \eta_{ru}^{\beta}\} \\ 0, & \text{en otro caso} \end{cases} \tag{3.4}$$

CALCULAR_PROBABILIDADES_DE_TRANSICIÓN$(k, L, N_k(r))$

1 **for** each $s \in N_k(r)$

2 $\quad p^k_{r,s} = \frac{\tau^\alpha_{rs} \bullet \eta^\beta_{rs}}{\sum_{u \in N_k(r)} \tau^\alpha_{rs} \bullet \eta^\beta_{rs}}$

3 **return** P

ACTUALIZACIÓN_FEROMONA_EN_LINEA_PASO_A_PASO()

1 **return** FALSE

ACTUALIZACIÓN_FEROMONA_EN_LINEA_A_POSTERIORI()

1 **return** TRUE

DEPOSITAR_FEROMONA_EN_ARCO(k, r, s)

1 $\tau_{rs} \leftarrow \tau_{rs} + f(C(S_k))$

EVAPORACIÓN_DE_FEROMONA(G)

1 **for** each $a_{rs} \in G$

2 $\quad \tau_{rs} \leftarrow (1 - \rho) \bullet \tau_{rs}$

Programa 3.2: *Algoritmo AS*

si no

$$p^k_{r,s} = \begin{cases} \frac{\tau^\alpha_{rs} \bullet \eta^\beta_{rs}}{\sum_{u \in N^k_r} \tau^\alpha_{rs} \bullet \eta^\beta_{rs}} & , \text{ si } \quad s \epsilon N_k(r) \\ 0 & \quad en\ otro\ caso \end{cases} \tag{3.5}$$

Como puede observarse, la regla tiene una doble intención: cuando $q < q_0$, explota el conocimiento disponible, eligiendo la mejor opción con respecto a la información heurística y los rastros de feromona.

Sin embargo, si $q > q_0$ se aplica una exploración controlada, tal como se hacia en el AS. En resumen, la regla establece un compromiso entre la exploración de nuevas conexiones y la explotación de la información disponible en ese momento.

2. Sólo el demonio (y no las hormigas individualmente) actualiza la feromona, es decir, se realiza una actualización de feromona fuera de línea de los rastros. Para llevarla a cabo, el ACS sólo considera una hormiga concreta, la que generó la mejor solución global, $S_{mejor-global}$ (aunque en algunos trabajos iniciales se consideraba también una actualización basada en la mejor hormiga de la iteración [59], en ACO casi

siempre se aplica la actualización por medio de la mejor global). La actualización de la feromona se hace evaporando primero los rastros de feromona en todas las conexiones utilizadas por la mejor hormiga global (es importante recalcar que, en el ACS, la evaporación de feromona sólo se aplica a las conexiones de la solución, que es también la usada para depositar feromona) tal como sigue:

$$\tau_{rs} \leftarrow (1 - \rho) \bullet \tau_{rs}, \ \forall a_{rs} \in S_{mejor-global} \tag{3.6}$$

A continuación, el demonio deposita feromona usando la regla:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho \bullet f(C(S_{mejor-global})), \ \forall a_{rs} \in S_{mejor-global} \tag{3.7}$$

Adicionalmente, el demonio puede aplicar un algoritmo de búsqueda local para mejorar las soluciones de las hormigas antes de actualizar los rastros de feromona.

3. Las hormigas aplican una actualización en línea paso a paso de los rastros de feromona que favorece la generación de soluciones distintas a las ya encontradas. Cada vez que una hormiga viaja por una arista rs , aplica la regla:

$$\tau_{rs} \leftarrow (1 - \phi) \bullet \tau_{rs} + \phi \bullet \tau_0 \Delta\tau^k_{rs}, \ \forall a_{rs} \in S_k \tag{3.8}$$

donde $\phi \in (0, 1]$ es un segundo parámetro de decremento de feromona. Como puede verse, la regla de actualización en línea paso a paso incluye tanto la evaporación de feromona como la deposición de la misma. Ya que la cantidad de feromona depositada es muy pequeña (de hecho, τ_0 es el valor del rastro de feromona inicial y se escogiese de tal manera que, en la práctica, se corresponda con el límite menor de rastro de feromona; esto es, con la elección de las reglas de actualización de feromona del ACS ningún rastro de feromona puede caer por debajo de τ_0), la aplicación de esta regla hace que los rastros de feromona entre las conexiones recorridas por las hormigas disminuyan.

Así, esto lleva a una técnica de exploración adicional del ACS ya que las conexiones atravesadas por un gran número de hormigas son cada vez menos atractivas para el resto de hormigas que las recorren en la iteración actual, lo que ayuda claramente a que no todas las hormigas sigan el mismo camino.

El funcionamiento general de este algoritmo se describe en el Algoritmo §3.3.

```
CALCULAR_PROBABILIDADES_DE_TRANSICIÓN(k, L, N_k(r))
 1  for each s ∈ N_k(r)
 2        b_rs = τ_rs • η_rs^β
 3  q=random(0,1)
 4  if q < q0
 5        // Obtengo el arco con mayor b
 6        z = max(b_rs, N_k(r))
 7        p^k_{r,z} = 1
 8        p^k_{r,x} = 0, ∀x ≠ z
 9  else
10        for each s ∈ N_k(r)
11              p^k_{r,s} = b_rs / Σ_{u∈N_r(k)} b_ru
12  return P
ACTUALIZACIÓN_FEROMONA_EN_LINEA_PASO_A_PASO()
1  return TRUE
DEPOSITAR_FEROMONA_EN_ARCO_VISITADO(k, r, s)
1  τ_rs ← (1 − ϕ) • τ_rs + ϕ • τ_0
ACTUALIZACIÓN_FEROMONA_EN_LINEA_A_POSTERIORI()
1  return TRUE
DEPOSITAR_FEROMONA_EN_ARCO_VISITADO(k, r, s)
1  τ_rs ← (1 − ϕ) • τ_rs + ϕ • τ_0
EVAPORACIÓN_DE_FEROMONA(G)
1  for each a_rs ∈ S_mejor−global
2        τ_rs ← (1 − ρ) • τ_rs
ACCIONES_DEL_DEMONIO()
1  mejor − actual = S_1
2  for k = 1 → m
3        S_k = busqueda_local(S_k)
4        if C(S_k) < C(mejor − actual)
5              mejor − actual = S_k
6  if C(mejor_actual) < C(mejor_global)
7        mejor_global = mejor_actual
8  for each a_rs ∈ S_mejor−global
9        τ_rs ← (1 − ϕ) • τ_rs + ϕ • τ_0
```

Programa 3.3: *Algoritmo ACS*

3.6. El sistema de la mejor-peor hormiga

Dentro del campo de las colonias de hormigas, nuestra propuesta es el algoritmo denominado mejor-peor hormiga o BWAS [39, 40]. Este algoritmo incorpora conceptos de computación evolutiva y es otra extensión del AS cuyo modo de funcionamiento se detalla en el Algoritmo §3.4.

El BWAS utiliza la misma regla de transición de estados que el AS, así como la misma regla de evaporación de feromona que, al igual que en el AS con ordenación y el MMAS, se aplica a todas las transiciones. Además, tal como hace el MMAS, el BWAS siempre considera la explotación sistemática de optimizadores locales para mejorar las soluciones de las hormigas.

El núcleo del BWAS lo componen las siguientes tres acciones del demonio que se describen en los tres apartados siguientes.

3.6.1. Actualización de la feromona

La regla mejor-peor de actualización de rastros de feromona está basada en la regla de actualización del vector de probabilidades de PBIL que refuerza las aristas que se encuentran en la mejor solución global. Además, penaliza cada conexión de la peor solución generada hasta el momento, $S_{peor-actual}$, que no se encuentre en la mejor global realizando una evaporación de feromona adicional de esos rastros. Por tanto, la regla de actualización de feromona en el BWAS se convierte en:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho \bullet f(C(S_{mejor\_global})),\ \forall a\ rs \in S_{mejor\_global} \tag{3.9}$$

$$\tau_{rs} \leftarrow (1-\rho) \bullet \tau_{rs},\ \forall a\ rs \in S_{peor\_global} \wedge rs \notin S_{mejor\_global} \tag{3.10}$$

3.6.2. Mutación de la feromona

Se realiza una mutación de los rastros de feromona para introducir diversidad en el proceso de búsqueda. Para llevarla a cabo, el rastro de feromona asociado a cada una de las transiciones, desde cada nodo se muta con una probabilidad P_m utilizando cualquier operador de mutación con codificación real.

La propuesta original del BWAS aplicaba un operador que alteraba los rastros de feromona de cada transición mutada añadiendo o restando la misma cantidad en cada iteración. El rango de mutación $mut(it, \tau_{umbral})$, que

```
CALCULAR_PROBABILIDADES_DE_TRANSICIÓN(k, L, N_k(r))
 1  for each s ∈ N_k(r)
 2        b_rs = τ_rs • η_rs^β
 3
 4  q=random(0,1)
 5  if q < q0
 6        // Obtengo el arco con mayor b
 7        z = max(b_rs, N_k(r))
 8        p^k_{r,z} = 1
 9        p^k_{r,x} = 0, ∀x ≠ z
10  else
11        for each s ∈ N_k(r)
12              p^k_{r,s} = b_rs / Σ_{u∈N_r(k)} b_ru
13  return P
ACTUALIZACIÓN_FEROMONA_EN_LINEA_PASO_A_PASO()
1  return TRUE
DEPOSITAR_FEROMONA_EN_ARCO_VISITADO(k, r, s)
1  τ_rs ← (1 − ϕ) • τ_rs + ϕ • τ_0
ACTUALIZACIÓN_FEROMONA_EN_LINEA_A_POSTERIORI()
1  return TRUE
DEPOSITAR_FEROMONA_EN_ARCO_VISITADO(k, r, s)
1  τ_rs ← (1 − ϕ) • τ_rs + ϕ • τ_0
EVAPORACIÓN_DE_FEROMONA(G)
1  for each a_rs ∈ S_mejor−global
2        τ_rs ← (1 − ρ) • τ_rs
ACCIONES_DEL_DEMONIO()
1  mejor − actual = S_1
2  for k = 1 → m
3        S_k = busqueda_local(S_k)
4        if C(S_k) < C(mejor − actual)
5              mejor − actual = S_k
6  if C(mejor_actual) < C(mejor_global)
7        mejor_global = mejor_actual
8  for each a_rs ∈ S_mejor−global
9        τ_rs ← (1 − ϕ) • τ_rs + ϕ • τ_0
```

Programa 3.4: *Algoritmo BWAS*

depende de la media de los rastros de feromona en las transiciones de la mejor solución global, τ_{umbral}, es más suave en las primeras etapas del algoritmo -donde no hay riesgo de estancamiento- y más fuerte en las últimas etapas, donde el peligro de estancamiento es más fuerte:

$$\tau'_{rs} = \begin{cases} \tau_{rs} + mut(it, \tau_{umbral}), & si\ a = 1 \\ \tau_{rs} - mut(it, \tau_{umbral}), & si\ a = 0 \end{cases} \tag{3.11}$$

donde a es una valor aleatorio en $0, 1$ e it es la iteración actual.

3.6.3. Reinicialización

Como en otros modelos de ACO, el BWAS considera la reinicialización de los rastros de feromona cuando se estanca la búsqueda, lo que se lleva a cabo fijando cada rastro de feromona a τ_0. En las primeras versiones del algoritmo se comprobaba si el porcentaje de arcos distintos entre la mejor solución y la peor de la población actual era menor que un cierto valor umbral. En versiones más recientes se utiliza un concepto diferente para evaluar si el algoritmo se ha estancado o no: se reinicializa sólo si el algoritmo está sin obtener una mejor solución durante un porcentaje de iteraciones.

3.7. Resumen

En este capítulo hemos estudiado la metaheurística ACO, detallando las bases biológicas que la sustentan y estableciendo una relación directa entre el comportamiento de las hormigas naturales y las artificiales. Además, se han dado las pautas a seguir para solucionar cualquier problema de optimización combinacional usando este tipo de técnicas. Por último, se ha hecho un estudio de los algoritmos ACO más referenciados en la bibliografía y comparamos la metaheurística ACO frente a otras técnicas usadas en la resolución de problemas de optimización combinatoria. Esta comparación ha servido como base para presentar el algoritmo BWAS que se puede considerar un algoritmo híbrido ya que mezcla conceptos aplicados en distintas metaheurísticas.

Capítulo 4

Clasificadores de una clase

Se dice que existen tres clases de testigos: Los que han visto bien, pero dudan de lo que han visto. Los que han visto mal, pero creen haber visto bien. Y los que no han visto nada y aseguran haber visto todo.

Marco Aurelio Almazán, Escritor y diplomático mexicano (1922-1991)

Las técnicas de clasificación multiclase suelen asumir la perfecta caracterización de todas las clases a las que pertenecen los elementos a etiquetar. En problemas de clasificación reales esto no es así ya que, habitualmente, una o varias de estas clases no se encuentren suficientemente representadas cuando construimos el clasificador. En estos casos, es necesario usar otras técnicas de clasificación denominadas de una clase (one class).

Este capítulo está organizado de la siguiente manera: en la Sección §4.1 introduciremos los clasificadores de una clase; en la Sección §4.2 explicaremos los conceptos básicos de clasificación; en la Sección §4.3 profundizaremos en el estudio de los clasificadores de una clase comparándolos con los multiclase y enumerando aquellas características a tener en cuenta a la hora de elegir entre un clasificador de una clase u otro; en la Sección §4.4 expondremos distintos criterios que se han venido utilizando a la hora de agrupar a los clasificadores de una clase; en la Sección §4.5 estudiaremos las familias y métodos de clasificación de una clase más conocidos; en la Sección §4.6 hablaremos de técnicas de selección de características que son especialmente importantes para los clasificadores de una clase; por último, resumiremos los distintos conceptos tratados en este capítulo en la Sección §4.7.

4.1. Introducción

El problema de la clasificación se puede definir como la tarea de asignar etiquetas o clases a objetos (también denominados instancias o elementos observados). Éstos son representados mediante vectores que almacenan los valores resultantes de aplicar una serie de características sobre ellos.

En los problemas de clasificación supervisada [118], el clasificador cuenta con un conocimiento a priori, es decir, para cada una de las clases contamos con objetos ya clasificados. A esta serie de objetos se le denomina conjunto de entrenamiento y, gracias a él, los clasificadores pueden crear modelos o reglas para hacer el etiquetado de instancias desconocidas a posteriori.

En la clasificación no supervisada [118], no contamos con ese conocimiento a priori, esto es, para definir los modelos o reglas tendremos que usar un conjunto de entrenamiento formado por objetos sin etiquetas.

A caballo entre ambos tipos de clasificación, nos encontramos con problemas en los que el conocimiento a priori del que disponemos está sesgado. Esto es, tenemos objetos etiquetados pero estos suelen ser pocos o no son estadísticamente representativos para todas las clases [145]. Situaciones de este estilo están presentes en áreas tan dispares como el control industrial [144], clasificación de textos [112] o en sistemas de detección de intrusos [30].

Para resolver este tipo de situaciones, en los últimos años han surgido un conjunto de nuevas técnicas, denominadas clasificadores de una clase (OCC, *One Class Classification*) [83, 93, 95, 112, 145]. Para este tipo de problemas, los objetos se pueden etiquetar como "objetivo" (*target*) o como "valores atípicos" (*outliers*). El objetivo de estos clasificadores es reconocer elementos de la clase *target* y rechazar al resto.

La Figura §4.1 representa un problema de clasificación donde durante la fase de entrenamiento disponemos de ejemplares de dos clases distintas (cuadrados y círculos). Usando este conjunto obtenemos dos tipos de frontera, la definida con línea continua y las elipses marcada con líneas discontinuas que envuelve a los ejemplos de cada una de las dos clases. El primer tipo de frontera se infiere aplicando un clasificador multiclase; el segundo tipo lo generan dos clasificadores de una clase (el primero tiene definido como *target* todo elemento etiquetado como circulo mientras que para el segundo los *target* son los objetos cuadrados). Cuando los clasificadores etiqueten nuevos elementos, se basarán en las posiciones que estos tengan respecto a las fronteras

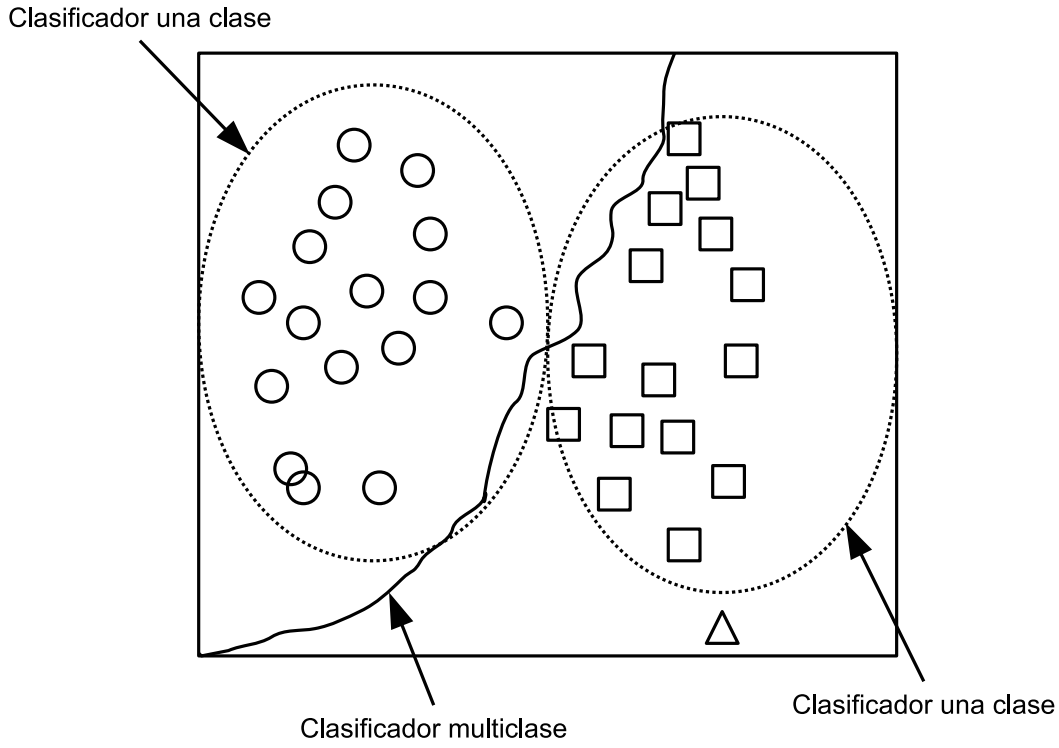


Figura 4.1: *Clasificadores de una clase versus clasificadores multiclase*

previamente calculadas. Así, el clasificador multiclase comprobaría si dicho punto está a la derecha o a la izquierda de la frontera para clarificarlo como cuadrado o círculo mientras que los clasificadores de una clase comprobarán si está dentro de alguna de las eclipses para etiquetarlo como *target* o como *outlier*. Observe que un multiclasificador siempre etiqueta el elemento con alguna de las etiquetas conocidas en el entrenamiento mientras que un clasificador de una clase no siempre lo hace. Ante la presencia de un elemento que no es círculo ni cuadrado (en este caso triángulo), el clasificador multiclase lo etiquetaría como cuadrado. Por otro lado, los dos clasificadores de una clase lo etiquetarían como *outlier* lo que nos indica que ni es cuadrado ni círculo.

Si bien el uso de los OCC es relativamente reciente, cada vez aparecen más estudios donde se demuestra su buen rendimiento [15, 96]. No obstante, todavía hay muchas líneas de investigación abiertas en este área que tratan de mejorar los resultados obtenidos hasta la fecha. Uno de los aspectos en los que más se está trabajando, es el uso de técnicas de reducción de características [49, 52, 152] que permitan mejorar y reducir los conjuntos de entrenamiento.

4.2. El problema de la clasificación

Sea Z un elemento representado mediante un vector n_f-dimensional de características $f = [f_1, f_2, ..., f_{n_f}] \mid f_i \in \mathbb{R}$ que representa un punto en $\mathbb{R}^{n_f}$. El problema de la clasificación se puede definir como la tarea de diseñar una función que asigne a Z una clase c_i, donde $c_i, i = 1, .., n_c$ denota la etiqueta asignada a la clase i. De forma genérica, al conjunto de todas las etiquetas lo denominaremos c.

Como se dijo en la sección anterior, para construir el clasificador partimos de un conjunto de entrenamiento CE formado por vectores, que representan a los elementos observados, junto con sus etiquetas de clase: $CE = \{(f_1, y_1), (f_2, y_2), ..., (f_{|CE|}, y_{|CE|})\}$ donde $y_i \in c$ representa la etiqueta de la clase asociada al objeto i.

A partir del conjunto de entrenamiento, empieza el proceso de aprendizaje del clasificador. Este proceso consiste en estimar una serie de parámetros Θ que definen al modelo de clasificación encargado de establecer las fronteras de decisión para distinguir entre los distintos tipos de ejemplos contenido en el conjunto de entrenamiento. La complejidad de un clasificador se refleja en la cantidad de parámetros que tiene que ajustar.

Una vez definidos estos parámetros, nuestro clasificador recibirá una serie de elementos no etiquetados y representados, cada uno de ellos, por un vector de características f. A cada uno de estos elementos, el clasificador le asignará una etiqueta $y \in c$ partiendo del modelo y los parámetros aprendidos durante el entrenamiento.

Para este fin, los clasificadores calculan un valor de pertenencia del vector f para cada una de las clases. Al vector f se le asignará la etiqueta de la clase para la que dicho valor es mayor:

$$\psi(\Theta, f) = \max_{i = 1, 2, ..., |c|} u_{c_i}(\Theta, f) \tag{4.1}$$

donde u es una función que determina el grado de pertenencia del vector f a la clase c_i.

Un clasificador multiclase permite tratar con un número indiferente de clases, siempre mayor que uno. Ahora bien, cualquier problema de clasificación de tres o más clases, se puede descomponer en dos o más problemas de

clasificación de dos clases [72]. Para cada uno de estos problemas binarios el clasificador se reduce a:

$$\psi(f,\Theta) = \begin{cases} c_1, & si\, u_{c_1}(f,\Theta) - u_{c_2}(f,\Theta) \geq 0 \\ c_2, & otro\, caso \end{cases} . \tag{4.2}$$

El aprendizaje de los parámetros Θ y el ajuste de la función u es un proceso iterativo que busca minimizar el error de clasificación:

$$min\ \epsilon(\psi,\Theta,CE) = \frac{1}{|CE|}\sum_{i=1}^{|CE|} \epsilon\left(\psi(f_i,\Theta), c_i\right) \tag{4.3}$$

donde $\epsilon\left(\psi(f_i,\Theta), c_i\right)$ es una función de error que refleja la capacidad del clasificador para etiquetar correctamente los elementos del conjunto de entrenamiento.

Se asume que aquellos valores de Θ que minimizan la función de error sobre el conjunto de entrenamiento, también harán que el error cometido por el clasificador al etiquetar elementos que no están en dicho conjunto también sea mínimo. A esta capacidad del clasificador de generalizar más allá del conjunto de entrenamiento se le denomina generalización.

4.3. Clasificadores de una clase

El proceso de aprendizaje de un OCC debe afrontar las siguientes restricciones:

- Sólo disponemos de elementos etiquetados como *target* en el conjunto de entrenamiento. Esta restricción puede suavizarse ya que, a veces, disponemos de algún tipo de *outlier*.
- La frontera de una clase se induce usando únicamente elementos de la clase *target*.
- La frontera que rodea a los elementos de la clase *target* debe calcularse de forma que acepte el mayor número de elementos *target* y minimice la aceptación de elementos *outliers*.

En general los clasificadores de una única clase se pueden definir como:

$$\psi(f,\Theta) = \begin{cases} 1, & si\, d(f,\Theta) < \varphi\ f\ es\ etiquetado\ como\ target \\ 0, & si\, d(f,\Theta) \geq \varphi\ f\ es\ etiquetado\ como\ outlier \end{cases} \tag{4.4}$$

donde d es una función que calcula lo similar que es f respecto al conjunto de entrenamiento, φ es un umbral y Θ son los parámetros estimados durante el proceso de aprendizaje.

Un aspecto bastante importante a destacar de los clasificadores de una clase es que el valor de φ se ajusta de forma que el error de clasificación ϵ sea algo superior a 0 para así evitar sobreaprendizaje. Esto es, si durante el proceso de entrenamiento indicamos que el error de clasificación debe ser de un 0.05, el umbral se ajustará de forma que el 95 % de los elementos del conjunto de entrenamiento se clasifiquen como *target* y el 5 % como *outliers*. No obstante, el valor concreto de φ se establece atendiendo a las características del problema de clasificación que queramos abordar. Así, si disponemos de buenos conjuntos de entrenamiento, libres de *outliers* y ruido, el valor de φ debería ser 0. En cambio, si dicho conjunto contiene *outliers* o los elementos *target* que contienen no son lo suficientemente representativos, entonces φ debe ser mayor que 0. En la Figura §4.2 se muestra la frontera definida por un clasificador de una clase cuando φ toma los valores 0.05, 0.1 y 0.2. Los elementos marcados con + son los *outliers* y los etiquetados como * son los *target*. Como vemos, el aumento del valor de φ es directamente proporcional al número de ejemplares etiquetados como targets que se quedan fuera de las fronteras inferidas por el clasificador.

4.3.1. Características de los clasificadores de una clase

A la hora de aplicar un método de clasificación u otro debemos tener en cuenta una serie de factores. Según [87, 145], las características más importantes a considerar a la hora de elegir un clasificador de una clase son:

- **Robustez**: Durante la fase de aprendizaje de los clasificadores, se suele suponer que el conjunto de entrenamiento está libre de ruido y *outliers*. Ahora bien, en algunas ocasiones pueden aparecer elementos mal etiquetados o que los valores de algunas de las variables están fuera de rango. Un clasificador es robusto, si su proceso de aprendizaje no se ve afectado por la presencia de elementos anómalos en el conjunto de entrenamiento. Observe que en el caso de los clasificadores de una clase, los conjuntos de aprendizaje suele provenir de sensores que controlan procesos reales y, por tanto, están sujetos a una gran variabilidad.

- **Incorporación de *outliers* conocidos**: En algunas ocasiones disponemos de *outliers* que introducimos dentro del conjunto de entrenamiento. El

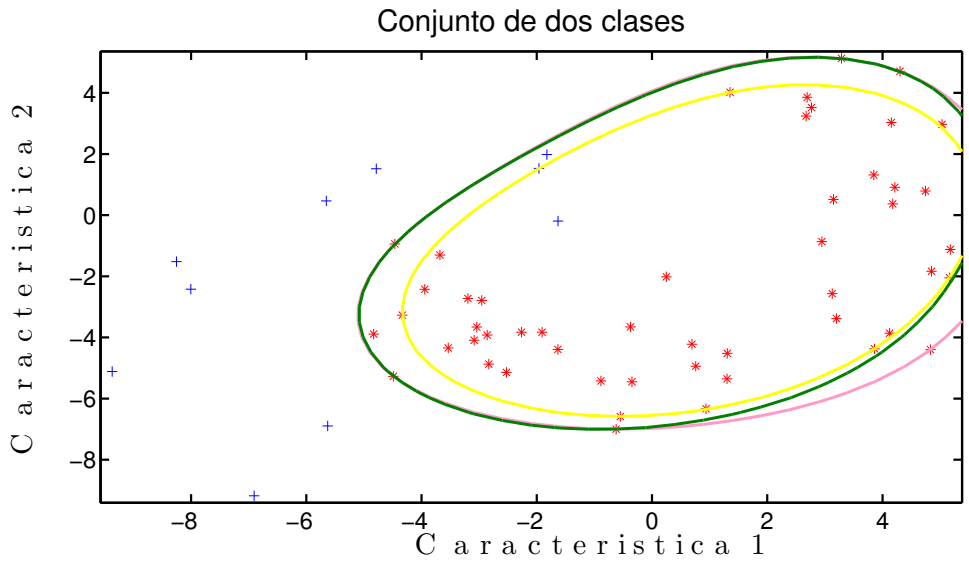


Figura 4.2: *Establecimiento de las fronteras de decisión*

objetivo de esto es, obviamente, mejorar el proceso de aprendizaje. Como veremos no todos los métodos de una clase se pueden beneficiar de estos elementos.

- **Requerimientos computacionales**: Si bien los avances en el hardware de los últimos años hacen que no sean determinantes los requisitos de cómputo y almacenamiento, todavía hay que tenerlos en cuenta. Los clasificadores de una clase están sujetos a fuertes restricciones de tipo temporal. Además, las bases de datos a clasificar suelen estar formadas con millones de datos.

- **Número de parámetros a estimar**: En el campo de la algorítmica siempre se han buscado algoritmos que sean fácilmente parametrizables. Esto es, que su comportamiento se pueda adaptar al problema modificando pocos parámetros y que éstos sean inteligibles. En general, buscamos clasificadores estables, esto es, que se comporten más o menos igual independientemente de los valores de sus parámetros.

En la Tabla §4.1 se indica el comportamiento que tienen los clasificadores que discutiremos en la Sección §4.5 para las características que acabamos de indicar. En cada casilla pueden aparecer los valores MB (muy bajo), B (bajo), A(alto), MA (muy alto).

Método	Sensible a		Requisitos computacionales		Requisitos de memoria		Número de parámetros	
	Ruido	outliers	Aprendizaje	Clasificación	Aprendizaje	Clasificación	Def. por usuario	Aprendidos
Métodos de densidad								
Histograma	B	B	MB	MB	B	B	1	n_{cubos}
Modelos de Markov	B	B	MB	MB	B	B	0	$\|s\|^2$
Gausianas	A	A	A	B	A	MB	1	$O(n_{mg} \bullet n_f^2)$
Estimador de Parzen	B	B	MB	A	A	A	1	0
Métodos frontera								
K-centros	MA	MA	A	MB	A	MB	1	K
K-vecinos más cercanos	B	A	MB	B	A	A	1	0
Máquina vector soporte	B	B	MA	MB	A	MB	1	N
Métodos reconstructivos								
k-medias, SOM	B	A	MB	MB	MB	MB	1	K
Componentes principales	A	A	B	MB	B	MB	1	$O(n_f^2)$
Redes neuronales	B	A	MA	MB	A	B	1	n_c
Reglas de asociación	B	B	MA	B	MA	B	2	$2 \bullet \|CR\|$
Series temporales	B	A	MB	MB	MB	MB	1	p

Tabla 4.1: *Características de los distintos clasificadores de una clase*

4.4. Taxonomía de los métodos OCC

Los algoritmos de una clase se han venido usando en distintas ramas del conocimiento. Como resultado de esto, conceptos como detección de outliers [89], de ruido [113], de anomalías [25], de variaciones [4], de novedades [10] o aprendizaje de conceptos [93] se usan para definir problemas equivalentes.

Con el fin de homogeneizar conceptos y técnicas, se han propuesto distintas taxonomías de clasificación [95, 114, 145]. En general, estas clasificaciones tienen en cuenta los siguientes aspectos:

1. Si las variables son consideradas independientes (univariantes o multivariantes).
2. Si presuponemos que los datos siguen una distribución previamente establecida (paramétricos o no paramétricos).
3. El campo de investigación donde se han usado (estadístico, aprendizaje automático o redes neuronales).
4. El tipo y cantidad de información con la que trabajan (no supervisados, supervisados o semi supervisados).

La taxonomía más extensa es la propuesta en [114]. En esta taxonomía (Figura §4.3) las técnicas de una clase se clasifican atendiendo a los siguientes criterios: Modelo de clasificación, (Sección §4.4.1), Tipos de datos (Sección §4.4.2) y Posibilidad de tener en cuenta relaciones temporales entre características (Sección §4.4.3).

4.4.1. Modelos de clasificación

Atendiendo a lo indicado en [145], los clasificadores de una clase se pueden dividir en tres grandes grupos: métodos de densidad, de fronteras y reconstructivos:

- **Métodos de densidad**: Se basan en estimar la función de probabilidad de los elementos etiquetados como *outliers* y los etiquetados como *target*. Estas técnicas suelen suponer que los *outliers* siguen una distribución uniforme mientras la distribución de los *target* se estima de forma distinta dependiendo del método. Cuando se tiene que etiquetar un nuevo elemento, se calcula la probabilidad de que ese elemento sea *target* y se compara con el umbral φ establecido en el entrenamiento.

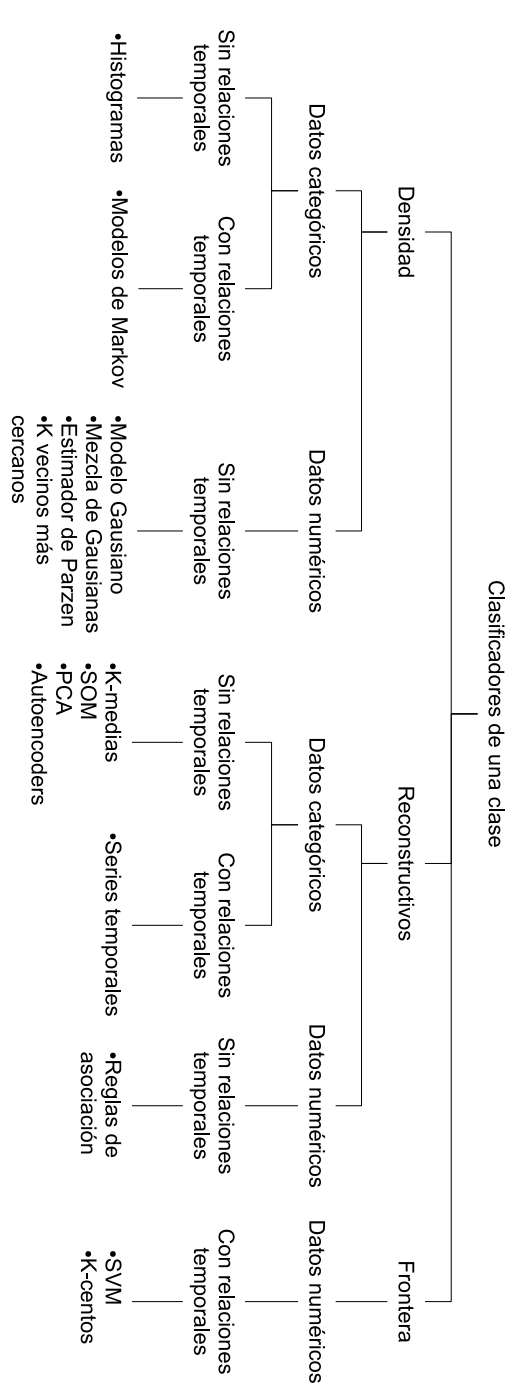


Figura 4.3: *Taxonomía de los clasificadores de una clase*

- **Métodos reconstructivos**: Estos métodos suponen que los datos siguen un modelo de generación paramétrico preestablecido. Durante el proceso de aprendizaje el clasificador deberá calcular el valor de estos parámetros. Durante el proceso de clasificación se calcula el denominado error de reconstrucción que mide si el elemento a ajustar se adapta al modelo antes calculado.

- **Métodos frontera**: Estos métodos calculan la distancia entre el elemento observado y la frontera que rodea a los elementos presentes en el conjunto de entrenamiento. Este tipo de métodos están especialmente diseñados para hacer clasificaciones de una clase.

4.4.2. Tipos de datos

Dependiendo del tipo de características, los clasificadores se pueden dividir en [87]:

- **Clasificadores que tratan con datos categóricos** (también llamados cualitativos, discretos o simbólicos): para ellos los objetos son una serie de vectores de variables cuyos valores son categorías o clases excluyentes.

- **Clasificadores que tratan con datos numéricos** (también denominados reales, continuos o cuantitativos): se emplean cuando puede establecerse algún tipo de medida de distancia entre dos elementos. En muchas ocasiones las características categóricas se transforman en numéricas por lo que se pueden aplicar este tipo de clasificadores.

4.4.3. Posibilidad de tener en cuenta relaciones temporales entre características

Otro aspecto a considerar cuando queremos hacer una clasificación son las relaciones temporales existentes entre las características de los objetos. Atendiendo a esto, podemos hablar de dos tipos de clasificadores:

- **Clasificadores que ignoran las relaciones temporales**: Estos métodos no consideran la evolución de las características a lo largo del tiempo. La clasificación se hace usando únicamente el valor actual de la variable.

- **Clasificadores que tienen en cuenta las relaciones temporales**: En este caso se analiza la evolución de los valores de las características a lo largo del tiempo.

4.5. Métodos de clasificación de una clase

En las subsecciones siguientes vamos a introducir distintos algoritmos OCC. Siguiendo la taxonomía propuesta en [145], estos algoritmos los englobaremos en tres grandes familias: métodos de densidad (Subsección §4.5.1), de frontera (Subsección §4.5.2) y los reconstructivos (Subsección §4.5.3).

4.5.1. Métodos de densidad

Estos métodos estiman la función probabilidad de densidad suponiendo que los datos siguen una distribución preestablecida. Existe distintos métodos para hacer esta estimación, entre ellos podemos destacar los histogramas (Subsección §4.5.1.1), los modelos de Markov (Subsección §4.5.1.2), las modelos gausianos (Subsección §4.5.1.3) o el estimador de Parzen (Subsección §4.5.1.4).

4.5.1.1. Histogramas

Este método se basa en realizar una representación gráfica de una variable en forma de barras, donde la superficie de cada barra es proporcional a la frecuencia de los valores representados [113]. Se puede usar tanto para variables numéricas como categóricas. En este último caso, los posibles valores que puede tomar la variable se asocian a una serie de grupos (también denominados cubos) preestablecidos.

La densidad de probabilidad se estima para cada uno de estos grupos, teniendo en cuenta los vectores del conjunto de entrenamiento que se han asociado a cada uno de ellos [11].

Estos métodos son muy robustos frente a la aparición de ruido o elementos mal etiquetados durante el proceso de entrenamiento. No obstante, necesitan unos conjuntos de entrenamiento con una gran cantidad de objetos [11, 131].

El rendimiento de este método, se ve muy influenciado por la calidad de los cubos generados. Para hacer estas particiones, se han propuesto diversos métodos como técnicas basadas en equi-profundicadad, equi-anchura o el particionado V-optimal [128].

Tanto el número de parámetros a proporcionar como los parámetros que estos algoritmos necesitan estimar depende del tipo de histograma que

se use. En general podemos decir que estas técnicas reciben como mínimo un parámetro mientras que el número de parámetros que estiman es mayor o igual al número de cubos definidos.

Los requisitos computacionales de estos métodos son prácticamente nulos ya que los histogramas se pueden ir calculando incrementalmente por lo que no es necesario almacenar en memoria todos los elementos del conjunto de entrenamiento. Una vez que tenemos construido el histograma, la estimación de la función de densidad es prácticamente inmediata.

4.5.1.2. Modelos de Markov

Las cadenas o modelos de Markov son un modelo estocástico discreto en el tiempo en el que se asume que los cambios en una variable se producen en puntos concretos de tiempo. Cada variable puede pasar por una serie de valores y estos determinan el estado global del sistema. Este tipo de técnicas se usan cuando tratamos con variables categóricas y queremos modelar eventos que se producen de forma regular.

Si $e_\tau = \{e_1, e_2, .., e_n\}$ son los posibles estados por los que puede pasar un sistema, el modelo de Markov tradicional asociado a dicho sistema se puede definir como:

- Una matriz de transición de probabilidad a:

$$a = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1|e|} \\ a_{21} & a_{22} & \cdots & a_{2|e|} \\ & & & \\ a_{|e|1} & a_{|e|2} & & a_{|e||e|} \end{bmatrix} \tag{4.5}$$

  donde $a_{ij} = P(e_j^{\tau+1}|e_j^\tau)$.

- Un vector de distribución de probabilidad $\Pi = \left[P(e_1^0), P(e_2^0), ..., P(s_{|e|}^0)\right]$ que estimada la probabilidad de que el estado inicial del sistema sea s_i.

Una vez calculado el modelo, la probabilidad para una secuencia de estados en los instantes $1, 2, ..., \tau$ se estima como:

$$p_{mm}(e_1, e_2, .., e_\tau) = \phi_{e_1} \prod_{k=2}^{\tau} a_{k-1,k}. \tag{4.6}$$

Los modelos de Markov se comportan bastante bien ante la presencia de ruido o elementos mal etiquetados en el conjunto de entrenamiento [132]. Además, el número de parámetros a estimar es igual al cuadrado de los estados del sistema. Estos modelos no requieren ningún parámetro por parte del usuario.

Para el modelo de Markov tradicional, el coste computacional del proceso de aprendizaje es prácticamente nulo. Por otro lado, el espacio de almacenamiento necesario está directamente relacionado con el número de estados por los que puede pasar un sistema, que es relativamente pequeño.

4.5.1.3. Modelo Gausiano y mezcla de gausianas

Estos métodos asume que los datos se distribuyen siguiendo una distribución gausiana o mezcla de distribuciones gausianas [10, 11]. El fundamento teórico de estos métodos se basa en el teorema central del límite [151] según el cual cuando los objetos de una clase han sido generados a partir de un prototipo al que se le han ido aplicando una serie de pequeñas perturbaciones, entonces podemos suponer que siguen una distribución gausiana.

La distribución de probabilidad para un modelo gausiano se define como:

$$p_g(x, \mu, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \times exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\} \tag{4.7}$$

donde μ es el vector medio y Σ es la matriz de covarianza.

El modelo gausiano establece grandes restricciones sobre el modelo subyacente, éste debe ser convexo y unimodal. Por lo general, los conjuntos que queremos clasificar no suelen respetar estas premisas por lo que este modelo se extiende a un modelo que establece una combinación lineal de modelos gausianos [130]:

$$p_{mg}(x) = \frac{1}{n_{mg}} \sum p_g(x, \mu, \Sigma_i)P(i) \tag{4.8}$$

donde $P(i)$ establece la importancia relativa del model gausiano i y n_{mg} es el número de distribuciones gausianas que forman el modelo.

En general, los métodos gausianos son bastante sensibles a la presencia de ruido en el conjunto de entrenamiento. Además, su rendimiento está directamente relacionado con que los datos a clasificar sigan una distribución

normal [95, 145]. El proceso de aprendizaje se puede ver negativamente afectado por la presencia de outliers, si bien en [102] se propone métodos para tratar estos casos.

El número de parámetros a estimar por estos métodos es de $n_{paramG} = n_f + \frac{1}{2}n_f(n_f - 1)$, donde n_f es la dimensionalidad del espacio, para el método gausiano y de $n_{paramMG} = n_{MG}/(n_{paramG+1})$ para la mezcla de gausianas.

Los requisitos de memoria de estos algoritmos durante el proceso de entrenamiento son bastante elevados ya que necesitamos almacenar el conjunto completo de entrenamiento, si bien es cierto que estos requerimientos se pueden reducir aplicando algún tipo de aprendizaje incremental como el descrito en [121]. En cambio, durante la clasificación, sólo es necesario almacenar los parámetros calculados durante el entrenamiento.

4.5.1.4. Estimador de Parzen

Es una extensión del método anterior donde a cada elemento del conjunto de entrenamiento se le asocia una distribución normal con media dicho punto, y todas las distribuciones con la misma varianza [124]. En general, la distribución de probabilidad estimada por este método es:

$$p_p(x) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V}\varphi(\frac{x - x_i}{h}) \tag{4.9}$$

donde $V = h^{n_f}$ es el volumen de un hipercubo n_f-dimensional con vértices de longitud h, N es la cardinalidad del conjunto de entrenamiento y $\varphi(v)$ es una función kernel que debe cumplir que $\varphi(v) > 0$ y $\int \varphi(v)dv = 1$. Una función kernel que se usa habitualmente es la distribución gausiana discutida en el apartado anterior.

Los estimadores de Parzen son robustos ante la existencia de outliers en el conjunto de aprendizaje ya que éstos sólo influyen en la densidad estimada para las regiones donde estos puntos están presentes [145]. Respecto a la robustez ante el ruido, depende directamente el valor del parámetro h.

Los requerimientos de almacenamiento son proporcionales al conjunto de elementos presentes durante la fase de entrenamiento. Estos requerimientos se ven muy menguados si reducimos el número de funciones kernel y adaptamos h al conjunto de datos.

En principio, estos métodos no tienen que estimar ningún parámetro ya que el valor h lo suele indicar el usuario. Su valor no puede ser muy

grande, ya que tendríamos unas funciones de densidad demasiado generalistas, ni muy pequeño, la función se ajustará demasiado a los valores del conjunto de entrenamiento. En [101] se propone un método para calcular h automáticamente teniendo en cuenta los valores del conjunto de entrenamiento.

4.5.2. Métodos de frontera

Estos métodos establecen una frontera alrededor de los datos que forman el conjunto de entrenamiento. La clasificación se puede realizar gracias a la definición de una función que estima la distancia existente entre un ejemplo y la frontera previamente calculada. Dentro de esta categoría de algoritmos, consideraremos los k-centros (Subseccion §4.5.2.1), los k-vecinos más cercanos (Subseccion §4.5.2.2), y las máquinas de soporte vectorial (Subseccion §4.5.2.3).

4.5.2.1. K-centros

Este método cubre el conjunto de entrenamiento con k hiperesferas todas del mismo radio [161]. Los centros de estas hiperesferas se corresponden con algunos de los puntos del conjunto de entrenamiento. Los centros y los radios de estas hiperesferas se deben calcular de forma que se minimice la función siguiente:

$$\epsilon_{KC} = \max_{i=1}^{N} \left(\min_{k=1}^{K} \|x_i - \mu_k\|^2 \right). \tag{4.10}$$

Durante el proceso de clasificación, la función de distancia que se usa es la siguiente:

$$d_{KC} = \min_{k=1}^{K} \|x_i - \mu_k\|^2. \tag{4.11}$$

Este método es muy sensible a la existencia de *outliers* o ruido en el conjunto de entrenamiento ya que la distancia de los radios se puede disparar [145].

El número de parámetros que este método recibe es de K, correspondientes a los centros de las K hiperesferas. Por otro lado, tiene que calcular otros K parámetros, los radios de cada una de estas hiperesferas. El cálculo de las distancias es relativamente rápido si bien es necesario almacenar todos los elementos del conjunto de entrenamiento.

4.5.2.2. Método del vecino más cercano

Este algoritmo [130] se diferencia del método de Parzen en que el cálculo de la función de densidad de probabilidad se calcula a partir de K observaciones y el volumen crece hasta la observación K más cercana a cada punto. La probabilidad estimada por este método se define como:

$$p(f) = \frac{K}{NV_K} \tag{4.12}$$

donde N es la cardinalidad del conjunto de entrenamiento, V_K es el volumen de la hiperesfera más pequeña con centro f y que engloba a las K observaciones más cercanas a f. A diferencia del método de Parzen, los volúmenes de las hiperesferas se van ajustando a cada zona del espacio independientemente.

Este método no tiene que estimar ningún parámetro, pero sí hay que proporcionarle el valor de K. El valor elegido, como pasaba en Parzen a la hora de definir h, puede provocar una generalización (si K es muy grande) o un sobreajuste (si K es muy pequeño). En [131] se proporciona un método automático para estimar el valor óptimo de K. Observe que la elección de este valor incide directamente sobre la robustez del método ante la aparición de ruido dentro del conjunto de entrenamiento, no así ante la existencia de *outliers* para los que este método es muy sensible.

4.5.2.3. Máquina vector soporte

A diferencia del método de los K-centros el algoritmo de SVDD intenta buscar la hiperesfera de volumen mínimo que englobe a todos los elementos del conjunto de entrenamiento [11, 145]. Durante el aprendizaje se intenta buscar el valor α que minimice la siguiente función de error

$$\epsilon_{SVDD} = \sum_i \alpha_i (f_i \bullet f_i) - \sum_{i,j} \alpha_i \alpha_j (f_i \bullet f_j) \tag{4.13}$$

donde se debe cumplir que $\sum_i \alpha_i = 1$ y $0 \leq \alpha_i \leq C$ y C determina el número de vectores que aún no han sido cubiertos por la hiperesfera.

Para dar mayor flexibilidad al modelo, el término $(f_i \bullet f_j)$ se suele sustituir por funciones kernel $K(f_i \bullet f_j)$.

Durante la fase de clasificación, la distancia entre cada elemento x y el centro de la hiperesfera a se calcula y compara con el valor del radio:

$$\gamma(x) = \begin{cases} C_1 & si\ \|f - a\|^2 \leq R^2 \\ C_2 & en\ otro\ caso \end{cases} \tag{4.14}$$

donde $a = \sum_i \alpha_i x_i$ y el radio se calcula como:

$$R = (x_k \bullet x_k) - 2\sum_i \alpha_i (f_i \bullet f_k) + \sum_{i,j} \alpha_i \alpha_j (f_i \bullet f_j) \tag{4.15}$$

donde f_k son los vectores para los que $\alpha_k = C$.

Los métodos basados en máquinas vector soporte son resistentes al ruido y a elementos mal etiquetados siempre y cuando el valor de C se ajuste adecuadamente [145]. Este parámetro es el único que debe proporcionar el usuario (salvo los propios de las funciones kernel que se quieran utilizar) mientras que los parámetros a ajustar son iguales al tamaño del conjunto de entrenamiento. La estimación de estos parámetros es un proceso costoso, con órdenes de complejidad de $O(N^3)$ [139]. A esto, se le une que es necesario almacenar en memoria todos los elementos del conjunto de trabajo para poder estimar el valor de los parámetros.

4.5.3. Métodos reconstructivos

Los últimos métodos que vamos a tratar son los reconstructivos que asumen que los datos siguen una determinada estructura. Los métodos más conocidos dentro de esta categoría son las k-medias y los mapas autoorganizativos (Sección §4.5.3.1), el análisis de componentes principales (Sección §4.5.3.2), las redes neuronales (Sección §4.5.3.3), las reglas de asociación (Sección §4.5.3.4) y los métodos basados en series temporales (Sección §4.5.3.5).

4.5.3.1. K-medias, LVQ, SOM

Estos algoritmos han sido ampliamente usados como técnicas de agrupación. En estas técnicas cada unas de las K-agrupaciones, a las que se asocian los datos, se pueden describir mediante una serie de vectores prototipo $\mu_k, k = 1, 2, .., K$ [145]. El número de agrupaciones y, por tanto, de vectores es un parámetro que el usuario debe establecer antes de ejecutar el algoritmo.

Durante el proceso de entrenamiento estas técnicas tratan de minimizar el siguiente error de reconstrucción:

$$\epsilon_{reconstruccion} = \min_k \|f - \mu_k\|^2 \tag{4.16}$$

El cálculo es los μ_k es un proceso iterativo que acaba cuando $epsilon_{reconstruccion}$ está por debajo de un umbral preestablecido.

En el caso de las k-medias, los prototipos se van ajustando de forma que se minimice el siguiente error:

$$\epsilon_{KM} = \sum_{i=1}^{N} \left(\min_{k} \ \|f - \mu_k\|^2 \right) \tag{4.17}$$

Para calcular los valores óptimos de los prototipos que minimicen este error podemos aplicar dos métodos:

- **Proceso por lotes**: Los elementos del conjunto de entrenamiento se van asociando al prototipo k más cercano (inicialmente los prototipos han sido inicializados aleatoriamente). Una vez hecho esto, el valor de cada uno de los prototipos se recalcula aplicando:

$$\mu_k = \frac{1}{|S_k|} \sum_{i \in S_k} f_i \tag{4.18}$$

  donde S_k representa a los elementos pertenecientes al conjunto de entrenamiento que tiene al prototipo k como prototipo más cercano.

- **Proceso en línea**: en este caso los valores de los prototipos se van actualizando teniendo en cuenta su valor actual y el valor del elemento que tengan a μ_k como prototipo más cercano:

$$\mu_k(\tau + 1) = \mu_k(\tau) + \eta(\tau)(f_i - \mu_k) \tag{4.19}$$

  donde $\eta(\tau)$ es una constante con valores entre 0 y 1.

En los mapas autoorganizativos (SOM) [97], la actualización de los prototipos se realiza teniendo en cuenta el valor actual de los prototipos y la distancia a los elementos más cercanos:

$$\mu_k(\tau + 1) = \mu_k(\tau) + \eta(\tau) f_{wind}\left(|f_i - \mu_k|\right)(f_i - \mu_k) \tag{4.20}$$

La función f_{wind} cuantifica la distancia existente entre un elemento y un prototipo de forma que toma el valor 1 si son iguales y valores cada vez más pequeños conforme la distancia entre ambos sea cada vez mayor [131]. Para hacer estos cálculos de vecindades, los elementos del conjunto de entrenamiento se mapean en una matriz regular de dos o tres dimensiones.

Tanto el algoritmo de las k-medias como el de los mapas autorganizativos son muy sensibles a la existencia de *outliers* y/o ruido en el conjunto de

entrenamiento ya que afectan directamente al cálculo de los prototipos. Este problema se palía si cada una de las K agrupaciones están suficientemente caracterizadas por el conjunto de entrenamiento.

Como único parámetro de entrada, ambos algoritmos necesitan que se les indique el número de prototipos, K. Las K-medias presentará un mal comportamiento si este valor de K es muy diferente al real. En cambio, SOM no es tan sensible al valor de K como a la topología de vecindad usada para definir la función f_{wind} [11].

Desde el punto de vista de los requisitos de memoria y computación, ambos algoritmos tienen unas necesidades mínimas ya que únicamente almacenan en memoria los prototipos y el valor del elemento que se esté usando para entrenar.

Por último un par de reseñas, el algoritmo de Aprendizaje de cuantificación vectorial (LVQ) [19] es una versión supervisada de las k-medias y, por tanto, se usa primordialmente en problemas de clasificación. Existen pues bastantes semejanzas entre este método y el de los k-centros discutido en la Sección §4.5.2.1, la principal diferencia radica en que, en los k-centros, los centros se localizan en los ejemplos que conforman el conjunto de entrenamiento mientras que en las k-medias pueden estar en cualquier punto del espacio. Además, en el caso de los k-centros se toma como referencia para calcular el error el elemento más lejano, mientras que en las k-medias se usa el valor medio.

4.5.3.2. Análisis de componentes principales

Para estudiar las relaciones que presentan una serie de variables correlacionadas $(f_1, f_2, ..., f_{N_f})$, se puede transformar este conjunto de variables en otro nuevo conjunto $(f_1', f_2', ..., f_{N_f}')$ de variables incorreladas entre si, denominado conjunto de componentes principales [11]. Estas nuevas variables son combinación lineal de las originales:

$$f_j' = a_{j1}f_1 + a_{j2}f_2 + ... + a_{jN_f}f_{N_f} = a_j'x \tag{4.21}$$

siendo $a_j' = (a_{1j}, a_{2j}, ..., a_{N_fj})$ un vector de constantes y f:

$$f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_{N_f} \end{bmatrix}. \tag{4.22}$$

El primer componente principal f_1' se calcula eligiendo un a_1 que maximice su varianza:

$$max\ Var(f_1') = Var(a_1'f) = a_1'\Sigma a_1 = a_1'\Sigma I a_1 = \lambda a_1' a_I = \lambda \bullet 1 = \lambda \tag{4.23}$$

donde I es la matriz identidad y λ representa al conjunto de autovalores de la matriz de covarianza Σ. Por tanto, para maximizar el valor de la varianza de f_j' hay que seleccionar el mayor autovalor λ_{max} y su correspondiente autovector a_{max}.

De modo análogo, para el segundo componente principal, elegimos el segundo mayor autovalor de la matriz Σ con su autovector asociado. Para los siguientes componentes, el proceso se repite obteniendo variables cada vez con menor varianza. Idealmente, el conjunto de componentes principales elegidos debería describir perfectamente la varianza y covarianza de la serie de datos.

Para poder construir un clasificador de una clase basado en este tipo de técnicas, calculamos el error de reconstrucción como la distancia de Mahalanobis entre el elemento y su media en el espacio transformado [95, 145]:

$$\epsilon_{reconstruccion} = \sum_{i=1}^{n_f} \frac{f_i'}{\lambda_i}. \tag{4.24}$$

El uso de este tipo de técnicas es especialmente útil cuando el número de características es alto. El único parámetro que debemos aportar al método es el número de componentes principales que queremos usar. Computacionalmente hablando, un clasificador de una clase PCA requiere pocos recursos siempre y cuando el número de variables no sea demasiado grande ya que los parámetros a calcular equivale al calculado por un método gausiano más el número de dimensiones.

4.5.3.3. Auto-codificadores y redes Diábolo

Las redes neuronales son redes de unidades de procesamiento interconectadas y organizadas en distintas capas. Los parámetros de estas unidades se ajustan durante el proceso de aprendizaje para minimizar la función de error.

Los auto-codificadores [94] y las redes diábolo son aproximaciones al aprendizaje de datos usando redes neuronales [5, 86]. La diferencia entre ambos métodos radica en el número de capas ocultas que posean. En el caso de

los auto-codificadores, sólo disponemos de una capa oculta mientras que para las redes diábolo disponemos de 3.

El error de reconstrucción se calcula como la diferencia entre la salida producida por la red y el vector de entrada x:

$$\epsilon_{recons}(x) = \|f_{auto}(x) - x\|^2 . \tag{4.25}$$

Este tipo de algoritmos son muy sensibles a la presencia de *outliers* y ruido en el conjunto de entrenamiento, aunque existen diversos métodos [11] que permiten tratar de forma eficiente con estos elementos.

El principal punto débil de estos algoritmos es la cantidad de parámetros que debe aportar el usuario. Éstos son, entre otros, la cantidad de unidades por capa o las funciones de transformación. Además, hay que estimar una serie de pesos para las capas ocultas y las unidades de entrada.

Durante el proceso de aprendizaje, la estimación de los pesos requiere de mucha memoria y mucho tiempo de computación ya que es un proceso iterativo que no finaliza hasta que se cumpla la condición de finalización. En [93] se estima que este proceso tiene una eficiencia de $O(N_w^3)$ donde N_w es el número de pesos a estimar. Durante el proceso de clasificación, estos requisitos se relajan considerablemente.

4.5.3.4. Reglas de asociación

Las reglas de asociación [127] sirven para modelar la correlación existente entre variables categóricas. Una regla de asociación es una implicación de la forma $r : r^A \rightarrow r^B$ donde r^A y r^B son predicados lógicos. Una elemento f satisface la regla r si $r^A(f) = verdad$ y $r^C(f) = verdad$.

Cada regla de asociación tiene asociado dos valores, el soporte y la confianza. El soporte de un antecedente, $sop(r_A)$, se define como la fracción de elementos del conjunto de entrenamiento que satisfacen el antecedente. El soporte de una regla, $sop(r)$, es la fracción de vectores del conjunto de entrenamiento que satisfacen la regla. La confianza de una regla, $conf(r)$, se define como el soporte de la regla entre el soporte del antecedente.

En clasificación, la confianza de una regla se usa para calcular el error de reconstrucción:

$$\epsilon_{recons}(f, RS) = \max_{m \in RS} conf(r_m) : r_m^A(f) = verdad,\ r_m^C(f) = falso \tag{4.26}$$

donde RS es el conjunto de reglas. Este error se puede interpretar como el grado de reglas que no se cumplen.

El grado de robustez de estos algoritmos se ajusta mediante los valores de confianza y soporte mínimo. Si estos valores son altos, la robustez es mayor. El usuario debe establecer estos valores mínimos. En relación con el número de parámetros a estimar, depende de la cantidad de reglas: $n_{paramRA} = 2|RS|$. Por último, recalcar que el proceso de creación de las reglas es costoso tanto en consumo de memoria como del procesador [3].

4.5.3.5. Series temporales

Las series temporales se definen como un conjunto finito y ordenado de valores que puede tomar una variable a lo largo del tiempo [129]. Por simplificar, supondremos que estas variables son unidimensionales. En este caso, el conjunto de entrenamiento está formado por una serie de elementos f_k para uno de los cuales se puede definir como un vector que contiene valores que va tomado dicha variable en los últimos $n_f - 1$ instantes de tiempo.

Los algoritmos basados en series temporales modelan la correlación entre elementos de dicha serie temporal. En la literatura [140] se han propuesto distintos modelos para representar el valor de la variable f en el instante k. Un modelo muy conocido es el autoregresivo que representa el valor de la variable como una función que se calcula a partir del pasado inmediato de los valores de la variable y un pequeño error aleatorio:

$$f_k = \sum_{i=1}^{p} \theta f_{k-i} + e_k \tag{4.27}$$

donde p establece el número de valores pasados de la variable que tendremos en cuenta, e_k es un error independiente que sigue una distribución normal y θ_i son los parámetros del modelo. Estos parámetros se estiman de forma que se minimice la suma de los errores residuales para todo el conjunto de entrenamiento. Un punto fuerte de estos algoritmos es que el usuario no tiene que establecer ningún parámetro.

Una vez que estos parámetros se han calculado, el error de reconstrucción ante un nuevo valor de la variable es:

$$\epsilon_{recons}(f, RS) = f_{k+1} - \sum_{i=1}^{p} \theta_i f_{k+1-i} \tag{4.28}$$

Estos métodos son muy robustos ante la presencia de ruido en el conjunto de entrenamiento. Sin embargo, la presencia de elementos mal etiquetados afectan negativamente al cálculo de los parámetros.

Los requisitos de memoria y procesador durante la fase de aprendizaje dependen del método para calcular los parámetros. Por otro lado, los requisitos durante la clasificación son despreciables.

4.6. Técnicas de selección de características

Los algoritmos de aprendizaje basados en instancias son muy susceptibles a características irrelevantes ya que toman sus decisiones basándose en los datos contenidos en el conjunto de entrenamiento. Por tanto, antes de aplicar alguno de estos algoritmos es necesario aplicar algún tipo de preprocesamiento, en general, y de selección o reducción de características (DR) [41] en particular.

La selección de características se puede definir como un proceso que elige un conjunto mínimo de M características de entre un conjunto original de N ($M \leq N$). Intenta que el espacio de características sea óptimamente reducido de acuerdo con un criterio de evaluación. Como en general N es muy grande, encontrar el mejor subconjunto de características es un problema **NP**-completo [118].

4.6.1. Esquema de funcionamiento

El proceso de selección de características consiste en cuatro pasos básicos (Figura §4.4):

- **Procedimiento de selección**. Se selecciona el posible subconjunto de características. Si se dispone de subconjuntos de N características este proceso debería generar, en el peor de los casos, 2^N conjuntos diferentes evaluables.
- **Función de evaluación**. Se evalúa el subconjunto de características.
- **Criterio de detención**. Se comprueba si el subconjunto cumple diversas propiedades deseables.
- **Procedimiento de validación**. Se valida la calidad del mejor subconjunto de características seleccionado, es necesario conocer el dominio de aplicación.

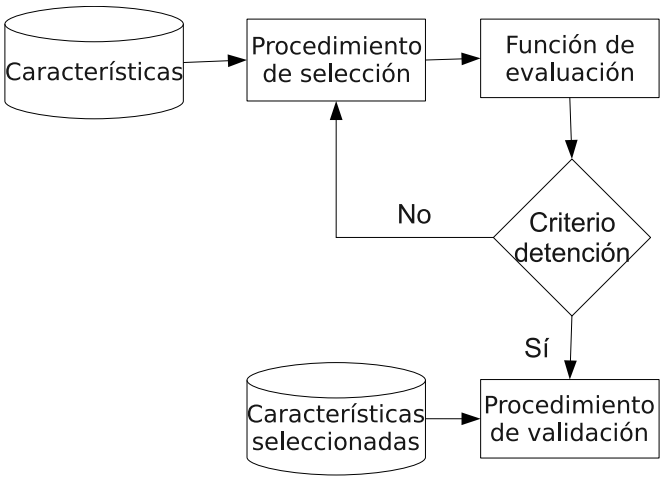


Figura 4.4: *Selección de características*

Existen gran cantidad de técnicas de selección de características como pueden ser BFF, MDLM, Focus, FSBC, Relief, CFS, Set Cover, Mitra's, ELSA, LVI, etc. Podemos consultar [81, 82] para un estudio más detallado de cada una de estas técnicas.

4.6.1.1. Selección de subconjuntos

Si atendemos al procedimiento de selección de subconjuntos, la naturaleza de este proceso viene determinado por dos elementos: el punto de comienzo de la búsqueda y la estrategia de búsqueda.

La búsqueda puede comenzar con el conjunto vacío e ir añadiendo características (**búsqueda hacia delante**), o empezar con el conjunto completo de características e ir quitando características (**búsqueda hacia atrás**). Incluso podemos partir de ambos conjuntos (o un conjunto aleatorio de características) a los que iremos quitando o añadiendo características (**búsqueda bidireccional**).

Las diversas estrategias de búsqueda nos permiten explotar el espacio de búsqueda de diferentes maneras. Si usamos una **búsqueda completa**, como *branch and bound* [119], tenemos garantizado un óptimo global respecto al criterio de evaluación usado. También podemos optar por una **búsqueda secuencial**, como son las aproximaciones basadas en técnicas *greedy* [108], reducimos el área de búsqueda y, por tanto, aumentamos el riesgo de caer en óptimos locales. Por último, tenemos la **búsqueda aleatoria**,

como *simulated annealing* [92], que va generando nuevos conjuntos de características siguiendo un patrón aleatorio y así intentan reducir el espacio de búsqueda y evitar caer en máximos locales.

4.6.1.2. Evaluación de subconjuntos

Si ahora nos fijamos en la función de evaluación, hablaremos de tres tipos de técnicas de selección de características. Los **métodos de filtrado o independientes** evalúan la bondad de una característica o conjunto de ellas basándose en las características intrínsecas del conjunto de entrenamiento. Los diversos algoritmos que se usan están basados en medidas de distancia, información, dependencia y/o consistencia.

Los **métodos de envoltura o dependientes** hacen uso de un algoritmo de minería de datos para realizar la evaluación. Se obtienen mejores resultados que en el caso anterior, pero traen consigo un costo computacional mucho mayor.

Por último, los **métodos mixtos** intentan aprovechar las ventajas de los dos modelos anteriores. Durante diferentes fases del proceso de búsqueda, explotan la forma totalmente diferente de evaluar los subconjuntos de características de los dos métodos anteriores.

Respecto a la evaluación hay que tener en cuenta que la salida generada por los distintos algoritmos varía. Esto es, algunos devuelven un conjunto mínimo de características y otros devuelven la lista completa de características ordenadas. Si disponemos de un subconjunto mínimo de características es difícil eliminar alguna de ellas ya que no disponemos de información sobre su capacidad discriminatoria individual. Este problema no aparece si el algoritmo nos devuelve un ranking de características donde podemos eliminar las más irrelevantes, aunque sí es cierto que hay que establecer un umbral de corte.

4.6.1.3. Criterios de detención

Otro aspecto importante es establecer el criterio de parada de nuestro algoritmo de selección de características. Existen criterios tan variopintos como una búsqueda completa, establecer unos límites como un número mínimo de características o de iteraciones, finalizar la búsqueda si sucesivos subconjuntos de características no mejoran lo que ya teníamos, etc.

4.6.1.4. Procedimiento de validación

Una vez seleccionado el mejor conjunto de características es necesario comprobar si dicho subconjunto permite resolver el problema de forma óptima. Hay que recordar que el criterio de evaluación (salvo para el caso de los algoritmos de envoltura) no tiene por qué ser el mismo al usado por la técnica concreta de resolución que usemos.

4.6.2. Aplicación a los clasificadores de una clase

La selección de características es especialmente importante cuando tratamos con algoritmos de una clase ya que los problemas que pretendemos resolver con este tipo de técnicas son especialmente complejos en términos de dimensionalidad. En [52, 109, 152] se detallan distintas técnicas de selección de características que se han venido aplicando en el campo de los clasificadores de una clase. Es importante resaltar, que las técnicas de selección de características aplicadas cuando trabajamos con clasificadores multiclase no son las más adecuadas para los clasificadores de una clase ya que buscamos técnicas que seleccionen las características que mejor caractericen a las clases y no las que mejor las discriminen respecto a otras [50]. En [152] se presenta una evalución de distintos métodos de selección de características especialmente diseñados para los clasificadores de una única clase. Los métodos que se comparan son:

- **Análisis de componentes principales (PCA)**: es la técnicas más usada dentro del campo de la selección de características no supervisados. Se centra en encontrar las proyecciones lineales que mejor capta la variabilidad de los datos (Seccción §4.5.3.2).
- **El algoritmo Q-α**: descubre un subconjunto característica que permitirá encontrar un particionamiento de los datos tal que éstos están bien separados de acuerdo con un criterio *graph-cut* [85].
- **Preservación local de proyecciones (LPP, *Locality Preserving Projection*)** [85]: Este algoritmo construye un grafo etiquetado que incluye información de vecindad, obtenida gracias al algoritmo de las k-medias, de los diferentes elementos que forman el conjunto de datos. Con esta información calculamos la matriz de transformación que mapea los datos a un subespacio de forma que se preserve lo más posible la información de vecindad entre objetos.

- **Score laplaciano (LS, *Laplacianan Scoring*)** [84]: Es similar a LPP pero el grafo se etiqueta teniendo en cuenta el ranking laplaciano. Una mejora de este algoritmo es el LSE (Score Laplaciano combinado con medidas basadas en la entropía) que usa distancias basadas en la entropía en vez del algoritmo de las k-medias.

  En [152] se hizo un estudio pormenorizado sobre el uso de este tipo de técnicas para clasificadores de una clase. En este estudio se desaconseja el uso de técnica basadas en PCA. Entre el resto de técnicas, no hay un ganador claro y depende en gran medida del conjunto de datos.

4.7. Resumen

En este capítulo se han introducido los conceptos básicos sobre clasificación, clasificadores de una clase y técnicas de selección de características. En las primeras secciones se justificó la necesidad de las técnicas de una clase y se compararon lo con clasificación multiclase. A continuación se enumeraron las cualidades que deben cumplir los clasificadores de una clase como, por ejemplo, la robustez o sus necesidades computacionales. Además se hizo una taxonomía de este último tipo de técnicas, que en muchos casos son parecidos a los clasificadores multiclase, basada en criterios como el modelo de verificación, el tipo de datos y el uso de relaciones temporales. Finalmente, tomando como referencia la taxonomía anterior, se ha expuesto una clasificación de las principales técnicas de clasificación de una única clase y se han tratado el problema de la selección de características aplicados a este tipo de algoritmos

Parte III

Propuesta

Capítulo 5

Marco de trabajo para construir verificadores

El prudente elabora un plan minucioso, emprende y ejecuta al detalle, ata todos los cabos y nada se le pasa desapercibido.

Sun Tzu, El arte de la guerra (siglo iv antes de Cristo)

En la literatura especializada sobre verificación, se han propuesto diversos algoritmos que enfocan este problema desde distintos puntos de vista. Aún así, comparten varios elementos que han permitido desarrollar un marco de trabajo orientado a la creación modular y sistemática de verificadores. Este capítulo se organiza de la siguiente manera: en la Sección §5.1, introduciremos y justificaremos el uso de un marco de trabajo para la creación de sistemas de verificación; la Sección §5.2 presenta el funcionamiento general de este marco de trabajo; en las Secciones §5.3, §5.4, §5.5 y §5.6, detallaremos las funciones de cada una de sus fases de desarrollo y uso del verificador; por último, en la Sección §5.7 resumiremos los contenidos vistos en el capítulo.

5.1. Introducción

En el Capítulo §2 se hizo un estudio del estado del arte en el campo de la verificación. Dicho estudio presentaba los tres principales sistemas de verificación más usados hasta la fecha [99, 105, 115].

Fruto de este estudio, se ha desarrollado un marco de trabajo [51] en el que se establecen las distintas fases por las que tiene que pasar un verificador en su proceso de construcción y uso. En este capítulo se expone este marco de trabajo detallando cada una de éstas. En cada una de ellas se han estableciendo las relaciones oportunas con los sistemas de verificación actuales y hemos añadido nuevos elementos que consideramos interesantes para construir verificadores más robustos y fiables. Además, gracias a este marco de trabajo, se han identificado los puntos débiles de los sistemas actuales (descritos en la Sección §2.2) y se han propuesto nuevas líneas de investigación (Sección §2.4).

5.2. Visión general

Los pasos que se siguen a la hora de construir un verificador se identifican en la Figura §5.1. Para explicar mejor su funcionamiento supondremos que se ha desarrollado un *wrapper*, al que denotaremos como *w*, para extraer información de precios y fechas provenientes de Ebay. Este *wrapper* necesita un verificador para comprobar que la información extraída es siempre de nuestro interés (precios y fechas).

El proceso comienza (Sección §5.3) invocando al **recolector** que realizará un **plan de recolección**, un conjunto de consultas que el *wrapper* debe ejecutar. Para nuestro ejemplo, este plan de recolección podría ser el formado por las consultas *"Dame los precios y las fechas de finalización de las pujas por el móvil X11"* y *"Dame los precios y las fechas de finalización de las pujas por la tableta T26"*.

El recolector pasa cada una de estas consultas al *wrapper* que generará una serie de conjuntos de datos que denominaremos **conjunto de trabajo**. Un ejemplo de conjunto de trabajo podría ser `{$375.99, $310.00, 1994-11-05T08:15:30}` y `{$375.99, $310.00, 'X12 Phone'}`. Éste es examinado por un experto que se encargará de asignar a cada uno de los conjuntos de datos la etiqueta de `válido` o `inválido`. Un conjunto de datos es etiquetado como `válido` si sólo contiene información

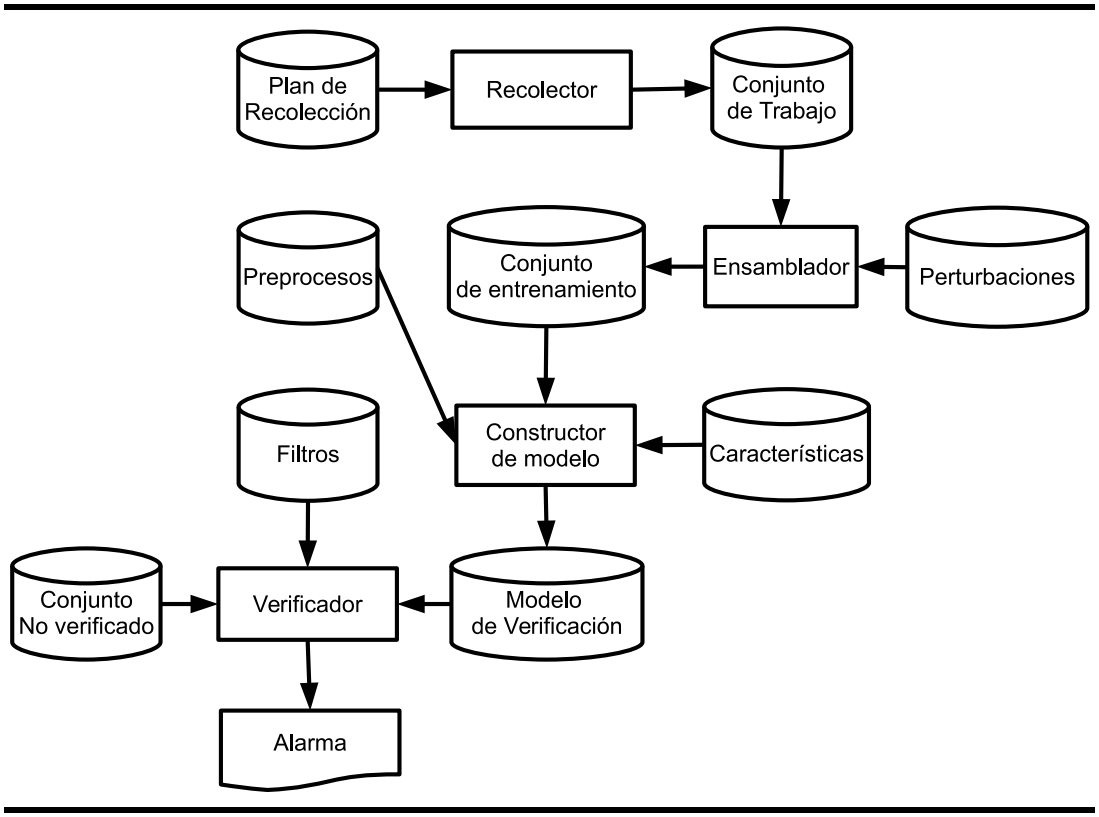


Figura 5.1: *Marco de verificación*

sobre lo que estamos interesados. En el ejemplo que nos ocupa, este experto etiquetará el conjunto `{$375.99, $310.00, 1994-11-05T08:15:30}` como `válido` y al conjunto `{$375.99, $310.00, 'X12 Phone'}` como `inválido` ya que el elemento *'X12 Phone'* es el título de un producto, date este en el que no estamos interesados.

Durante el proceso de construcción del verificador, se espera que todos los conjuntos de datos que forman el conjunto de trabajo sean etiquetados como válidos ya que, si fuera de otro modo, el proceso de construcción del verificador se pararía y se arreglaría el *wrapper* para evitar que éste extraiga información no deseada.

Para mejorar la calidad del conjunto de trabajo, el **ensamblador** ejecuta una serie de **perturbaciones** que tienen como fin generar nuevos conjuntos de datos sintéticos, algunos de los cuales serán válidos y otros inválidos (Sección §5.4). Al conjunto de trabajo al que se le añaden estos nuevos conjuntos de datos se le denomina **conjunto de entrenamiento**. Siguien-

do con el mismo ejemplo, un conjunto de entrenamiento podría ser `{$375.99, $310.00, 1994-11-05T08:15:30}`, `{$375.99, $310.00, 'X12 Phone'}`, `{375.99€, 310.00€, 11-05-1994T08:15:30}`, `{375.99€, 310.00€, 'X12 Phone'}`. Observe que las perturbaciones aplicadas han consistido en cambiar la moneda y el formado de la fecha.

Una vez que tenemos definido el conjunto de entrenamiento, el **constructor del modelo** obtiene el **modelo de verificación** que no es más que una caracterización matemática del conjunto de entrenamiento (Sección §5.5). Para ello, primero lo caracteriza, esto es, aplica una serie de **características** (Sección §5.5.1) a cada uno de los elementos que lo forman para que cada uno de ellos pase a estar representado por un vector de números. Si, por ejemplo, el conjunto de características está formado por una característica que cuenta el número de caracteres que tiene una cadena y otra que indica si la cadena incluye el signo de moneda, el conjunto de trabajo anterior pasaría a tener la forma `{[7,1], [7,1], [19,0]}, {[7,1], [7,1], [11,0]}, {[7,1], [7,1], [19,0]}, {[7,1], [7,1], [11,0]}` si asumimos que 1 es cierto y 0 falso. Además, algunas características se pueden aplicar al conjunto de datos completo como, por ejemplo, el número de elementos contenidos en dicho conjunto.

A continuación (Sección §5.5.2), el conjunto de entrenamiento ya caracterizado se pasa por una serie de **preprocesados** que tienen como objetivo la supresión de elementos o características que no contribuyan a mejorar la calidad del modelo de verificación. Esto es, se pueden eliminar elementos repetidos o aquellos que, estadísticamente, se alejen del resto. Este distanciamiento se puede deber a distintos factores como un error de etiquetado del experto.

Por último, se crea el modelo de verificación propiamente dicho teniendo en cuenta que el rango de las características es diferente y que, a la hora de verificar, no todas tienen que tener la misma importancia (Sección §5.5.3). Un ejemplo de modelo de verificación muy simple sería el que se obtendría tras aplicar la media aritmética, por característica, a los elementos de nuestro conjunto de entrenamiento. Por tanto, para nuestro ejemplo el conjunto de entrenamiento quedaría modelado como el vector `[9.66, 0.66]`.

Finalmente, cuando el **verificador** (Sección §5.6) recibe un **conjunto no verificado**, comprueba si éste sigue el modelo de verificación previamente calculado. Si no se ajusta al modelo, se aplican una serie de filtros cuyo objetivo es confirmar que el conjunto de datos es inválido. En caso de confirmarse, se lanzará una alarma para que un experto analice si dicho conjunto es o no válido.

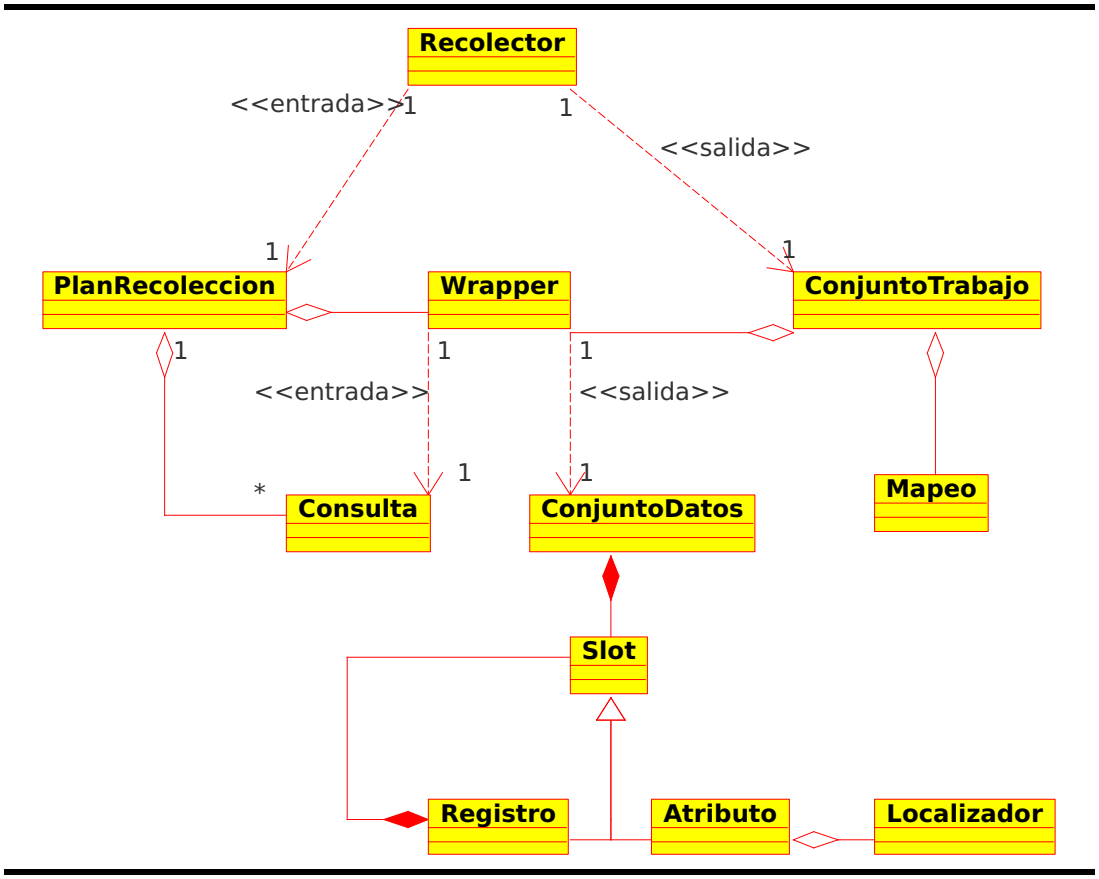


Figura 5.2: *Modelo de clases del recolector*

Una vez descrito el funcionamiento general del marco de trabajo pasamos a detallar, en los siguientes apartados, cada una de las fases.

5.3. Obtención de los conjuntos de datos

El primer paso para construir un verificador es obtener un conjunto de datos que represente, adecuadamente, aquello en lo que estoy interesado. Dentro del marco de verificación, el elemento encargado de obtener ésto es el recolector, en la Figura §5.2 se muestra el modelo de clases que forman este elemento.

El recolector usa un plan de recolección que no es más que un conjunto de consultas Q que serán ejecutadas por el *wrapper* w sobre un determinado sitio web $h \in H$ donde H representa al conjunto de los sitios web. Estos planes

son más o menos complejos dependiendo de lo dinámico que sea *h*. Esto es, si la información que alberga varía con frecuencia, el plan de recolección se transformará en la ejecución repetitiva de las consultas a lo largo de un periodo de tiempo lo suficientemente largo que nos asegura obtener valores que representen todo el universo del discurso [99]. Debe quedar claro que no existe una receta única para crear dichos planes de recolección ya que hay una gran dependencia con el sitio web y el *wrapper* usado.

Para el verificador, el *wrapper* actúa como una caja negra que va ejecutando las consultas y devolviendo una serie de **conjuntos de datos**. Un conjunto de datos, RS, es un conjunto de *slots*, s_i, donde el término *slot* se refiere a un atributo o a un conjunto de ellos:

$$RS = \{s_1, s_2, \ldots s_{|RS|}\} \tag{5.1}$$

Cada *slot* se representa como un par de valores:

$$s_i = (a_i, l_i) \mid l_i \in \mathcal{L},\ i = 1..|RS| \tag{5.2}$$

donde a_i es el atributo -un dato a cuyo valor concreto accedemos gracias a un **localizador**- asociado al *slot* i, l_i es la etiqueta asignada al *slot* i y denota su rol, y $\mathcal{L}$ es el conjunto de etiquetas de los roles. Por ejemplo, los valores de los títulos y precios de los libros que aparecen en la web de Amazon serían atributos, los XPointer de estos elementos serían los localizadores mientras que el conjunto de etiquetas asociadas a las clases, $\mathcal{L}$, sería `Título` y `Precio`.

Una vez obtenidos los conjuntos de datos, el recolector crea el **conjunto de trabajo**, definido como:

$$WS = \{(w, q_1, RS_1), (w, q_2, RS_2), \ldots, (w, q_{|WS|}, RS_{|WS|})\} \tag{5.3}$$

donde w es el *wrapper*, $q_i \in Q$ y RS_i es el conjunto de datos obtenidos por el *wrapper* w al ejecutar la consulta q_i.

5.4. Creación de los conjuntos de entrenamiento

Después de construir el conjunto de trabajo, un experto se encarga de etiquetar cada uno de los conjuntos de datos que lo forman como `válido` o `inválido`. Un conjunto de datos será etiquetado como `válido` si todos los *slots* que lo forman tienen asignadas las clases correctas (por ejemplo, todos los títulos están etiquetados como títulos y no hay precios etiquetados como fechas). Si algún *slot* no se etiquetó bien, entonces el conjunto de datos es inválido.

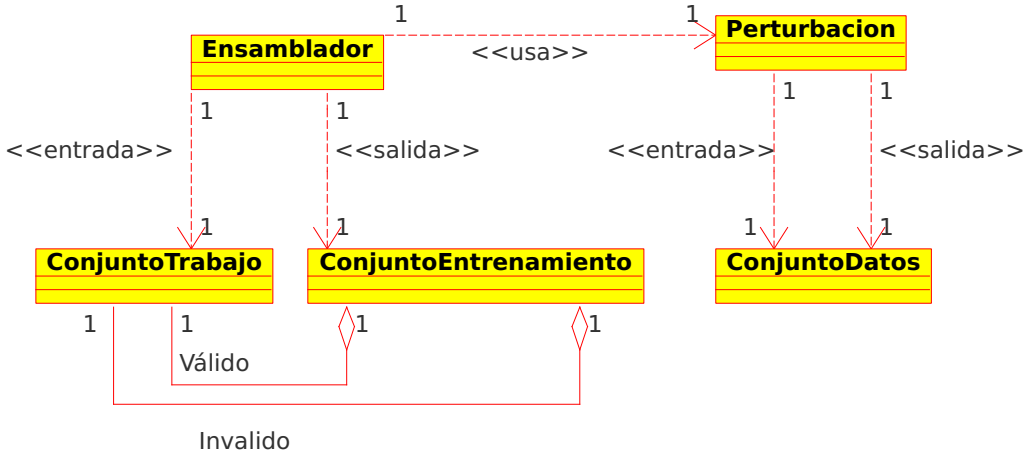


Figura 5.3: *Modelo de clases del ensamblador*

Al conjunto formado por estos conjuntos de datos etiquetados se le denomina conjunto de entrenamiento. Formalmente, definimos un conjunto de entrenamiento como:

$$TS = \left\{(w, q_1, RS_1, \delta_1), (w, q_2, RS_2, \delta_2), \ldots, (w, q_{|TS|}, RS_{|TS|}, \delta_{|TS|})\right\} \tag{5.4}$$

donde $\delta_i = 1$ si el conjunto de datos RS_i es válido y $\delta_i = 0$ en otro caso.

En otras palabras, el conjunto de entrenamiento también lo podemos ver con una tupla formado por dos conjuntos:

$$TS = \{TS_-, TS_+\} \tag{5.5}$$

donde TS_+ representa al conjunto de trabajo formado por conjuntos de datos válidos:

$$TS_+ = \{(w, q_i, RS_i)\} \mid (w, q_i, RS_i, \delta_i) \in TS \wedge \delta_i = 1 \tag{5.6}$$

y TS_- representa al conjunto de trabajo formado por conjuntos de datos inválidos:

$$TS_- = \{(w, q_i, RS_i)\} \mid (w, q_i, RS_i, \delta_i) \in TS \wedge \delta_i = 0. \tag{5.7}$$

Por lo general, todos los conjuntos de datos devueltos por el *wrapper* serán etiquetados como válidos, con lo cual, $TS_- = \emptyset$. Esto se debe a que durante el proceso de construcción del verificador, cualquier conjunto de datos etiquetado como `inválido` implica un mal funcionamiento del *wrapper*

por lo que éste será modificado para evitar que cometa estos errores. Es más, en algunos casos el plan de recolección está mal diseñado y puede dar lugar a conjuntos de trabajo todos válidos pero muy parecidos entre si, $|TS_+| \simeq 0$. Por tanto, estaríamos intentando diseñar un verificador usando datos que no se corresponden a lo que éste puede encontrar en un futuro dando lugar así a problemas de sobreaprendizaje [118].

El objetivo del ensamblador, cuyo diagrama de clases se puede consultar en la Figura §5.3, es generar un buen conjunto de entrenamiento aplicando una serie de perturbaciones $\mathcal{P}$ a los conjuntos de trabajo devueltos por el recolector:

$$TS = TS \cup p(TS) \; \forall p \in \mathcal{P} \tag{5.8}$$

Dentro del mundo de la verificación, las perturbaciones más conocidas son las propuestas por McCann y otros [115] (Consulte Sección §2.3): cambiar el interfaz de la consulta, cambiar la fuente de datos, cambiar el formato de presentación y uso de múltiples fuentes. El problema que presenta su uso es que, los de datos generados (ya sean válidos o inválidos), son demasiado sintéticos por lo que el verificador no acaba de beneficiarse de ellos.

Otro tipo de perturbaciones son las propuestas en [145] orientadas al campo de la clasificación. En este caso las perturbaciones se aplican directamente a la caracterización de los *slots*, Sección §5.5. Los métodos propuestos por Tax [145] tienen, como principal desventaja, que, para grandes dimensionalidades, son ineficientes.

Independientemente del tipo de caracterización que se use, todos los conjuntos generados deben ser examinados por un experto para que verifique sus etiquetas.

5.5. Creación de los modelos de verificación

Un modelo de verificación es un formulismo matemático que nos permite expresar las relaciones existentes entre los distintos *slots* que conforman el conjunto de entrenamiento. Su objetivo es facilitar el estudio del grado de similitud entre nuevos *slots* y los usados para el modelo de verificación.

A la hora de construirlos hay que tener en cuenta tres grandes aspectos:

1. **Caracterización**: El conjunto de entrenamiento está compuesto por una serie de cadenas de texto que son de difícil procesamiento por modelos matemáticos. En esta primera fase, tanto los conjuntos de datos como cada uno de los *slots* se convertirán en vectores de números (Sección §5.5.1).

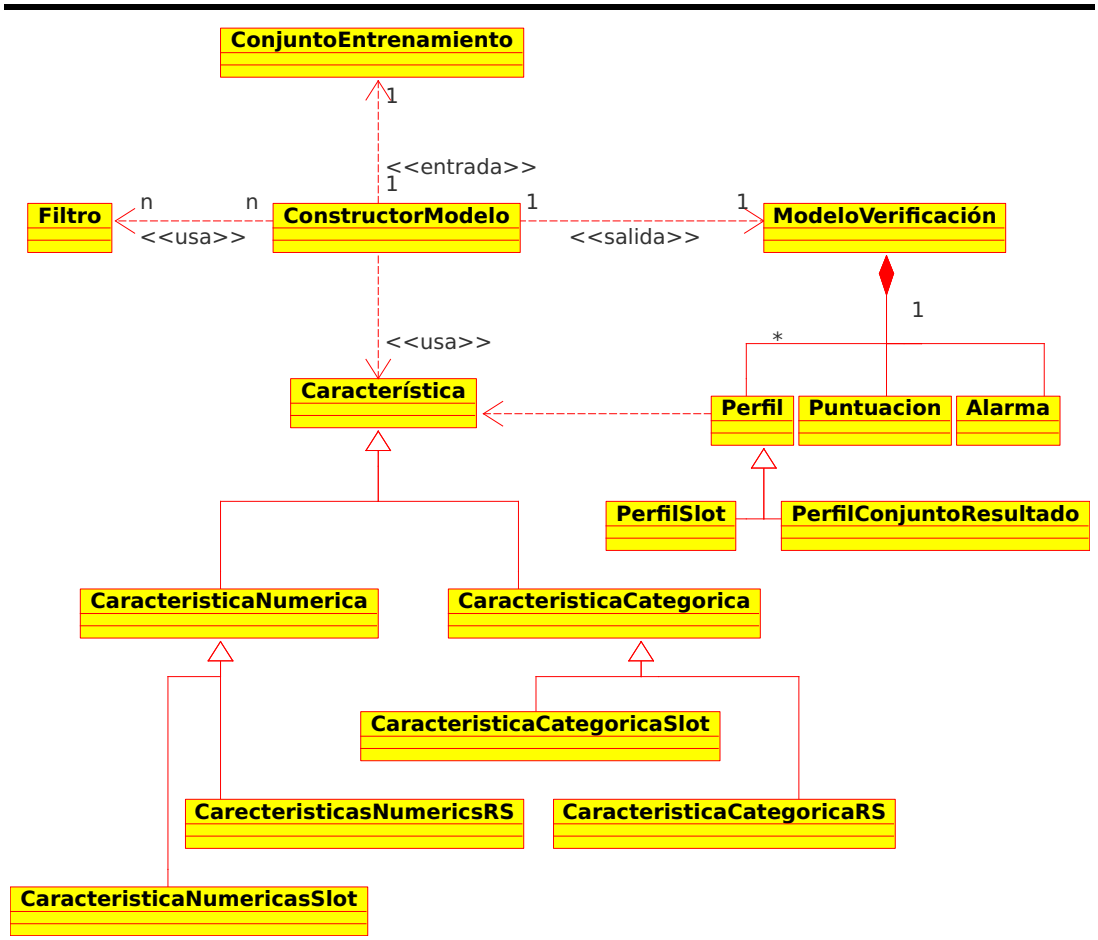


Figura 5.4: *Modelo de clases del verificación*

2. **Preprocesado**: Después de caracterizar al conjunto, se aplican una serie de procesos $\mathcal{M}$ orientados a normalizar los rangos de las distintas características y disminuir tanto la cardinalidad como la dimensionalidad del conjunto de entrenamiento (Sección §5.5.2).

3. **Perfilado y puntuación**: El perfilado busca modelos que intenten explicar los valores logrados por cada una de las características. Durante la fase de puntuación, se ponderará la importancia relativa de cada una de los valores obtenidos tras el perfilado y se combinarán con el objeto de obtener un único valor que permita dilucidar si un nuevo elemento es o no válido (Sección §5.5.3).

En los Algoritmo §5.1 y §5.2 se detallan los pasos que hay que aplicar pa-

```
CONSTRUYEVERIFICADOR(w, RP, P, F, L, M)
1  WS = RECOLECTOR(w, RP)
2  TS = EXPERTO(WS)
3  TS = ENSAMBLADOR(TS, P)
4  return CONSTRUIRMODELO(TS, L, F, M)

RECOLECTOR(w, RP)
1  ⟨w, h, Q⟩ = RP
2  WS = {}
3  for each q ∈ Q
4       RS = w(h, q)
5       WS = WS ∪ {(w, q, RS)};
6  return WS

EXPERTO(WS)
1  for each tupla ∈ WS
2       ⟨w, q, RS⟩ = tupla
3       // El valor de la etiqueta lo decide un experto
4       δ = DecisionExperto
5       TS = TS ∪ {(w, q, RS, δ)};
6  return TS

ENSAMBLADOR(TS, P)
1  TS' = {}
2  for each p ∈ P
3       TS' = TS' ∪ p(TS)
4  return TS ∪ TS'
```

Programa 5.1: *Algoritmo para la construcción de un verificador*

ra construir un modelo de verificación. La función CONSTRUYEVERIFICADOR recibe como argumentos el *wrapper* w, el plan de recolección RP que ejecutará el *wrapper*, el conjunto de perturbaciones $\mathcal{P}$ a aplicar, las características $\mathcal{F}$ con la que se caracterizarán los slots, el conjunto de posibles etiquetas $\mathcal{L}$ y el conjunto de actividades de preprocesamiento $\mathcal{M}$.

La invocación a la función RECOLECTOR tiene como objetivo la obtención de conjuntos de datos (Sección §5.3) mientras que las funciones EXPERTO y ENSAMBLADOR permiten obtener los conjuntos de entrenamiento (Sección §5.4). En las secciones siguientes estudiaremos detenidamente los distintos elementos de la función CONSTRUIRMODELO: CARACTERIZACIÓN en la Sección §5.5.1, PREPROCESOS en la Sección §5.5.2,

```
CONSTRUIRMODELO(TS, 𝓛, 𝓕, 𝓜)
1  X_TS = CARACTERIZACION(TS, 𝓕)
2  X_TS = PREPROCESOS(X_TS, 𝓜)
3  perfilado = CONSTRUIRPERFILADO(X_TS)
4  puntuacion = CONSTRUIRPUNTUACION(perfilado)
5  alarma = CONSTRUIRALARMA(perfilado, X_TS)
6  return ⟨perfilado, puntuacion, alarma⟩
CARACTERIZACION(TS, 𝓕)
1  for each RS ∈ TS
2      for each f ∈ 𝓕_RS
3          Y = Y ∪ f(RS)
4      for cada s ∈ RS
5          for cada f ∈ 𝓕_s
6              x = x ∪ f(s)
7          X_RS = X_RS ∪ x
8      X_TS = X_TS ∪ ⟨Y, X_RS⟩
9  return X_TS
CONSTRUIRPERFILADO(TS, X_TS, 𝓕)
1  for each X_RS ∈ X_TS
2      ⟨Y, X⟩ = X_RS
3      Y_all = Y_all ∪ Y
4      for each X_s ∈ X
5          X_all = X_all ∪ X_s
6  // el perfilado se hace por característica
7  for i = 1..columnas_de(Y_all) perfilado_Yi = calculaPerfilado(Y_*,i)
8  for i = 1..columnas_de(X_all) perfilado_Xi = calculaPerfilado(X_*,i)
9  return perfilado_Y ∪ perfilado_X
```

Programa 5.2: *Algoritmo para la construcción de un modelo de verificación*

CONSTRUIRPERFILADO y CONSTRUIRPUNTUACION en la Sección §5.5.3.

5.5.1. Caracterización

La primera fase dentro del proceso de construcción del modelo de verificación es representar cada conjunto de datos y cada *slot* contenido en el conjunto de entrenamiento como un vector (ver Figura §5.5). De esta labor se encarga la función abstracta CARACTERIZACION. Para obtenerlo aplicamos una serie de características $\mathcal{F}$ que son un rasgo cuantificable o bien de un

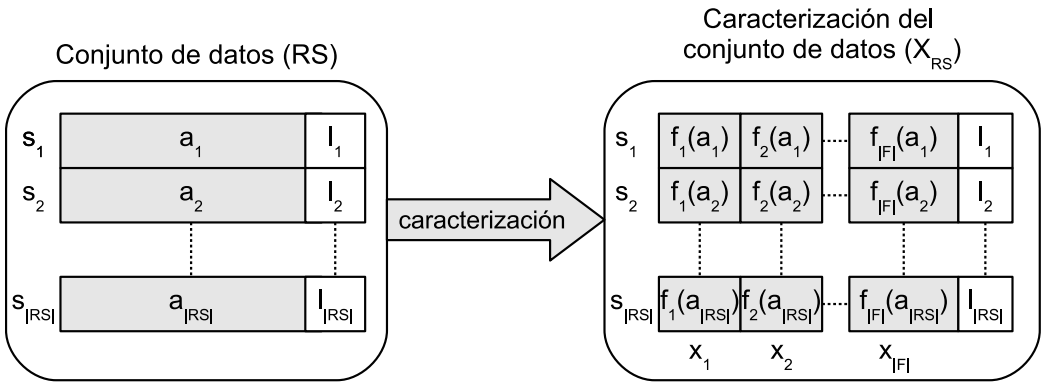


Figura 5.5: *Proceso de caracterización*

slot o del conjunto de datos. Los valores de cada una de estas características se usarán para obtener evidencias de la validez o no de los datos. El conjunto de todas las características se define como:

$$\mathcal{F} = \mathcal{N} \cup \mathcal{C} \tag{5.9}$$

donde $\mathcal{N}$ es el conjunto de todas las características numéricas (Sección §5.5.1.1) y $\mathcal{C}$ es el conjunto de todas las características categóricas (Sección §5.5.1.2). Además, y atendiendo a su ámbito de aplicación, $\mathcal{F}$ se puede dividir en $\mathcal{F}_{RS}$ y $\mathcal{F}_s$ que están formados, respectivamente, por las características aplicables a conjuntos de datos y a *slots*.

Formalmente, la caracterización de un conjunto de entrenamiento TS se define como:

$$X_{TS} = \{(w, q_1, X_{RS1}, \delta_1), (w, q_2, X_{RS2}, \delta_2), ...(w, q_{|RS|}, X_{|RS|}, \delta_{|RS|})\} \tag{5.10}$$

donde

$$X_{RS} = \{Y, [(X_{s1}, l_1), (X_{s2}, l_2), \ldots, (X_{s|X_{RS}|}, l_{|X_{RS}|})]\} \tag{5.11}$$

siendo $X_{s_i} = [x_1, x_2, ..., x_{|\mathcal{F}_s|}] = [f_{map(1,\mathcal{F}_s)}(a_i), f_{map(2,\mathcal{F}_s)}(a_i), ..., f_{map(|\mathcal{F}_s|,\mathcal{F}_s)}(a_i)]$ donde $f_i \in \mathcal{F}_S$, map es una función de mapeo que indica cuál es la i-ésima característica que se puede aplicar a un *slot* e $Y = [y_1, y_2, ..., y_{|\mathcal{F}_{RS}|}] = [f_{map(1,\mathcal{F}_{RS})}(RS), f_{map(2,\mathcal{F}_{RS})}(RS), ..., f_{map(|\mathcal{F}_{RS}|,\mathcal{F}_{RS})}(RS)]$ donde $f_i \in \mathcal{F}_{RS}$, map es una función de mapeo que indica cuál es la i-ésima característica que se puede aplicar a un conjunto de datos.

Observe que en esta definición estamos suponiendo que los conjuntos de trabajo y todos los *slots* se caracterizan con el mismo número de características. Esto es una generalización ya que podemos encontrarnos con características que no se puedan aplicar atendiendo a la semántica propia de un conjunto de trabajo o un *slot*.

5.5.1.1. Características numéricas

Las características numéricas $\mathcal{N}$ son aquellas que transforman un *slot* o un conjunto de datos en números reales. Este tipo de características se pueden englobar en las siguientes familias:

1. **Conteo**: Dentro de esta categoría tenemos las características que cuentan el número de *slots* de una determinada clase que hay en un conjunto de datos [99, 100, 105] o cuentan el número de registros que satisfacen algún tipo de restricción [133]. También dentro de esta categoría, encontramos aquellas que cuentan los atributos que empiezan y acaban con un determinado patrón [105]. Otros autores proponen como funciones de conteo aquellas que cuentan el número de dígitos, letras, n-gramas, etc. que forman un atributo.

2. **Léxicas**: Estas características se centran en estudiar rasgos relacionados con el juego de caracteres que usan los atributos de una determinada clase. Dentro de esta categoría podemos incluir características como: longitud de un atributo, número de palabras contenidas en un atributo, longitud media de estas palabras, densidad de dígitos, densidad de letras, densidad de letras en mayúscula, densidad de letras en minúscula, densidad de símbolos de puntuación o densidad de etiquetas HTML [99, 100]. Otras características a considerar dentro de esta categoría serían aquellas que determinan si un atributo comienza con una letra mayúscula y acaba con un símbolo de puntuación [159], o las que calculan la proporción de números, números en coma flotante o palabras en mayúscula [158]. En [159] se proponen una serie de características lógicas, como determinar si el atributo está justo antes de un precio, cuyo objetivo es caracterizar el contexto de los atributos. Finalmente, en [120] se proponen 42 características más pero no se describen con la profundidad necesaria como para implementarlas.

3. **Similaridad**: El objetivo de estas funciones es determinar cuan similar es un atributo respecto a un conjunto de atributos de su misma clase. Por ejemplo, la presencia de las cadenas "NH" o "TRYP" dentro de un

atributo nos puede llevar a la conclusión de que estas cadenas representan nombres de hoteles. Así pues, un atributo formado por palabras que no se asemejen lo suficiente a las que típicamente nos encontramos en atributos válidos de la misma clase, pueden indicarnos que dicho atributo es inválido. Para calcular cuan similar es un atributo sin verificar a respecto al conjunto de atributos verificados A aplicamos:

$$max_{b \in A} SIM(a, b) \tag{5.12}$$

donde $SIM(a, b)$ es una función de similitud.

En [159] se propone la función de similitud coseno introducida en la teoría del modelo espacio vectorial del texto [136]. En este modelo los atributos se representan como vectores $\vec{a} \in R^{|W|}$ donde W es el conjunto formado por las raíces de las palabras obtenidas de un conjunto representativo de atributos de la misma clase. Cada elemento de $\vec{a}$ representa a cada una de las palabras w y su valor se calcula de la siguiente manera:

$$a_w = \begin{cases} 0 & si\, w \notin a \\ (1 + \log TF_{a,w}) \cdot \log IDF_w & en\, otro\, caso \end{cases} \tag{5.13}$$

donde TF_a, w denota el número de veces que la palabra w aparece dentro del atributo a y IDF_w es la fracción de atributos que contienen la palabra w respecto al número total de atributos.

Más tarde en [158] propusieron una nueva función de similitud que se calcula a partir de la distancia de edición[†1] entre palabras ya normalizada a nivel de atributo. Es esta nueva función los costes de inserción y borrado valen 0, y el coste de reemplazo es igual a la distancia de edición a nivel de caracteres entre las palabras a intercambiar.

Además, existen funciones de similitud que permiten comparar registros desde un punto de vista estructural. Para nosotros, un registro se puede ver como un árbol ordenado y etiquetado. Por tanto, la similitud entre dos registros se puede calcular usando medidas de edición de árboles, en general, o los algoritmos de distancias de alineación [8, 28]. Como pasaba en el caso de Wong y Lam [158], hay que definir un coste a cada una de las operaciones a aplicar. En [6] se propone un método automático para calcular estos costes.

[†1] También denominada Distancia de Levenshtein, es el número mínimo de operaciones requeridas para transformar una cadena de caracteres en otra

Si los registros se representan mediante XML y está disponible su definición DTD, entonces podemos aplicar otras medidas de similitud más específicas ([22, 137, 146]).

4. **Entorno**: Tienen en cuenta la disposición de los atributos dentro de una página HTML. Así si sabemos que los *slots* correspondientes a los autores de un libro están muy cerca de los *slots* pertenecientes al libro, si detectamos un registro etiquetado como `Libro` que no tiene cerca *slots* etiquetados como `Autor` entonces puede ser que el *slot* Libro esté mal etiquetado. En [115] se propone la característica denominada ordenación de tuplas que tienen en cuenta la separación de los registros en un documento HTML. Si dos registros se superponen entonces son iguales, en otro caso son distintos.

   En [42] se presentan muchas características de este tipo, todas basadas en la siguiente definición: dado un conjunto de datos formado por una serie de registros $r_1, r_2, ..., r_n$, cada uno de estos registros se alinea mediante una lista que contiene todos sus atributos $\langle a_{i,0}, a_{i,1}, ..., a_{i,n}\rangle$ $(1 \leq i \leq m, n_i \geq 0)$; también asumen que el localizador de cada atributo está formado por dos índices (a^s y a^e) que representan el comienzo y fin del atributo dentro del documento del que fueron extraídos. Los autores suponen que $a^s_{i,0} < a^s_{i,1} < ... < a^s_{i,n_i}$ para cada atributo siendo $a^s_{i,0}$ el atributo raíz. Para cada uno de estos atributos raíz, de Oliveira y da Silva [42] definen las siguientes medidas:

   - Localización de la raíz: $a^s_{i,j} - a^s_{i,0}$
   - Localización absoluta de la raíz: $a^e_{i,j} - a^s_{i,0}$
   - Localización relativa de la raíz: $a^s_{i,j} - a^s_{i,0} - \sum_{k=1}^{j-1}(a^e_{i,k} - a^s_{i,k})$
   - Localización previa: $a^s_{i,j} - a^s_{i,j-1} - \sum_{k=1}^{j-1}(a^e_{i,k} - a^s_{i,k})$
   - Localización absoluta previa: $a^e_{i,j} - a^s_{i,j-1}$
   - Longitud de la localización: $a^e_{i,j} - a^s_{i,j} - (a^e_{i,j-1} - a^s_{i,j-1})$
   - Localización hermana: $a^s_{i,j} - a^s_{i-1}$
   - Localización hermana absoluta: $a^e_{i,j} - a^s_{i-1,j}$
   - Estructura: $a^s_{i,j} - a^s_{i-1,j} - \sum_{j=1}^{n_i}(a^e_{i-1,j} - a^s_{i-1,j})$

   En la definición de todas estas medidas se presupone que la alineación de los registros es comparable y que todos los registros provienen del mismo documento y que los localizadores no tienen en cuenta la resolución de la pantalla.

5. **Tendencia**: estas características sirven para estudiar las fluctuaciones de los valores obtenidas por otras características numéricas [115]. En cambio, no considera la posibilidad de comprobar las fluctuaciones de los valores de los propios atributos. Por ejemplo, si detectamos que la variación del precio de un libro en las últimos días es de más de un 25 % puede ser que estemos ante datos erróneos.

5.5.1.2. Características categóricas

Las características categóricas, $\mathcal{C}$, engloban tanto a patrones que describen la estructura de un registro, como restricciones matemáticas de los valores que puede tener un atributo o relaciones entre ellos. Por lo general, estas características se suelen usar como base para características numéricas como, pueden ser, las de conteo. A continuación, describimos algunas de las características más usadas:

1. **Estructura del registro**: Se centran en analizar los registros desde un punto de vista estructural. Por ejemplo, si se extraen registros que representan a libros y sabemos que todo libro tiene un nombre y un precio, si extraigo un registro etiquetado como `Libro` que no contenga precio me puede indicar que dicho registro ha sido mal etiquetado. Propuestas concretas de estas características las encontramos en [115], donde se usa una característica que considera el orden de los atributos dentro de un registro, y en [100] que usa una característica que comprueba el número de atributos de cada clase que hay en un registro.

2. **Clase lingüística**: Intentan buscar la clase linguística (nombre, verbo, ...) que caracteriza a un atributo [120]. Por ejemplo, del análisis de todos los atributos etiquetados con la clase `Título` se puede llegar a la conclusión que todos son nombres. Este tipo de características se suelen construir usando software del tipo *part-of-sppech taggers* [117] que son capaces de leer textos en lenguaje natural, dividirlos e ir asignado etiquetas a trozos de texto. Para el desarrollo de este tipo de software se usan los Modelos de Markov discutidos en la Sección §4.5.1.2. Dentro de estas características también se suelen usar diccionarios que sirven para localizar nombres de ciudades, ríos, etc.

3. **Expresiones regulares**: Estas características buscan expresiones regulares que cumplan todos los atributos de una clase [120] o bien que cumplan los atributos parcialmente [105]. Ejemplos del primer tipo de características lo tendríamos en aquellas que comprueban si una cadena

representa una URL o un número de cuenta bancaria. Una característica que compruebe si una cadena empieza por '<a>' y termina por '' sería un ejemplo del segundo tipo.

En [20] se propone el algoritmo Allergia que es capaz de generar una gramática regular usando los valores de un conjunto de atributos. El principal problema que presenta este algoritmo, así como todas sus variantes [16, 21, 77, 135, 147] es que tiende a generar expresiones demasiado generales. El algoritmo DATAPROG desarrollado en [105] palía este hecho intentando caracterizar cada atributo mediante una secuencia de tokens que describen el comienzo y fin de cada atributo. El algoritmo DATAPROG tiende a generar muchas reglas y muy específicas, hecho este que ha sido resuelto en [13, 150]. No obstante, el principal obstáculo para poder aplicar el DATAPROG es que hay que definir previamente los *tokens* a usar y estos dependen del dominio del sitio web.

4. **Restricciones semánticas**: Estas características se centran en determinar si ciertas restriciones son verificadas por los atributos de una determinada clase. Por ejemplo, el porcentaje de descuento de un producto siempre debería ser un valor entre 0 y 100.

   Ernst y otros [66, 67] propusieron DAYCON, un sistema automático para la obtención de restricciones semánticas. Desgraciadamente, este algoritmo no es directamente aplicable a los datos obtenidos de la web ya que no soporta trabajar con conjuntos expuestos al ruido. En este caso tendremos que usar las modificaciones hechas en [133] para este algoritmo. Dentro de esta categoría también podemos usar las técnicas de reglas de asociación discutidas en la Sección §4.5.3.4, en concreto en [88] se usan este tipo de técnicas para hacer limpieza de datos (*data cleansing*) mientras que en [103] aplican lógica de primer orden para establecer restricciones semánticas.

5.5.2. Preprocesado

Una vez caracterizado el conjunto de entrenamiento, es necesario simplificarlo para así poder obtener modelos robustos y simples. Para ello, la función PREPROCESOS aplica una serie de procesos $\mathcal{M}$ que, de alguna forma, "saneen" el contenido de la caracterización conjunto de entrenamiento X_{TS}. Estos procesos los podemos agrupar en cuatro grandes familias:

1. **Normalización de dominios**: Estos procesos se encargan de igualar los dominios de los valores devueltos por las diferentes características para hacer posible su comparación. Aquí se pueden aplicar métodos como

el descrito en [141] donde los dominios de todas las variables pasan a ser muestras de una distribución de varianza 1 y media 0.

2. **Reducción de dimensionalidad**: Este tipo de técnicas ya fueron descritas en la Sección §4.6 y su objetivo es eliminar aquellas características que no vayan a contribuir positivamente en la construcción del modelo.

3. **Reducción de instancias**: Estos filtros reducen la cardinalidad del conjunto de entrenamiento manteniendo la calidad del mismo [123, 156]. Formalmente, el resultado de aplicar este tipo de filtros es un subconjunto de instancias seleccionadas $S \subseteq TS$ cuyo objetivo es mejorar el comportamiento del verificador. En la literatura existen multitud de propuestas [1, 79, 156] pero no hay ninguna específica para tratar con datos obtenidos en el entorno web.

4. **Detección de elementos anómalos**: Si bien el conjunto de trabajo ha pasado el filtro de un experto humano, también es cierto que este se ha podido confundir a la hora de clasificar los elementos del conjunto de trabajo. En el campo de la verificación, los conjunto de trabajo suelen ser muy grandes de aquí que debemos tener en cuenta este extremo a la hora de construir el modelo de verificación. Los filtros de detección de elementos anómalos tienen como objetivo detectar este tipo de elementos mal etiquetados de forma que o bien se avise al experto para que confirme la etiqueta o bien se elimine del conjunto de entrenamiento. Técnicas como las basadas en el test χ^2 [54], el de Dixon [134] o el de Grubbs [80] nos permiten hacer esta labor.

5.5.3. Perfilado y puntuación

Tras las dos fases anteriores, el conjunto de entrenamiento estará representado por una serie de vectores n dimensionales. Para poder comparar si un vector se parece a los ya calculados hay que perfilar cada una de las características, lo que implica buscar una función o modelo matemático que resuma los valores de cada característica. De estas acciones se encarga la función CONSTRUIRPERFILADO.

A continuación es necesario crear una puntuación (*score*) que nos permita obtener un único valor que combine la información dada por todas las características aplicadas. La función CONSTRUIRPUNTUACION crea esta puntuación partiendo del perfilado previamente calculado.

Los esquemas de perfilado y puntuación usados por los principales sistemas de verificación son:

- Para Kushmerick [99], las características son consideradas variables aleatorias cada una de las cuales siguen una determinada distribución de probabilidad (consultar la Sección §2.3). En este caso, el perfil se puede interpretar como la probabilidad de que el valor observado siga la distribución de probabilidad previamente calculada.

  En cuanto a la puntuación, cuando llega un nuevo *slot* y queremos compararlo con los ya existentes, calculamos las probabilidades para cada una de las características. Esto nos dará un total de $|F|$ valores de probabilidad distintos, para resumir todos estos valores en uno solo, propone usar tres criterios: las variables aleatorias son independientes, hay una variable que es especialmente importante y todas las variables son igual de importantes.

- Lerman y otros [105], como se dijo en la Sección §2.3, sustentan su modelo de verificación en caracterizar los conjuntos de datos como un vector que alberga los valores medios de las características. Cuando tiene que comparar dos vector (el que representa al conjunto de entrenamiento y al que queremos verificar) usa el estadístico de Pearson que se define como:

$$p = \sum_{i=1}^{|\mathcal{F}|} \frac{mean_{US}(f_i) - mean_{TS}(f_i)}{d} \tag{5.14}$$

  donde $mean_{US}$ representa el valor media de la característica f_i en el conjunto de datos sin verificar y $mean_{TS}$ es la media de la característica f_i en el conjunto de entrenamiento.

  El perfilado de una característica consiste, simplemente, en calcular el sumatorio definido en el estadístico mientras que la puntuación es el cálculo del estadístico.

- McCann y otros [115] (consulte la Sección §2.3) proponen un modelo de verificación muy parecido al de Kushmerick [99] en donde se consideran conjunto de datos válidos e inválidos y se usa un cálculo de probabilidades normalizado.

  En ente caso, el perfilado es el cálculo de las funciones de densidades normalizadas mientras que la puntuación es la suma ponderada (asocia un peso a cada característica teniendo en cuenta su poder discriminador y si el conjunto de datos es válido o no) de las densidades normalizadas para cada una de las características.

```
APLICAVERIFICADOR(RS, F, perfilado, puntuacion, alarma, M, filtros)
 1  // Se aplican las mismas características tras hacer la fase de preproceso
 2  X_RS = CARACTERIZACION(RS, F)
 3  X_RS = PREPROCESADO(X_RS, M)
 4  perfilado = PERFILADO(X_RS)
 5  punt = PUNTUACION(perfilado);
 6  if ALARMA(punt, perfilado)
 7        // Pasamos los filtros sobre las componentes del perfil que generaron
 8        // la alarma
 9        if FILTROS(X_WS, perfilado, alarma)
10              ENVIO_ALARMA
11        else
12              SILENCIAR_ALARMA
13  return valido
```

Programa 5.3: *Algoritmo para el uso de un verificador*

5.6. Construcción del verificador

Una vez construido el modelo de verificación, definiremos el **verificador**, cuyo diagrama de clases lo podemos consultar en la Figura §5.6. Su función será recibir un **conjunto de datos** procedente del *wrapper* y comprobar si éste se ajusta al **modelo de verificación** previamente obtenido. Si no se cumple, el verificador lanzará una **alarma** para que un experto valide estos datos. En el Algoritmo §5.3 detallamos los pasos a realizar desde que el verificador recibe un conjunto de datos del verificador.

Podemos definir al verificador como una función matemática que recibe como argumento un conjunto de trabajo *WS* y devuelve un 1 si *WS* se asemeja al modelo de verificación y 0 en otro caso. Para hacer esta comprobación debe comprobar si cada uno de los RS que forman parte del *WS* siguen el modelo de verificación previamente calculado:

$$v : \underset{\left[\begin{array}{cccc} x_{1,1} & x_{1,2} & \dots & x_{1,|\mathcal{F}|} \\ x_{2,1} & x_{2,2} & \dots & x_{2,|\mathcal{F}|} \\ \vdots & \vdots & \dots & \vdots \\ x_{|RS|,1} & x_{|RS|,2} & \dots & x_{|RS|,|\mathcal{F}|} \end{array}\right]}{|RS| \times \mathbb{R}^{|\mathcal{F}|}} \begin{array}{c} \longrightarrow \\ \\ \\ \longmapsto \end{array} \begin{array}{c} \mathbb{R} \\ \\ \\ \{0,1\} \end{array} \tag{5.15}$$

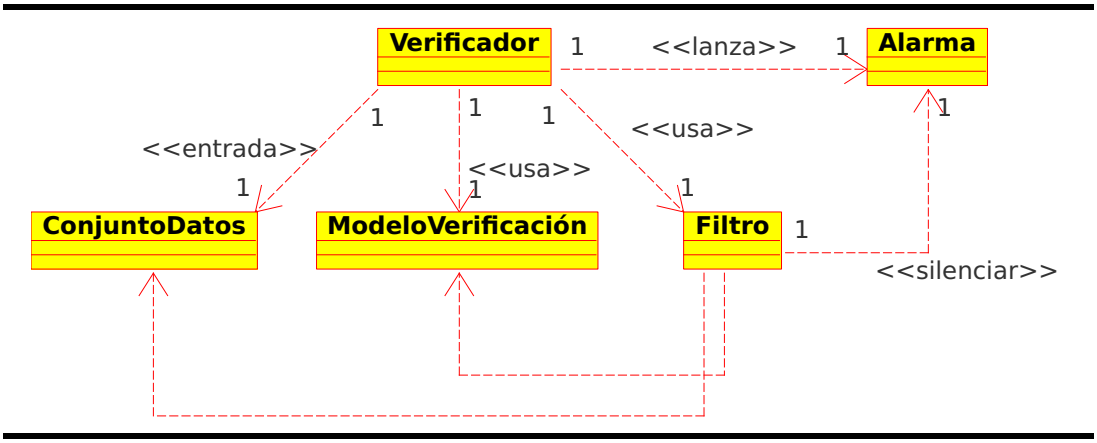


Figura 5.6: *Modelo de clases del verificador*

Ahora bien, en ciertas ocasiones, al comprobar la validez de un conjunto de datos, el verificador devuelve un falso negativo (indica que es inválido cuando realmente no lo es) debido a que dicho conjunto presenta algunos slots no deseados, esto es, elementos que se distancian mucho del resto de datos y que suelen indicar una anomalía temporal en el proceso de medición.

Para evitar este tipo de problemas, los verificadores incluyen **filtros** que se encargan de sanear el conjunto de datos, explorando sus valores en busca de elementos que puedan provocar una errónea clasificación del mismo.

Los filtros más usados dentro de este campo son [116]:

1. **Fuentes externas**: Usa una serie de fuente de datos externas semánticamente equivalentes. Dichas fuentes nos servirán para crear un nuevo modelo que se usará para comprobar la validez de aquellos atributos conflictivos del conjunto de datos a validar.

2. **Apendizaje basado en la web**: En este caso, se asocia a cada clase de slot una serie de elementos léxicos obtenidos de las frases en las que encontramos el atributo. Por ejemplo, los precios suelen estar asociados a los elementos 'coste' o '€'. Si necesitamos comprobar la validez de un atributo, usamos un motor de búsqueda para obtener aquellas páginas en las que se encuentra sólo el atributo y aquellas en las que se aparece acompañado de sus elementos léxicos. Si en dicha búsqueda el número de páginas que contienen los elementos léxicos es superior al que aparece simplemente el atributo, entonces dicho atributo se considerará válido.

Observe que el concepto de filtro y característica es bastante similar, la diferencia fundamental es que los filtros son mucho más costosos, computacionalmente hablando, que el cálculo de las características.

5.7. Resumen

A lo largo de este capítulo, hemos ido viendo las distintas fases por las que debemos pasar a la hora de construir un verificador. Estas fases se han integrado dentro de un marco de trabajo que permite sistematizar la creación de verificadores. En cada una de las fases se ha hecho referencia a los sistemas de verificación ya existentes para así poder compararlos y compartir terminología.

Capítulo 6

Verificando wrappers usando algoritmos bioinspirados

¡y por las hormigas oprimidas del mundo!

Bichos, Película (1998)

Los algoritmos bioinspirados, y en especial los basados en colonias de hormigas, se han venido usando en la resolución de diversos problemas de optimización combinacional. En este capítulo detallamos los pasos seguidos para resolver el problema de la verificación de wrappers usando este tipo de técnicas. En la Sección §6.1 introduciremos los principales elementos que permitirán reformular, en la Sección §6.2, el problema de verificación desde el punto de vista de optimización. En la Sección §6.3 explicaremos, detenidamente, cómo resolver este tipo de problemas usando algoritmos de hormigas. En la Sección §6.4 detallaremos los aspectos propios del algoritmo BWAS adaptado para afrontar la verificación de wrappers. Por último, la Sección §6.5 la dedicaremos a resumir todas las ideas tratadas en este capítulo.

6.1. Introducción

En este capítulo pretendemos solventar las carencias de los sistemas de verificación actuales, ya descritas en el Capítulo §2, gracias a una nueva propuesta de verificador basada en metaheurísticas bioinspiradas [106]; concretamente aplicaremos los sistemas de colonias de hormigas (ACO) [57, 62] ya discutidos en el Capítulo §3. Hasta la fecha este tipo de metaheurísticas no se habían aplicado para resolver la verificación de *wrappers*.

De entre las distintas propuesta de algoritmos ACO existentes, hemos usado el algoritmo BWAS (Sistema de la Mejor Peor Hormiga) que, como ya vimos en la Sección §3.6, es un ejemplo de metaurística híbrida que mezcla conceptos de la metaheurística ACO y otras metaheurísticas evolutivas [24, 32, 34].

Como algoritmo de optimización discreta, la propuesta descrita parte de la representaciones usadas para resolver el problema de la asignación cuadrática de tareas o *Quadratic Assignement Problem* (QAP) [75]. Como veremos, la representación se simplifica bastante ya que muchas de las restricciones planteadas por el QAP no las presenta el problema de la verificación. Para mejorar la fase de explotación del proceso de búsqueda, usaremos un proceso de búsqueda local que mejorará la calidad de las soluciones alcanzadas por la colonia.

6.2. La verificación como un problema de optimización computacional

Tal y como se detalló en el Capítulo §5, el problema de la verificación de información de *wrappers* se traduce en comprobar si los conjuntos de datos extraídos son o no válidos. Cada uno de los *slots* s_i que forman estos conjuntos de datos RS tienen asignado una etiqueta l_i que identifica su naturaleza. Es responsabilidad del verificador comprobar si cada uno de los *slots* contenidos en los conjuntos de datos tiene asignado la etiqueta que se corresponde con el atributo contenido en dicho *slot*.

Durante este capítulo asumimos que todos los *slots* son validos por lo que $\delta = 1$, el conjunto de entrenamiento TS sólo tiene un conjunto de datos RS y el conjunto $\mathcal{F}_{RS}$ está vacío (no hay características de conjunto de datos) por lo que $\mathcal{F} = \mathcal{F}_s$. Por tanto, $TS = \{(w, q_1, RS_1, 1)\}$, $X_{TS} = \{(w, q_1, X_{RS_1}, 1)\}$ y $X_{RS} = \{(X_{s1}, l_1), (X_{s2}, l_2), ..., (X_{s|RS|}, l_{|RS|})\}$.

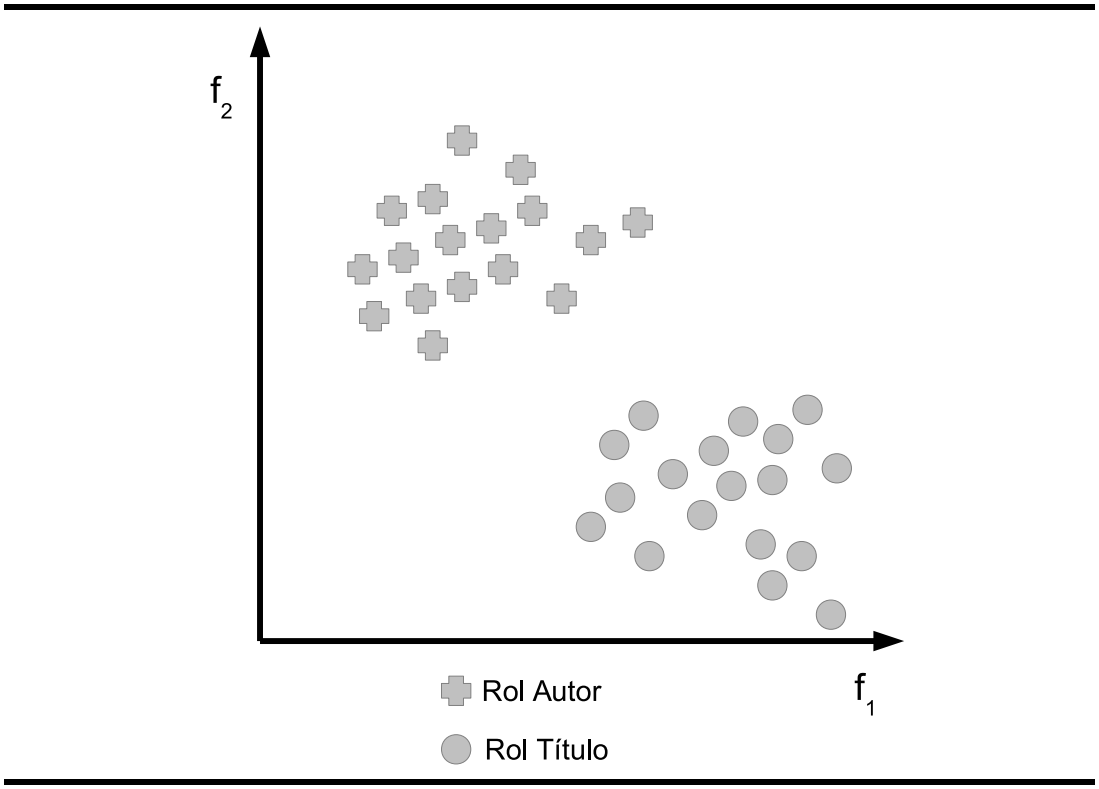
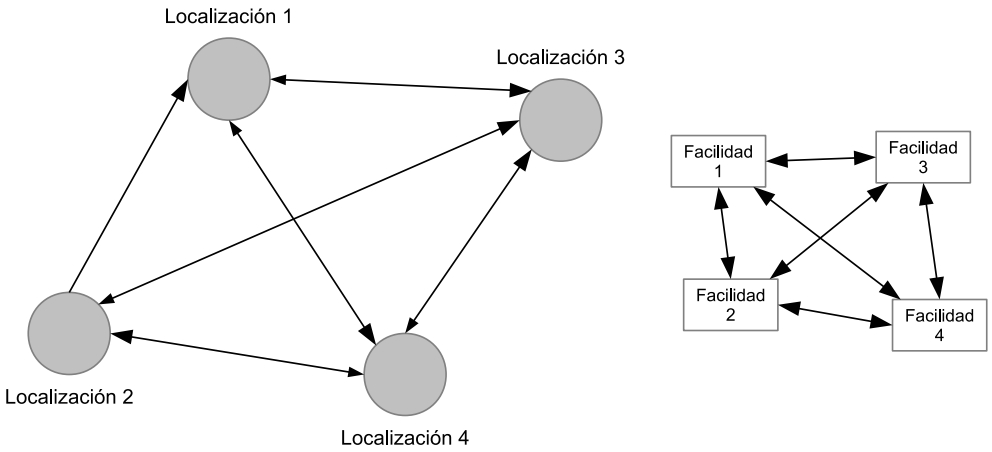


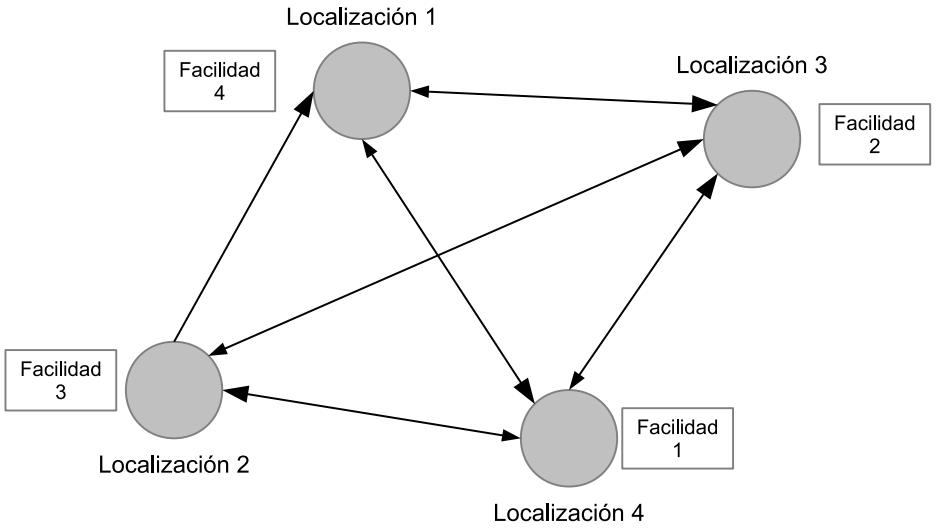
Figura 6.1: *Slots representados en un espacio 2-dimensional*

Como los *slots* se caracterizan mediante vectores de valores X_s, estos se pueden representar como puntos en un espacio n-dimensional, obteniendo distribuciones similares a las indicadas en la Figura §6.1. En ella se representa la localización de distintos *slots*, pertenecientes a los roles de interés: `título` y `autor`, una vez caracterizados usando únicamente dos características. Observe que este tipo de representación asume que las características categóricas se puede representar como numéricas. Por ejemplo, aquellas características de naturaleza lógica se puede pasarían a representarse mediante 1 (valor cierto) y 0 (valor falso).

Un problema de optimización combinacional consiste en optimizar una función real eligiendo, sistemáticamente, valores de entrada (tomados de un conjunto permitido) y computando el valor de la función [12]. Uno de los problemas clásicos dentro del campo de la optimización combinacional es el denominado asignación cuadrática de tareas (QAP). En este problema se parte de dos conjuntos, P ('facilidades') y L ('localizaciones' fijas) de igual tamaño, junto a una función de flujo $w : P \times P \rightarrow R$ y una



(a) *Definición de localización, facilidades y funciones de distancia*



(b) *Asignación de facilidades a localizaciones para minimizar el coste*

Figura 6.2: *Representación del QAP*

función $d : L \times L \rightarrow R$ que representa la distancia entre localizaciones (Figura §6.2(a)); debemos encontrar la biyección $f : P \rightarrow L$, también denominada de asignación, tal que se minimice la función de coste:

$$\min_{f \in \Omega} \sum_{a,b \in P} w(a,b) \bullet d\,(f(a), f(b)) \tag{6.1}$$

donde Ω representa el conjunto de biyecciones posibles (Figura §6.2(b)).

Por lo general, las funciones de flujo y distancia se pueden ver como matrices cuadradas simétricas de números reales, por lo cual, la función de coste se puede escribir como:

$$\min_{f\in\Omega} \sum_{a,b\in P} w_{a,b} \bullet d_{f(a),f(b)} \tag{6.2}$$

donde Ω representa el conjunto de biyecciones posibles, P es el número de facilidades, w es una función de flujo, d es una función de distancia y f es la función de asignación.

Por ejemplo, si disponemos de un sistema de procesamiento con tres ordenadores (localizaciones) y 6 tareas a realizar (facilidades), sabemos que todas las tareas se pueden hacer en cualquier ordenador y los costes asociados a cada una de las posibles asignaciones es el siguiente:

		Ordenadores		
		O_1	O_2	O_3
Tareas	T_1	18	16	12
	T_2	14	21	18
	T_3	16	24	24

La función objetivo a minimizar sería:

$$\begin{aligned} \min \quad & w_{1,1} \bullet 18 + w_{1,2} \bullet 16 + w_{1,3} \bullet 12 + w_{2,1} \bullet 14 + w_{2,2} \bullet 21 \\ & + w_{2,3} \bullet 18 + w_{3,1} \bullet 16 + w_{3,2} \bullet 24 + w_{3,3} \bullet 24 \end{aligned} \tag{6.3}$$

donde

$$w_{ij} = \begin{cases} 1, & \text{si la tarea } i \text{ se asigna al ordenador } j \\ 0, & \text{en otro caso} \end{cases} \tag{6.4}$$

Es este caso hemos formulado el QAP como un problema de programación entera binaria donde iremos asignando valores 0 o 1 a los distintos $a_{i,j}$ hasta conseguir minimizar el coste.

6.2.1. Relación entre la verificación de wrappers y el QAP

Intuitivamente existe una estrecha relación entre las Figuras §6.1 y §6.2. Estableciendo un paralelismo entre ambas podemos decir que los *slots* se pueden tratar como facilidades que se asignan a etiquetas (localizaciones).

Ahora bien, existen una serie de peculiaridades propias del problema de la verificación que hay que tener en cuenta antes de reformularlo como un caso particular de QAP. La primera diferencia está relacionada con la función de flujo, que mide el grado de interacción entre facilidades. En nuestro caso, la interacción entre *slots* es siempre la misma. Por tanto, la matriz w tendría el mismo valor independientemente de la posición que consultáramos.

La segunda diferencia la encontramos a la hora de tratar las etiquetas como si fueran localizaciones. Para nosotros, las etiquetas o roles no son elementos que se puedan representar en un espacio n-dimensional, como sí es el caso de las localizaciones, son simples cadenas de texto. Por tanto, para poder tratar el problema de la verificación como un problema QAP debemos encontrar la forma de representar las etiquetas dentro de un espacio n-dimensional.

El método seguido para lograrlo parte del perfilado expuesto por Lerman y otros [105] de los conjuntos de trabajo. En [105], este conjunto se dividía en tantos subconjuntos como etiquetas, y cada uno de ellos era modelizado mediante un vector que representaba el valor medio de las distintas características. Estos puntos medios los podríamos considerar como una representación de las etiquetas.

La idea propuesta generaliza este método ya que representa los roles mediante un conjunto de puntos $P_i = \{p_1, p_2, ..., p_K\}, i = 1...|\mathcal{L}|$, en lugar de usar un único vector de valores medios. El procedimiento de obtención de los K puntos por rol es un proceso iterativo, basado en una función de coste, que busca posicionarlos de forma que todos los *slots* igualmente etiquetados estén representados por ellos. Este conjunto de puntos representan el modelo de verificación. La Figura §6.3(a) ilustra este hecho, disponemos de *slots* de dos roles diferentes (`título` y `autor`), a cada uno de estos dos conjuntos se les representa mediante tres puntos. Así, todos los *slots* con el rol `autor` quedan representados por los puntos p1, p2 y p3 mientras que los etiquetados como `título` quedarán definidos por los puntos p4, p5 y p6

Antes de la creación del modelo de verificación, y como veremos en el capítulo siguiente, se ha procedido a dividir (Sección §7.3.1.1) y preprocesar (Sección §7.3.1.2) el conjunto de entrenamiento.

Cuando tengamos que verificar nuevos elementos, calcularemos la distancia al punto más cercano y, si este está lo suficientemente cerca el *slot*, debería estar etiquetado con la misma etiqueta a la que pertenece el punto. En la Figura §6.3(b) se pone un ejemplo de dicho comportamiento. Disponemos de dos *slots*, s_1 y s_2 que el *wrapper* etiquetó como del rol `autor`, el primero de ellos, s_1 tiene como punto más cercano el p_2, que representa el rol `autor`, por lo que el verificador entiende que está bien etiquetado. En cambio, el punto más cercano al segundo *slot* s_2 es p_4, que es del rol `título`, por lo que el verificador entenderá que ha sido mal etiquetado y generará una alarma. Si nos fijamos nuevamente en la figura podemos apreciar que hay *slots* etiquetados como `autor` que están más cerca de s_2 que el propio p_4 pero, como los *slots* de un rol están ahora representados únicamente por los puntos antes calculados y no por los propios *slots*, no le podemos asignar el rol `autor`.

En los apartados siguientes pasamos a detallar cómo obtener la localización óptima de los puntos usando algoritmos de hormigas.

6.3. Usando la metaheurística ACO para verificar wrappers

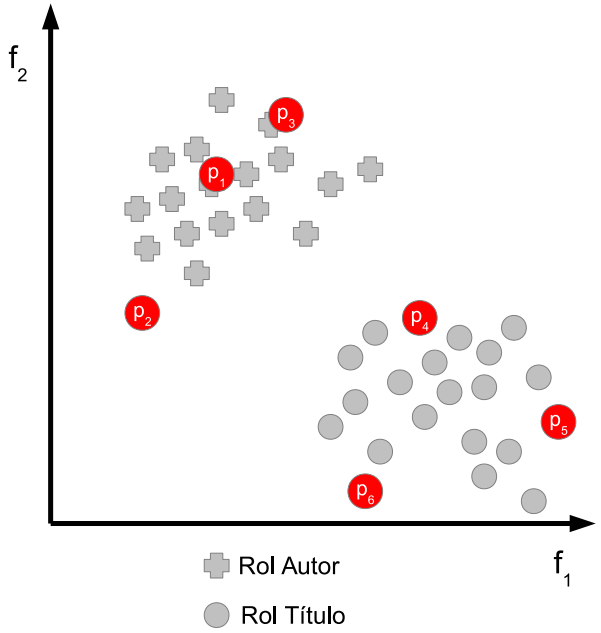
Los algoritmos ACO son técnicas de búsqueda global [62] que, desde que fueron propuestas, han ido evolucionando dando lugar a diferentes modelos y ampliando su ámbito de aplicación a un amplio abanico de problemas de optimización [35, 55, 57].

Como se detalla en [36], para poder aplicar ACO a un problema de optimización hay que definir:

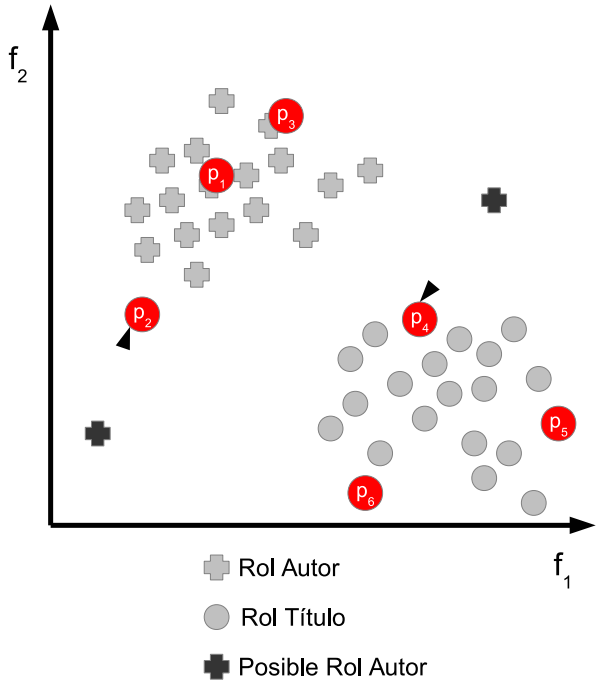
- La representación del problema (Sección §6.3.1),
- La función heurística (Sección §6.3.2),
- La inicialización de la feromona (Sección §6.3.3),
- La inicialización de la colonia (Sección §6.3.4) y
- La función de fitness (Sección §6.3.5).

6.3.1. Representación del problema

Para poder aplicar las técnicas basadas en colonias de hormigas al problema de la verificación de *wrappers*, es conveniente que lo tratemos como un



(a) *Representación de las etiquetas como puntos*



(b) *Asignación de slots al punto más cercano*

Figura 6.3: *Respresentación de etiquetas y asignación a slots*

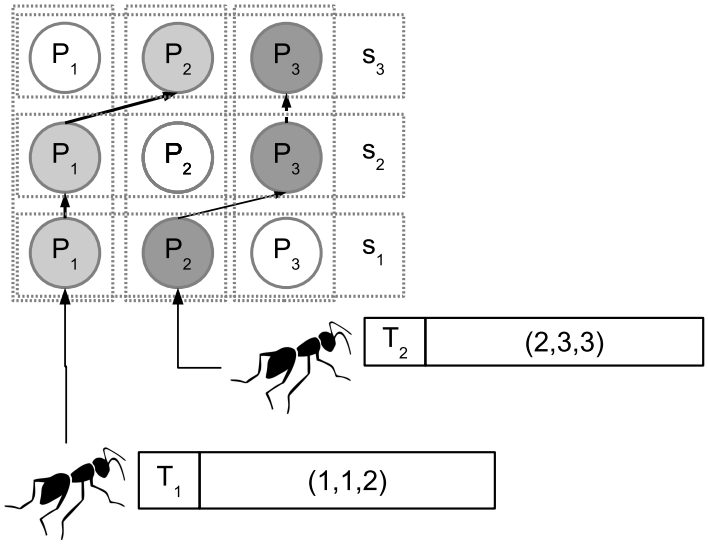


Figura 6.4: *Representación del problema de la verificación como grafo etiquetado*

problema de optimización combinatoria capaz de representarse mediante un grafo ponderado y donde una solución válida al problema se puede formular mediante el movimiento de una hormiga a lo largo del grafo.

Para el problema de la verificación, este grafo se puede representar mediante una matriz de tamaño $|RS|$ x K. Cada arco $A_{i,j}$ representa la asignación del *slot* i al punto *j*. Cada *slot* s sólo puede ser asignado a un punto p_j.

La Figura §6.4 muestra cómo sería esta matriz para, por ejemplo, el conjunto de entrenamiento considerando únicamente los slots que tienen como rol `autor` X_{RS} = *{([2, 0.5], autor), ([2, 0.5], autor), ([3, 0.66], autor)}* si usamos tres puntos para representarlo.

También en la Figura §6.4 se presentan los caminos seguidos por dos hormigas para solucionar el problema. Cada hormiga i está asociada con una matriz T_i donde se almacenan las decisiones que ha tenido que tomar cada vez que ha asignado un *slot* a un punto. Por ejemplo, la primera hormiga tiene almacenada la matriz $T_1 = (1, 1, 2)$ la cual se interpreta como: el primer y segundo *slots* se han asignado al primer punto y el tercer *slot* se asigna al segundo punto. Además, cada hormiga i tiene un vector Z_i, con dimensiones K x $|\mathcal{F}|$, para almacenar las coordenadas de cada uno de los puntos calculados.

6.3.2. Información heurística

La información heurística η asociada a que la hormiga i tome la decisión de asignar un *slot* s al punto p viene dada por la expresión:

$$\eta^{i}_{s,p} = 1/h(i,s,p) \tag{6.5}$$

donde:

$$h(i,s,p) = \sqrt{\sum_{v=1}^{|\mathcal{F}|}(x_{s,v} - Z_{i,p,v})^2} \tag{6.6}$$

siendo X_s la caracterización de s y $Z_{i,p,v}$ es la coordenada v del punto p inferido por la hormiga i. Por ejemplo, los valores heurísticos calculados por la primera hormiga cuando tomó la decisión de asignar s_1 a p_1 es $\eta^1_{s_1,p_1} = 1/0$, a p_2 es $\eta^1_{s_1,p_2} = 0$ y a p_3 es $\eta^1_{s_1,p_3} = 1/1.05$. Todo esto asumiendo que las coordenadas de los puntos calculados por la primera hormiga se inicializaron a $Z_1 = (2, 0.5), (2, 0.5), (3, 0.66)$.

6.3.3. Inicialización de la feromona

En algunos problemas, como el del viajante de comercio, el valor inicial de la feromona viene dado en función de los valores heurísticos. Para el problema de la verificación de *wrappers* no disponemos de esta información por lo que la matriz de feromona se inicializa a un valor de feromona próximo a 0.

6.3.4. Inicialización de la colonia

La inicialización de la colonia suele implicar el posicionamiento aleatorio de cada una de las hormigas en distintas posiciones del grafo que representa el problema. Para el problema de la verificación de *wrappers*, al igual que se aplica en el problema del viajante de comercio, todas las hormigas se sitúan inicialmente en la misma posición. Ahora bien, cada hormiga tiene asociado un vector Z (representa la posición de los puntos a calcular) que tiene que ser inicializado. Asumiremos que, inicialmente, los valores de Z se seleccionarán aleatoriamente de entre las posiciones de los *slots* $s \in X_{RS}$.

6.3.5. Función de fitness

La función de *fitness* asociada a la solución construida por una hormiga se calcula sumando las distancias entre los *slots* y el punto que tiene asociado. Más formalmente, ésta se define como:

$$fit(i) = \sum_{j=1}^{|RS|} \sum_{p=1}^{K} w_{j,p} \bullet \sqrt{\sum_{v=1}^{|\mathcal{F}|} (x_{j,v} - Z_{i,p,v})^2} \tag{6.7}$$

donde K es el número de puntos, $w_{j,p}$= 1 si el slot *j* ha sido asociado al punto *p* y $w_{j,p} = 0$ en otro caso. Esta función de *fitness* se asemeja mucho a la formulación entera binaria del QAP.

Para el ejemplo mostrado en la Figura §6.4, el valor de *fitness* de la solución obtenida por la primera hormiga, $T_1 = \{1, 1, 2\}$, sería:

$$\begin{aligned} fit(1) \;=\; & w_{1,1} \bullet d_1 + w_{1,2} d_2 + w_{1,3} + d_3 + \\ & w_{2,1} \bullet d_4 + w_{2,2} \bullet d_5 + w_{2,3} \bullet d_6 + \\ & w_{3,1} \bullet d_7 + w_{3,2} \bullet d_8 + w_{3,3} \bullet d_9 \end{aligned} \tag{6.8}$$

donde $w_{1,2} = w_{1,3} = w_{2,2} = w_{2,3} = w_{3,1} = w_{3,3} = 0$ ya que s_1 ha sido asociada a p_1, s_2 a p_1 y s_3 a p_2. Así pues la función de *fitness* se reduce a calcular

$$fit(1) \;=\; w_{1,1} \bullet d_1 + w_{2,1} \bullet d_4 + w_{3,2} \bullet d_8 \tag{6.9}$$

donde d_1 es la distancia entre el *slot* s_1 y el punto $Z_{1,1}$, d_4 la distancia entre el *slot* s_2 y el punto $Z_{1,1}$ y d_8 la distancia entre el *slot* s_3 y el punto $Z_{1,2}$. Si calculamos estas distancias, y teniendo en cuenta que $w_{1,1} = w_{2,1} = w_{3,2} = 1$, obtenemos un valor de *fitness* de 1.05. Todos estos cálculos suponiendo que las coordenadas de los puntos calculados por la primera hormiga se inicializaron a $Z_1 = (2, 0.5), (2, 0.5), (3, 0.66)$.

6.4. Usando el algoritmo BWAS para verificar wrappers

BWAS intenta mejorar el rendimiento de los modelos ACO usando diferentes métodos para las reglas de transición, actualización de rastros y añade una serie de nuevos elementos con una fase de búsqueda local [18, 60, 143]. Para aplicar BWAS a cualquier problema de optimización combinatoria, conforme al Algoritmo §3.4, es necesario definir los siguientes elementos [35]:

- Construcción de soluciones (Sección §6.4.1)
- Mecanismo de evaporación de la feromona (Sección §6.4.2)
- Proceso de búsqueda local (Sección §6.4.3)
- Actualización de la feromona (Sección §6.4.4)
- Mutación de la feromona (Sección §6.4.5)
- Reinicialización del proceso de búsqueda (Sección §6.4.6)

6.4.1. Construcción de soluciones

Durante el proceso de construcción de las soluciones, en cada paso, un *slot* $s \in X_{RS}$ es seleccionado aleatoriamente. Este slot s se asigna a un punto $p \in P$ con la probabilidad dada por la siguiente regla de transición:

$$\mathcal{P}(s,p) = \begin{cases} \frac{\tau_{s,p}^{\alpha} \bullet \eta_{s,p}^{\beta}}{\sum_{l \in P} \tau_{s,l}^{\alpha} \bullet \eta_{s,l}^{\beta}}, & \text{si } T_i[s] = 0 \\ 0 & \text{en otro caso} \end{cases} \tag{6.10}$$

con $\tau_{s,p}$ siendo la cantidad de feromona en el arco $A_{s,p}$ (la feromona asociada a la decisión de asignar el *slot* s al punto p), $\eta_{s,p}$ es la información heurística y α y β son los parámetros que determinan la influencia relativa del rastro de feromona y de la información heurística respectivamente.

Para este problema, es necesario mantener una lista tabú que permita recordar a cada hormiga las asignaciones que ha realizado con anterioridad, es así porque el mismo *slot* no puede ser asignado a dos puntos distintos.

Una vez que todas las hormigas han construido las soluciones, a cada punto p usado para construir el modelo X_{RS}, se actualizan sus coordenadas siguiendo la fórmula:

$$Z_{i,p,v} = \frac{\sum_{i=1}^{|RS|} w_{i,p} \bullet x_{i,v}}{\sum_{i=1}^{|RS|} w_{i,p}}, \quad v = 1..|\mathcal{F}| \tag{6.11}$$

donde $w_{i,p}$= 1 si el *slot* i ha sido asignado al punto p o $w_{i,p} = 0$ en otro caso. De nuevo usamos una formulación parecida a la entera binaria del QAP.

6.4.2. Mecanismo de evaporación de la feromona

El mecanismo de evaporación se realiza conforme a la formula:

$$\tau_{rs} \leftarrow (1 - \rho) \bullet \tau_{rs} \tag{6.12}$$

donde $\rho \in [0, 1]$ es el parámetro que controla la evaporación de la feromona.

6.4.3. Proceso de búsqueda local

Una de las formas más comunes para mejorar el rendimiento de los algoritmos de colonias de hormigas es usando técnicas de búsqueda local [63]. Esta aproximación implica el uso de técnicas de optimización que permita refinar las soluciones obtenidas por las hormigas en cada iteración. Una vez hecho esto, el rastro de la feromona se actualizará de la forma habitual (consulte la siguiente sección).

Este proceso de búsqueda local consiste en aplicar el algoritmo de las k-medias al conjunto de puntos Z_i calculados por cada una de las hormigas del algoritmo BWAS en cada una de las iteraciones. El algoritmo de las k-medias irá posicionando aleatoriamente estos Z_i puntos con el objetivo de optimizar la función:

$$h(i, Z_i) = \sum_{j=1}^{|RS|} \min_{k} \|X_j - Z_{i,k}\|^2 \tag{6.13}$$

donde X_j la caracterización de un *slot* s, $Z_{i,k}$ es el punto k inferido por la hormiga i.

6.4.4. Actualización de la feromona

Los rastros de actualización de la feromona se actualizan conforme a:

$$\tau_{rs} \leftarrow \tau_{rs} + \rho \bullet f(C(S_{mejor\_global})),\ \forall rs \in S_{mejor\_global} \tag{6.14}$$

donde S_{mejor_global} es el rastro de la mejor solución global y $f(C(S_{mejor\_global}))$ se corresponde al coste asociado a la asignación de *slots* a los puntos definido en la solución alcanzada por la mejor hormiga local.

A continuación, todos los arcos de la peor solución actual, $S_{peor\,actual}$, que no estén presentes en la mejor solución global, se penalizan reduciendo la cantidad de feromona que tiene asociada de la siguiente manera:

$$\tau_{rs} \leftarrow (1 - \rho) \bullet \tau_{rs},\ \forall rs \in S_{peor\_global} \wedge rs \notin S_{mejor\_global} \tag{6.15}$$

6.4.5. Mutación

Cada fila de la matriz de feromona se muta -con probabilidad P_m- sumando o restando la misma cantidad de feromona al rastro seleccionado (el valor depende de la iteración actual) de la siguiente manera:

$$\tau'_{rs} = \begin{cases} \tau_{rs} + mut(it, \tau_{umbral}), & si\ a = 1 \\ \tau_{rs} - mut(it, \tau_{umbral}), & si\ a = 0 \end{cases} \tag{6.16}$$

donde a es una valor aleatorio en $0, 1$, it es la iteración actual, τ_{umbral} es la media de los rastros de feromona depositados en los arcos que representan las asignaciones de *slots* a puntos hecha por la hormiga que ha alcanzado la mejor solución actual y $mut(\bullet)$ es:

$$mut(it, \tau_{umbral}) = \frac{it - it_r}{N_{it} - it_r} \bullet \sigma \bullet \tau_{umbral} \tag{6.17}$$

donde N_{it} es el número máximo de iteraciones del algoritmo, σ en una constante que establece la potencia de la mutación e it_r es la última iteración donde se realizó una reinicialización.

La función *mut* no previene que los valores de feromona sean negativos, por lo que es necesario comprobarlos tras aplicarla.

6.4.6. Reinicialización del proceso de búsqueda

El algoritmo realizará el reinicio asociando a cada uno de los elementos de la matriz de feromona el valor de τ_0, el valor inicial de feromona, cuando la mejor solución global no se mejoró durante un número fijo de iteraciones.

6.5. Resumen

Es este capítulo hemos tratado el problema de la verificación de *wrapper* como un problema de optimización combinatoria. De las distintas técnicas que se pueden aplicar, hemos propuesto el uso del paradigma bioinspirado de los algoritmos basado en colonias de hormigas. En concreto, aplicamos el algoritmo BWAS que combina elementos de otras técnicas bioinspiradas.

Capítulo 7

Verificación de wrappers usando clasificadores de una clase

Es preferible ser reconocido por desconocidos que desconocido por conocidos.

Anónimo, Anónimo

El problema de la verificación de wrappers se puede reformular como un problema de clasificación si lo tratamos desde el punto de vista de vectores y distancias entre ellos. En este capítulo lo enfocaremos como un problema de clasificación de una única clase ya que, como veremos, las instancias del conjunto de entrenamiento, usado para construir el verificador, no representan a todo el dominio del discurso. Este capítulo se organiza de la siguiente manera: en la Sección §7.1 introduciremos las ideas generales a tratar, la Sección §7.2 justifica el uso de las técnicas de clasificación de una única clase, en la Sección §7.3 tratamos la aplicación concreta de estas técnicas, en la Sección §7.4 propondremos MAVE, un sistema de verificación multinivel; finalmente en la Sección §7.5 resumiremos los puntos tratados en el capítulo.

7.1. Introducción

Tal y como se describió en el Capítulo §5, el problema de la verificación parte de la creación de un modelo de verificación muy robusto. Una vez creado, la verificación de nuevos *slots* consiste en comprobar si estos están, de alguna forma, relacionados con el modelo precalculado. Si tratamos a los *slots* como vectores y establecemos la similitud entre ellos atendiendo a las distancias existentes entre *slots* pertenecientes a la misma clase, el problema de la verificación de *wrappers* se puede ver como un problema de clasificación.

El principal inconveniente al que nos enfrentamos al reformular este problema como uno de clasificación, es que el conjunto de entrenamiento está sesgado. Esto es, los *slots* de los que disponemos durante la fase de entrenamiento no se corresponden con los diferentes tipos que nos encontraremos en la fase de uso del clasificador. Por ejemplo, podemos haber diseñado un *wrapper* para obtener `precios` y `títulos` de Ebay; puede ser que en un futuro, más o menos cercano, los responsables de esta web decidan hacer modificaciones sin previo aviso y provocar que, además de `títulos` y `precios` extraigamos, por ejemplo, `descuentos`. Cuando se creo el *wrapper* se desconocía la existencia de `descuentos` por lo que el conjunto de entrenamiento que se usó carecía de *slots* de esta clase. Este hecho introduce un problema añadido al proceso de clasificación que, como se explicó en el Capítulo §4, no puede ser subsanado por los clasificadores multiclase sino que hay que aplicar los clasificadores de una única clase.

La aplicación de este tipo de técnicas requiere que adaptemos algunos de los elementos del marco de trabajo para la creación de verificadores (Capítulo §5). En este capítulo nos centraremos en las fases de construcción y uso del modelo y del verificador.

7.2. El problema de la verificación como un problema de clasificación

Un problema de clasificación, como ya se indicó en el Capítulo §4, consiste en asignar una serie de etiquetas a un conjunto de vectores n-dimensionales. Es un proceso de aprendizaje que requiere de un conjunto de entrenamiento, formado por vectores etiquetados, para crear y parametrizar el clasificador. Durante la fase de uso del clasificador, irán llegando

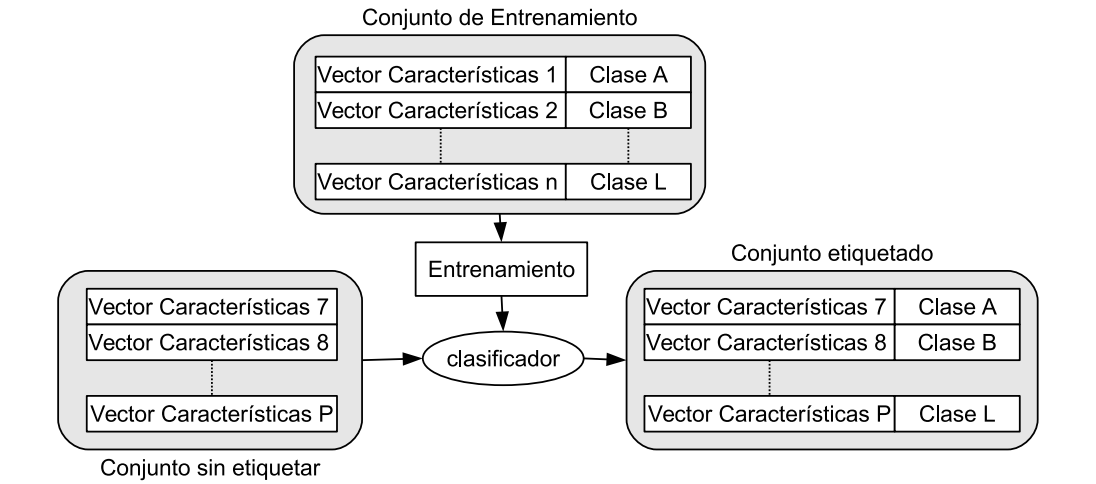


Figura 7.1: *Esquema básico de clasificación*

nuevos vectores a los que el clasificador les asocia una clase. Si el clasificador ha sido construido adecuadamente, esta etiqueta se corresponderá con el elemento que representa el vector (Figura §7.1).

Por otro lado, el problema de la verificación de *wrappers*, tal y como se detalló en el Capítulo §5, consiste validar si un conjunto de datos asociados a una etiqueta (o rol) es válido o invalido (en este caso se envía una alarma) dependiendo de si estos se asemejan a un modelo de verificación previamente establecido. Como en el caso de la clasificación, es un proceso de aprendizaje en el que se usa un plan de recolección para obtener los datos que se usarán durante la fase de entrenamiento. Dichos conjuntos están formados (ver Sección §5.5.1) por vectores n-dimensionales etiquetados.

Por tanto, existe cierta analogía entre ambos problemas ya que los dos parten de lo mismo: un conjunto de vectores con etiquetas. En cambio, mientras que un clasificador -durante la fase de uso- trata con elementos no etiquetados, el verificador recibe *slots* etiquetados. Estas etiquetas están asociadas automáticamente por el *wrapper* por lo que nos podemos encontrar con *slots* mal etiquetados. Es pues, el problema de verificación un problema de clasificación donde tenemos que comprobar si la etiqueta que asociaría un clasificador a un *slot* se corresponde con la que ha asociado el *wrapper* (Figura §7.2).

En los puntos siguientes justificaremos el tipo de clasificador que usaremos a la hora de verificar *wrappers*.

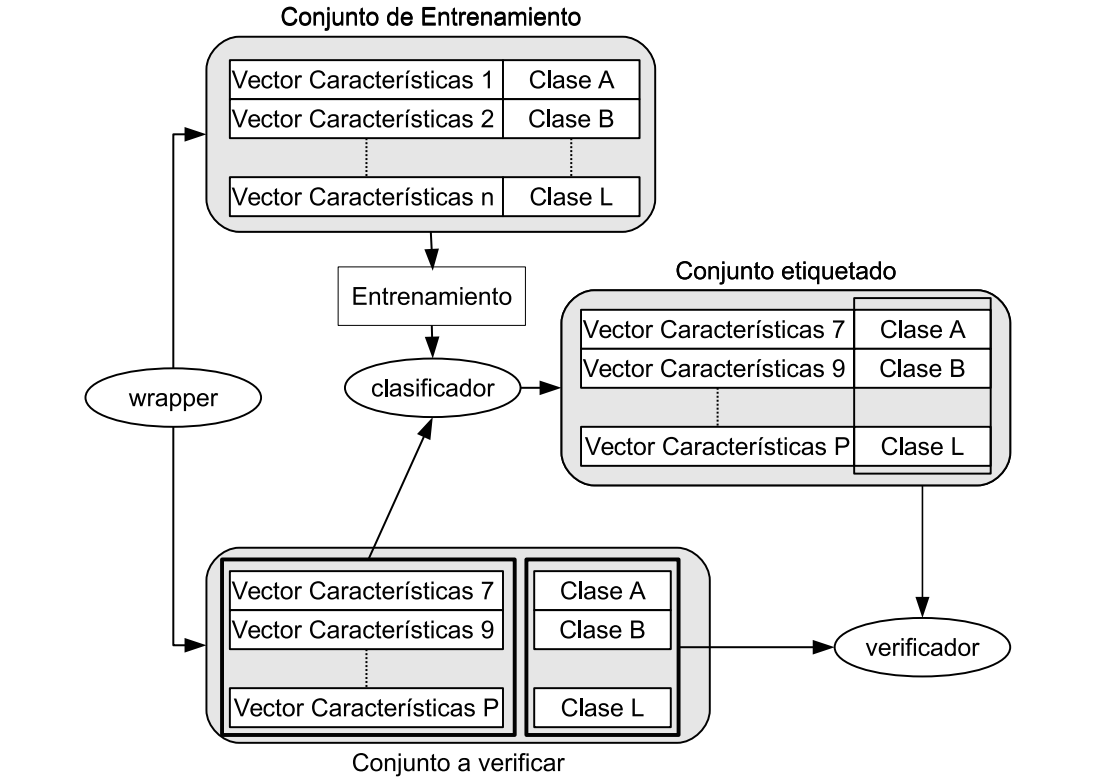


Figura 7.2: *Relación entre clasificación y verificación*

7.2.1. Enfoque de clasificación multiclase

Los clasificadores multiclase (Sección §4.2) parten de un conjunto de entrenamiento en el que están identificadas todas aquellas clases que quizás aparezcan durante la fase de clasificación o uso. En cambio, y tal y como se discutió en la Sección §5.4, en el problema de la verificación, los conjuntos de entrenamiento no contienen *slots* de todas las clases posibles. Esto es, el *wrapper* puede extraer elementos pertenecientes a clases para las que no se había entrenado el verificador.

Por tanto, no podemos usar este tipo de clasificadores ya que todo elemento nuevo lo clasificaría entre lo conocido aunque no se parezca. Para solucionarlo podemos optar por las técnicas ya comentadas en las Secciones §5.4 y §4.3. Estos procedimientos intentan generar ejemplos negativos que me permitan obtener clasificadores multiclase con mayor capacidad de generalización. El problema de estas técnicas, como ya se mencionó, es que o los

ejemplos negativos son demasiado sintéticos (propuestas en [115]) o se necesita una alta capacidad de cómputo para generar buenos ejemplos en altas dimensionalidades (propuestas en [95, 145]).

También podríamos enfocar el problema de la verificación de *wrappers* como el de una clasificación binaria. En este caso nuestro problema de clasificación de $|\mathcal{L}|$ clases lo convertiríamos en $|\mathcal{L}|$ problemas de clasificación binaria.

Cada uno de estos clasificadores sirven para distinguir una clase del resto. Así el clasificador para la primera clase usará, como ejemplos positivos, ejemplos de la primera clase que estén en el conjunto de entrenamiento y, como ejemplos negativos, el resto. A la hora de verificar tendríamos que usar el clasificador que coincida con la etiqueta con la que fue etiquetado el *slot*. Si el clasificador indica que es de la clase, entonces el *slot* es válido.

Aún así, seguimos teniendo el mismo problema, estamos generando clasificadores que establecen fronteras para separar elementos de clases conocidas.

7.2.2. Enfoque de clasificación de una clase

El comportamiento que deseamos de un clasificador para el problema de la verificación, no es que establezca fronteras que discriminen entre clases. Lo que buscamos, son clasificadores que caractericen las clases que conocemos y las clasifiquen correctamente en base al conocimiento que únicamente tengamos sobre ellas.

En la Figura §7.3 se representan *slots* de dos clases `título` y `autor` que están caracterizadas por dos características numéricas f1 y f2. Si usamos técnicas de una única clase para caracterizarla, obtendríamos algo parecido a las líneas punteadas que rodean a los *slots* de ambas clases. En cambio, si aplicamos clasificadores multiclase, generaríamos las fronteras que se representan mediante líneas sólidas. Si en un instante determinado a los clasificadores les llega un *slot* que pertenece a la clase `precio`, de los que no existen ejemplos en la fase de entrenamiento, el comportamiento de ambos tipos de clasificadores es distinto. Un clasificador multiclase comprobaría en qué zona de la frontera está y lo clasificaría como alguna de las clases conocidas, aunque claramente sea diferente. En cambio, el clasificador de una única clase comprobaría si dicha instancia está dentro de lo conocido (en nuestro caso dentro de las eclipses discontinuas), en caso de no estarlo etiquetará el *slot* como *outlier* y no le asignará ninguna clase.

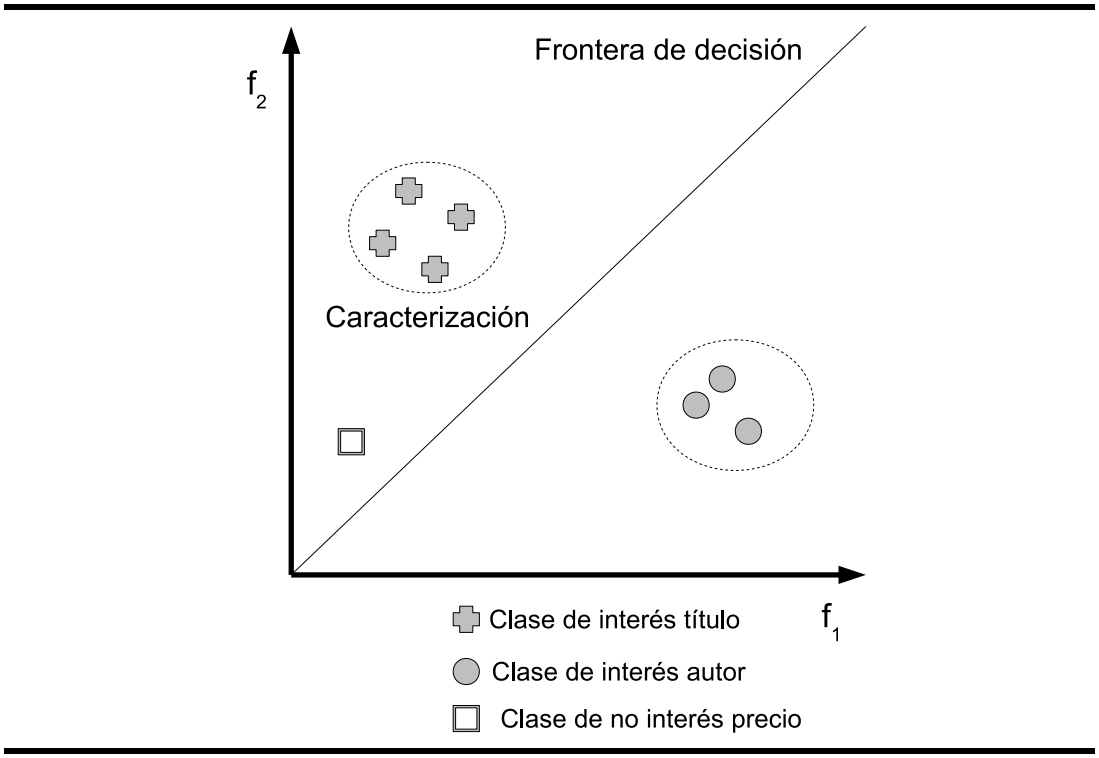


Figura 7.3: *Clasificadores multiclase vs. clasificadores de una clase*

Este último es el comportamiento que esperamos de un verificador, que sepa distinguir lo que conoce respecto a lo que no conoce, sin que necesariamente tenga que tener instancias de esto último.

7.3. Sistemas de verificación basados en clasificadores de una clase

Una vez que hemos justificado la aplicación de técnicas de una única clase para resolver el problema de la verificación de información, pasamos a detallar los pasos a realizar para crear sistemas de verificación que usen estas técnicas. En concreto, explicaremos cómo construir el modelo de verificación (Sección §7.3.1) y cómo diseñar el verificador para comprobar si nuevos elementos siguen el modelo de verificación (Sección §7.3.2).

Durante este capítulo asumimos que todos los *slots* son validos por lo que $\delta = 1$, el conjunto de entrenamiento TS sólo tiene un conjunto de datos RS y el conjunto $\mathcal{F}_{RS}$ está vacío (no hay características de conjunto de da-

tos) por lo que $\mathcal{F} = \mathcal{F}_s$. Por tanto $TS = \{(w, q_1, RS_1, 1)\}$, $X_{TS} = \{(w, q_1, X_{RS_1}, 1)\}$ y $X_{RS} = \{(X_{s1}, l_1), (X_{s2}, l_2), ..., (X_{s|RS|}, l_{|RS|})\}$

7.3.1. Creación del modelo de verificación

En la Figura §7.4 se ilustran los pasos a realizar para aplicar las técnicas de clasificación de una única clase al problema de la verificación de información. Estos pasos son:

- Paso 1: División del conjunto de entrenamiento en subconjuntos (Sección §7.3.1.1): Obtendremos tantos subconjuntos de entrenamiento como roles tengamos en $\mathcal{L}$, cada uno de ellos se usará para construir un clasificador.
- Paso 2: Aplicación de técnicas de reducción de características (Sección §7.3.1.2): La gran dimensionalidad y cardinalidad de los conjuntos de entrenamiento hace necesario aplicar este tipo de técnicas para lograr mejores clasificadores.
- Paso 3: Construcción del clasificador (Sección §7.3.1.3): Una vez que tenemos los subconjuntos de entrenamiento procedemos a construir los clasificadores, uno por rol.

7.3.1.1. División del conjunto de entrenamiento en subconjuntos

Tras obtener la caracterización del conjunto de datos, X_{RS}, (Sección §5.5.1) lo dividimos en tantos subconjuntos como roles distintas tengamos, en total tendremos $|\mathcal{L}|$ subconjuntos cada uno de ellos compuesto por ejemplos de la misma clase:

$$X_{RS,k} = \{(X, c) \in X_{RS} \mid c = l_k\}, \; k = 1 \ldots |\mathcal{L}| \tag{7.1}$$

donde X es la caracterización de cada uno de los slots, y c es la clase que tiene asignada.

Es importante resaltar, que a diferencia de las propuestas descritas en [100, 105, 115], el conjunto de características que se puede aplicar no está preestablecido. Por tanto, se puede elegir cualquiera de las técnicas discutidas en la Sección §5.5.1.

En el caso de las características cuyo ámbito de aplicación es el conjunto de datos, sus valores se añadirán a los valores de las características aplicadas a cada uno de los *slots* de dicho conjunto. Por tanto, los valores de estas características serán los mismos para todos los *slots* del conjunto de datos.

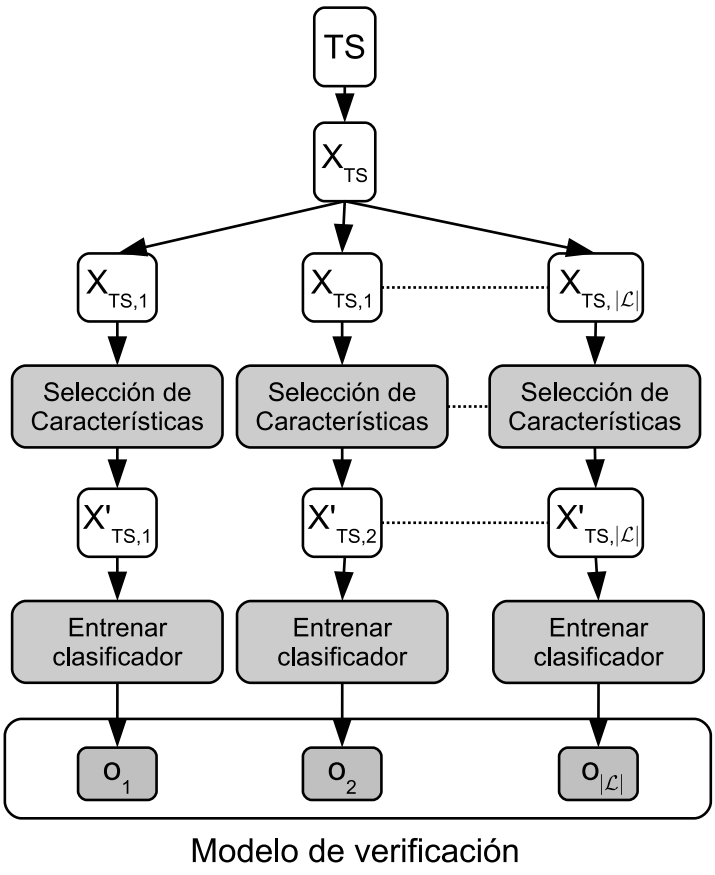


Figura 7.4: *Pasos para construir un modelo de verificación basado en clasificadores*

7.3.1.2. Aplicación de técnicas de reducción de características

Para construir modelos de verificación robustos, en la Sección §5.5.2 se proponía la fase de preprocesamiento. Esta fase cobra especial relevancia en los sistemas de verificación basados en clasificadores de una única clase ya que, como acabamos de indicar, tratamos con espacios de una alta dimensionalidad (al poder aplicar cualquier número y tipo de característica). Es, por tanto, especialmente importante aplicar alguna de las técnicas de selección de características descritas en [49, 109, 152] y tratadas en la Sección §4.6. Así, obtendremos una nueva caracterización del conjunto de datos:

$$X'_{RS,k} = \{(R(X_i), k) \mid (X_i, k) \in X_{RS,k}, k = 1..|\mathcal{L}|\} \tag{7.2}$$

donde

$$R(X_i) = [x_{i,f_1}, x_{i,f_2}, \ldots x_{i,f_d}] \mid d \leq |\mathcal{F}| \tag{7.3}$$

y $[f_1, f_2, \ldots, f_d]$ son las características seleccionadas por las técnicas de selección de características.

Observe que el conjunto concreto de características varía por clase, extremo éste no considerado por el resto de los sistemas actuales.

Además de la selección de características se realizan otras tareas de preprocesamiento. Primero, teniendo en cuenta la calidad de los modelos generados por los clasificadores de una clase si el conjunto de entrenamiento tiene *outliers*, se han aplicado técnicas de detección de *outliers*. En concreto, hemos usado una técnica basada en el test χ^2 [54]. Gracias a ella conseguíamos que, por ejemplo, *slots* con el valor `hhd/kk`, etiquetados erróneamente por el experto como `url`, formen parte del conjunto de entrenamiento que se usará para construir el clasificador que distingue `url`. Además, se han eliminado valores duplicados pero siempre manteniendo el mínimo necesario para que la fase de entrenamiento fuera exitosa. Esto se debe a que durante la experimentación constatamos que ciertos *slots* de una clase (por ejemplo de la clase `temperatura`) se caracterizaban de la misma forma, esto es, todos los vectores de características eran iguales. Al eliminar valores duplicados el conjunto de entrenamiento era unitario lo que impedía el proceso de aprendizaje de los clasificadores.

7.3.1.3. Construcción del clasificador

Cada uno de los conjuntos $X'_{RS,k}$ se usan como conjunto de entrenamiento para construir un clasificador o_k. Este clasificador estimará la posibilidad de que un *slot* s pueda ser etiquetado con la clase k. Cada o_k, para $k = 1..|\mathcal{L}|$, se define como:

$$\begin{array}{rccc} o_k: & \mathbb{R}^d & \longrightarrow & \mathbb{R} \\ & [x_{i,1} \ldots x_{i,d}] & \longmapsto & \{0,1\} \end{array} \tag{7.4}$$

Esto es, nuestro modelo de clasificación se basa en una colección de $|\mathcal{L}|$ clasificadores de una clase, donde o_k etiquetará como *target* todos aquellos *slots* pertenecientes a la clase *k* y como *outlier* al resto de *slots*. Son estas funciones o_k las encargadas de realizar el perfilado de cada una de estas características. Con esto estamos usando más de un modelo de verificación, y no solo uno que es uno de los puntos débiles de las técnicas usadas hasta la fecha.

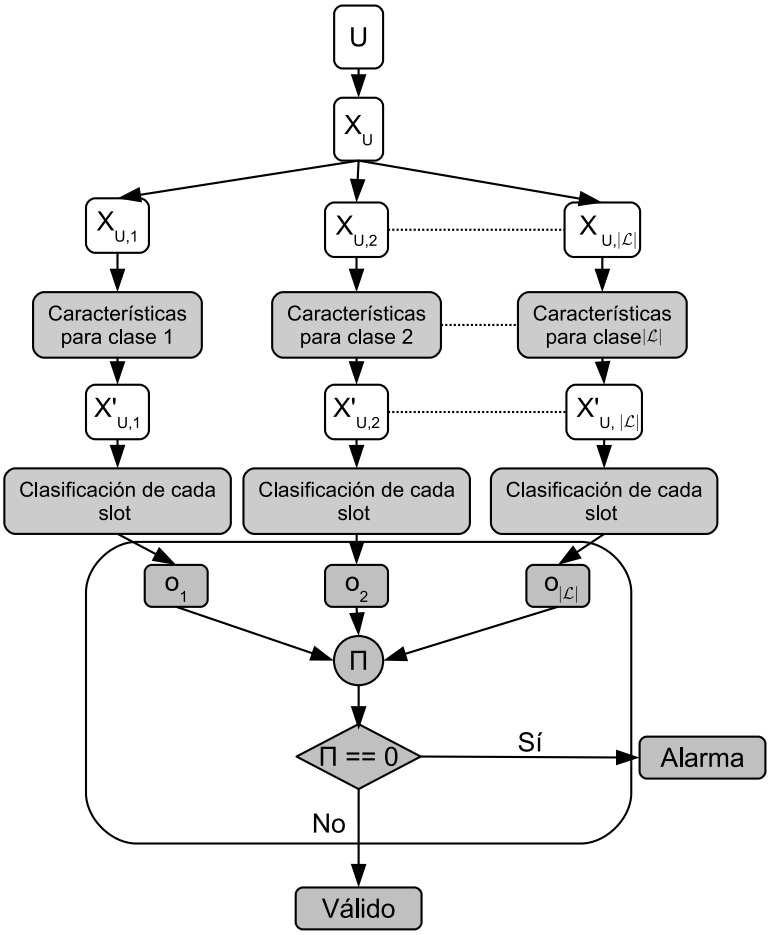


Figura 7.5: *Pasos para verificar información usando clasificadores*

7.3.2. Aplicación del verificador

En la Figura §7.5 se resumen los pasos a seguir para aplicar el modelo de verificación creado en el apartado anterior.

Durante la fase de producción, el *wrapper* extrae una serie de conjuntos de datos no verificados U, que tienen asignado un rol(clase) que debe ser verificada. Inicialmente, los distintos *slots* de U se agrupan por la clase que les asignó el wrapper, dando lugar a los subconjuntos $U_1, U_2, ...U_{|\mathcal{L}|}$. A cada uno de los *slots*, de cada una de las clases, los caracterizamos usando las características seleccionadas por las técnicas de reducción de características para su clase.

Una vez que tenemos caracterizados todos los *slots*, pasamos a clasificarlos usando el clasificador construido para cada clase. Esto es, los *slots* de la clase 1 se clasifican con o_1, los de la clase 2 con o_2 y así sucesivamente. Cada vez que clasifiquemos un *slot*, las funciones o_i devolverán un 1 si el elemento es *target* y 0 en otro caso.

Finalmente, el conjunto de datos no verificado, U, será clasificado como de interés (o válido) o de no interés (o inválido) aplicando la siguiente fórmula:

$$Y = \prod_{i=1}^{|\mathcal{U}|} m(s,k) \; \forall \, (s,k) \in U \; \wedge k \in \mathcal{L} \tag{7.5}$$

donde

$$m(s,k) = o_k(X_s) \tag{7.6}$$

Si $Y = 0$ entonces, al menos, un *slot* fue etiquetado como *outlier* y, por tanto, se debería lanza una alarma. En otro caso, U será etiquetado como válido.

Estableciendo una relación con los distintos elementos que forman parte del proceso de verificación descritos en el Capítulo §5, observamos que la función Y es la función de puntuación, comentada en la sección §5.5.3.

7.4. MAVE

MAVE (Multilevel WrApper VErification System) es un sistema de clasificación multinivel que verifica la información obtenida mediante *wrappers* tratando de forma independiente a las características categóricas y las numéricas. Su modelo de funcionamiento se divide en dos grandes pasos:

- **Primera fase**: verifica los *slots* usando únicamente características categóricas. Para tal fin, emplea un nuevo concepto denominado firma binaria.
- **Segunda fase**: aquellos *slots* que no han podido ser verificados en la primera fase, son cotejados usando clasificadores de una única clase que han generado los modelos de verificación .

Esta división en el proceso de selección en basa en los estudios sobre la influencia que tenía la naturaleza de las características sobre los modelos

de verificación construidos mediante clasificadores de una clase realizados en [43, 44]. Para ellos se usaron 27 bases de datos obtenidas de diversos sitios web. Cada uno de estas bases de datos se generó usando únicamente características categóricas, luego sólo numéricas y, finalmente, ambas. Las bases de datos resultantes se usaron para construir clasificadores de una única clase. Gracias a la aplicación de una serie de test no paramétricos, en dicho estudio se demostró que los clasificadores de una clase no alcanzaban unos buenos resultados si el conjunto de entrenamiento usado para construir el clasificador se basaba únicamente en características categóricas, aunque el poder de caracterización de ellas fuera muy alto.

En las secciones siguientes detallaremos en qué consisten cada uno de estas dos fases.

7.4.1. Construcción del modelo de verificación de MAVE

El sistema de verificación MAVE se construye usando los pasos descritos en la Figura §7.6. Los pasos son parecidos a los vistos en la Sección §7.3.1 salvo el tratamiento que se hace de las características numéricas y categóricas.

7.4.1.1. División del conjunto de entrenamiento en subconjuntos

Partiendo de la caracterización y división por clases del conjunto de datos, cada $X_{RS,k}$ se divide en dos conjuntos disjuntos: $X^{\mathcal{N}}_{RS,k}$ y $X^{\mathcal{C}}_{RS,k}$. $X^{\mathcal{N}}_{RS,k}$ está compuesto por características numéricas de los slots etiquetados con el rol r_k:

$$X^{\mathcal{N}}_{RS,k} = \left\{(X^{\mathcal{N}}_1, l_1), (X^{\mathcal{N}}_2, l_1), \ldots (X^{\mathcal{N}}_{|RS|}, l_{|RS|})\right\} \tag{7.7}$$

donde $l_1 = l_2 = ... = l_{|RS|} = r_k \in \mathcal{L}$,

$$\begin{aligned} X^{\mathcal{N}}_i &= \left[x_{i,1}, x_{i,2}, \ldots, x_{i,|N|}\right] = \\ &= \left[f_1(a_i),\ f_2(a_i), \ldots, f_{|\mathcal{N}|}(a_i)\right], \end{aligned} \tag{7.8}$$

y $\left(f_1, f_2 \ldots f_{|\mathcal{N}|}\right) \in \mathcal{N}$.

Por otro lado, $X^{\mathcal{C}}_{RS,k}$ lo forman las características categóricas de los slots etiquetados con el rol r_k:

$$X^{\mathcal{C}}_{RS,k} = \left\{(X^{\mathcal{C}}_1, l_1), (X^{\mathcal{C}}_2, l_2), \ldots (X^{C}_{|\mathcal{C}|}, l_{|RS|})\right\} \tag{7.9}$$

donde $l_1 = l_2 = ... = l_{|RS|} = r_k \in \mathcal{L}$,

$$\begin{aligned} X^{\mathcal{C}}_i &= \left[x_{i,1}, x_{i,2}, \ldots, x_{i,|\mathcal{C}|}\right] = \\ &= \left[f_1(a_i),\ f_2(a_i), \ldots, f_{|\mathcal{C}|}(a_i)\right], \end{aligned} \tag{7.10}$$

y $\left(f_1, f_2 \ldots f_{|\mathcal{C}|}\right) \in \mathcal{C}$.

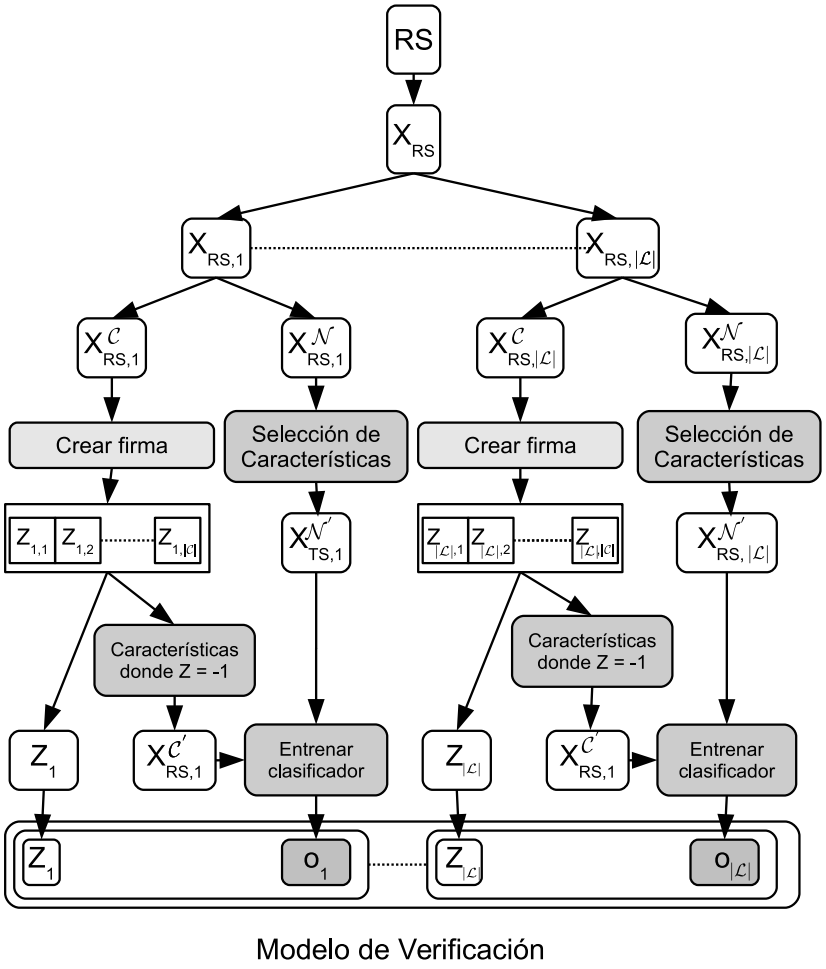


Figura 7.6: *Construcción del modelo de verificación de MAVE*

7.4.1.2. Aplicando técnicas de selección de características

MAVE, al considerar únicamente características numéricas para construir los clasificadores de una clase, sólo aplica la reducción de características a las características numéricas.

Por tanto, cada $X^{\mathcal{N}}_{RS,k}$ se reduce aplicando:

$$X'^{\mathcal{N}}_{RS,k} = \{(R(X_i), k) \mid (x_i, k) \in X_{RS,k}\} \tag{7.11}$$

donde:

$$R(X_i) = [x_{i,f_1}, x_{i,f_2}, \ldots x_{i,f_d}] \mid i \leq |\mathcal{RS}| \wedge d \leq |\mathcal{N}| \tag{7.12}$$

y $[f_1, f_2, \ldots, f_d]$ son el conjunto de características numéricas seleccionadas por las técnicas de reducción.

En concreto, MAVE usa **la técnica de reducción de puntuación laplaciana** [153] (Sección §4.6).

7.4.1.3. Entrenamiento de la firma

Para cada $X^{\mathcal{C}}_{RS,k}$ definimos una firma binaria Z_k como:

$$Z_k = Z(X^{\mathcal{C}}_{RS,k}) = \left[z_{k,1}, z_{k,2}, z_{k,3} \ldots z_{k,|\mathcal{C}|}\right]$$

$$z_{k,i} = \begin{cases} x_{i,1} & , \text{ if } \forall x_{i,j} \in X^{\mathcal{C}}_{RS,k},\ x_{i,j} = x_{i,1} \\ -1 & , \text{ En otro caso} \end{cases}$$

Esto es, la firma binaria de los valores de una característica categórica será -1 si dichos valores no coinciden en todas las instancias. Si todos los valores son iguales entonces la firma será igual al valor.

Por ejemplo, supongamos que $X^{\mathcal{C}}_{RS,1} = \{(1,1,0),(1,1,0),(1,0,0)\}$ y $X^{\mathcal{C}}_{RS,2} = \{(1,1,1),(1,0,0)\}$. Entonces, $Z_1(X^{\mathcal{C}}_{RS,1}) = [1,-1,0]$ y $Z_2(X^{\mathcal{C}}_{RS,2}) = [1,-1,-1]$. Tenemos que interpretar $Z_1(X^{\mathcal{C}}_{RS,1})$ como: todas las instancias de la clase 1 tienen como valor de la primera y última característica categórica el valor de 1 y 0 respectivamente. Por otro lado, el valor de la segunda característica categórica no es siempre el mismo. De forma similar, el significado de $Z_2(X^{\mathcal{C}}_{RS,2})$ es: después de aplicar la primera característica categórica a todos los *slots* etiquetados con la clase 2, podemos ver que esta característica tiene el valor de 1 para todos los *slots*. En cambio, después de aplicar la segunda y tercera características al mismo conjunto, vemos que los valores de estas características varían entre un *slot* y otro.

7.4.1.4. Construyendo el clasificador OCC

Después de construir las firmas binarias, para cada clase k, definimos $X^{\mathcal{C}'}_{RS,k}$ como todas las características categóricas cuyo $z_{k,i} = -1$ para $i = 1 \ldots |\mathcal{C}|$. Esto es, $X^{\mathcal{C}'}_{RS,k}$ está formado por las características que no siempre han devuelto el mismo valor para todos los *slots* de la misma clase k presentes en el conjunto de datos RS.

A continuación, inferimos el clasificador de una única clase o_k usando como conjunto de entrenamiento los valores de $X^{\mathcal{C}'}_{RS,k}$ y $X^{\mathcal{N}'}_{RS,k}$. Es decir, para todos

los *slots* del conjunto de entrenamiento sólo consideraremos las características numéricas seleccionadas por las técnicas de selección de características y aquellas categóricas que no serán consideradas durante la fase de validación de la firma binaria.

La razón por la que se añade este último tipo de características, es para evitar perder su información descriptiva: una variable que se comporta inútilmente por ella misma puede proporcionan una mejora significativa cuando se combina con otras [81]. Los resultado nos indican que la inclusión de este tipo de características no influye negativamente en el rendimiento del clasificador respecto a otro que sólo incluya características numéricas.

Por tanto, para MAVE el perfilado de cada una de las características es una combinación del valor de la firma y el clasificador de una clase que tenga asociado.

7.4.2. Aplicando el clasificador MAVE

Los pasos que usa MAVE para verificar elementos se muestran en la Figura §7.7. En el primer paso, para cada *slot* u perteneciente al conjunto por verificar U, se calculan los valores de todas las características categóricas, $X_u^{\mathcal{C}}$.

A continuación, se comparan $X_u^{\mathcal{C}}$ y Z_k siendo k la clase con la que está etiquetado el *slot* u. Ya que Z_k y $X_u^{\mathcal{C}}$ tienen el mismo tamaño, para cada posición i en la que $Z_{k,i} \neq -1$, para $i = 1..|\mathcal{C}|$, comparamos $Z_{k,i}$ y $X_{u,i}^{\mathcal{C}}$. Si alguna de las posiciones no es igual, entonces el clasificador MAVE marcará el *slot* como *outlier* o inválido (devuelve un 0) ya que u es diferente al resto de ejemplares de su misma clase.

Por ejemplo, si $X_u^{\mathcal{C}} = \{(1,1,1)\}$, u está etiquetado con l_1 y $Z_1 = \{(1,1,-1)\}$ entonces $X_u^{\mathcal{C}}$ es igual a Z_1 ya que ambos vectores son iguales posición a posición salvo en la último que no se considera al tener un valor de -1 para Z_1. En cambio si $X_u^{\mathcal{C}} = \{(1,0,0)\}$ entonces $X_u^{\mathcal{C}}$ no lo podemos considerar igual que Z_1 ya que difieren en la segunda posición.

Si $X_u^{\mathcal{C}} = Z_k$, actuamos tal y como se indicó en la Sección §7.3.2: calculamos $X_u^{\mathcal{N}'}$ y $X_u^{\mathcal{C}'}$ e intentamos hacer la verificación usando el clasificador de una clase o_k localizado en el segundo nivel. Si el clasificador o_k etiqueta al *slot* como *outlier* entonces el *slot* es inválido

Finalmente, el conjunto sin verificar, U, será clasificado como válido o inválido usando la siguiente función de puntuación:

$$Y = \prod_{i=1}^{|U|} m(x_i, k) \; \forall \; (x_i, k) \epsilon U \tag{7.13}$$

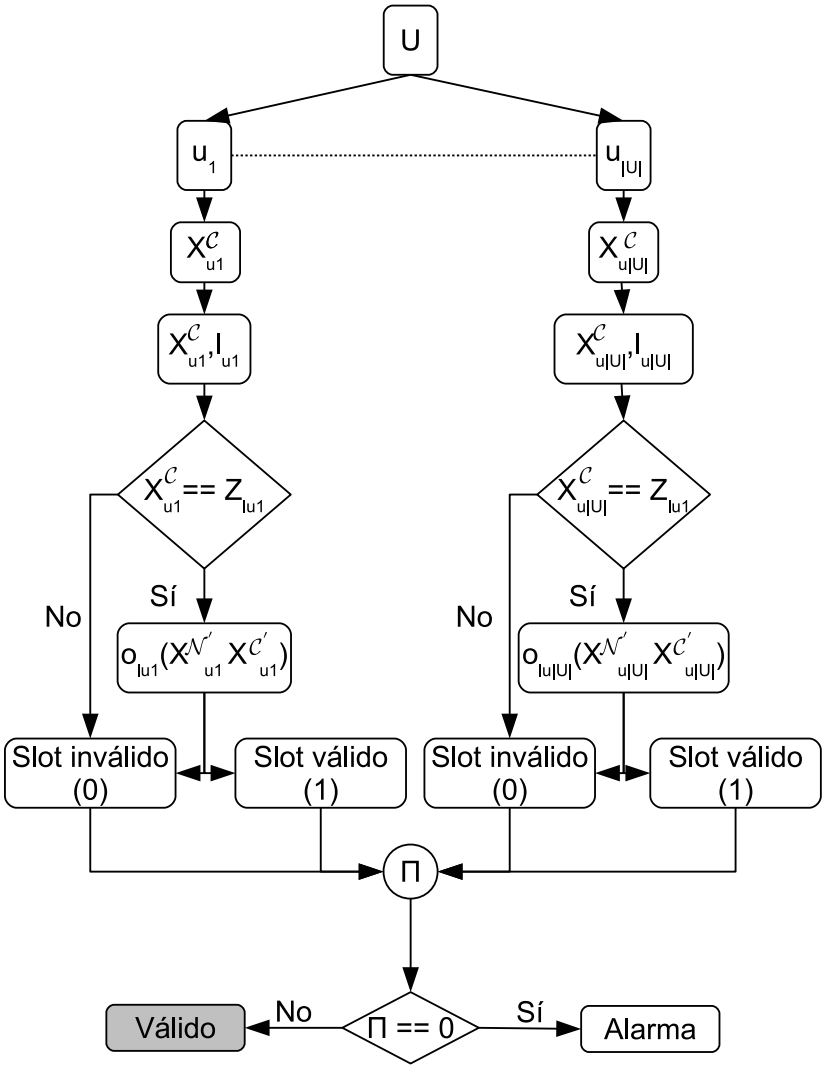


Figura 7.7: *Aplicación del verificador propuesto por MAVE*

donde

$$m(x,k) = \left(Z_k == X_x^C\right) \quad \text{and} \quad o_k(X_x^{\mathcal{N}'}, X_x^{e\prime}) \tag{7.14}$$

Si $Y = 0$ uno de los *slots* ha sido etiquetado como *outlier* y se lanza una alarma. En cualquier otro caso, el conjunto es válido.

El objetivo del primer nivel es descartar, rápidamente, aquellos *slots* que han sido etiquetados con una determinada clase y son claramente distintos de

todos los *slots* de la misma clase que formaban parte del conjunto de entrenamiento. Si el *slot* se parece a lo conocido, entonces usamos el segundo nivel para refinar la predicción usando el resto de características. Si establecemos una relación con los componentes del marco de trabajo discutido en el Capítulo §5, ese segundo nivel se puede ver como un filtro (Sección §5.6) pero cuyo objetivo no es silenciar falsas alarmas sino evitar falsos positivos.

7.5. Resumen

En este capítulo se han presentado los pasos a seguir cuando el problema de la verificación de *wrapper* se afronta desde el punto de vista de la clasificación de una única clase. Además de establecer el proceso general, se han propuesto dos nuevos sistemas de verificación basadas en este tipo de técnicas. El primero enfatiza la necesidad de usar algoritmos de selección de características para lograr modelos de clasificación robustos. El segundo, afronta la necesidad de tratar de forma independiente las características categóricas y las numéricas en un único sistema de clasificación multinivel denominado MAVE.

Capítulo 8

Experimentación y Resultados

Son vanas y están plagadas de errores las ciencias que no han nacido del experimento, madre de toda certidumbre.

Leonardo Da Vinci, Pintor, escultor e inventor italiano (1452-1519)

Para validar el diseño de nuestros sistemas de verificación, y poder compararlos con los ya existentes, hemos definido un marco de experimentación común. Éste será aplicado a lo largo de un conjunto de experimentos diseñados para justificar el buen comportamiento de los nuevos sistemas de verificación propuestos. Este capítulo se estructura de la siguiente manera: en la Sección §8.1 introduciremos la necesidad de crear un marco de experimentación; en la Sección §8.2 describiremos el marco de experimentación que hemos seguido; en la Sección §8.3 enumeraremos el conjunto de algoritmos estudiados; en la Sección §8.4 presentaremos los experimentos realizados indicando, para cada uno de ellos, los resultados alcanzados, un estudio comparativo de los algoritmos estudiados y una breve discusión sobre las conclusiones a las que se ha llegado tras realizar el experimento. Por último, en la Sección §8.5 resumiremos los aspectos más importantes vistos en el capítulo.

8.1. Introducción

En los artículos existentes en el campo de la verificación, se proponen distintos métodos para evaluar el rendimiento de los sistemas de verificación. Estas formas de evaluar presentan diversos problemas a la hora de aplicarlas ya que parten de bases de datos distintas, el proceso no se detalla adecuadamente, no se indica cómo se han tomado las medidas y tampoco se propone un método de comparación entre ellas.

Así pues, es necesario definir una metodología que permita su comparación. En este capítulo propondremos un marco de experimentación que permitirá estudiar, en igualdad de condiciones, a todos los sistemas de verificación.

Además, expondremos los resultados obtenidos por los diferentes algoritmos de verificación que se han venido exponiendo a lo largo de esta disertación, los analizaremos y discutiremos sobre los mismos.

8.2. Marco de experimentación

El marco de experimentación que nos permitirá comparar los sistemas de verificación conlleva la definición de los siguientes elementos:

- Base de datos: el conjunto de entrenamiento y el conjunto de características que están disponibles para poder aplicarse (Sección §8.2.1).
- Cómo realizar los experimentos para tomar medidas (Sección §8.2.2).
- Qué medidas tomar (Sección §8.2.3).
- Cómo comparar las medidas (Sección §8.2.4).

8.2.1. Base de datos

En la literatura especializada encontramos distintas bases de datos [98, 105, 115] que nos permitirían comparar los sistemas de verificación estudiados. Las más reseñables son:

Sitio	Número de Roles	Sitio	Número de Roles
1	3	15	2
2	3	16	3
3	8	17	7
4	4	18	4
5	2	19	2
6	2	20	3
7	7	21	8
8	4	22	4
9	2	23	2
10	3	24	3
11	4	25	4
12	4	26	2
13	4	27	3
14	5	Total	102

Tabla 8.1: *Características de la base de datos usada en la experimentación*

- Base de datos de Kushmerick [98]: está compuesta por 27 sitios web, cada uno de los cuales fue elegido para representar a diferentes tipos de servicios de búsqueda existentes en Internet. Para cada uno de estos sitios se ejecutaron un total de 15 consultas. En total, dispone de 867 páginas de media por sitio, sumando un total de 23416 páginas.

- Base de datos de Lerman y otros [105]: monitorizaron un total de 27 *wrappers* distintos que representaban a 23 sitios web diferentes durante un periodo de 10 meses. Cada *wrapper* ejecutaba entre 15 y 30 consultas que se ejecutaban cada 7-10 días. Las consultas eran las mismas para todos los sitios web salvo para un sitio web que aceptaba consultas sobre fechas.

- Base de datos de McCann y otros [115]: está formada por 114 sitios web pertenecientes a 6 dominios diferentes. Los autores no indican el número de consultas que se hizo ni su frecuencia.

De las tres bases de datos, sólo las propuestas por Kushmerick [98] y Lerman y otros [105] están disponibles. De estas dos la segunda está incompleta, dependiendo del sitio web encontraremos más o menos información. Por tanto, para hacer la experimentación usaremos la base de datos propuesta por Kushmerick [98] que, aunque antigua, es la única que nos ha permitido reproducir, hasta cierto punto, los resultados ya publicados. En la Tabla §8.1 se detalla el número de roles que hay por cada uno de los sitios web que forman esta base de datos.

Para la caracterización de los distintos *slots* contenidos en la base de datos anterior, disponemos de las características numéricas y categóricas que se enumeran en las Tablas §8.2 y §8.3. Estas características son las usadas en [100, 105, 115] además de añadir bastantes de las características descritas en la Sección §5.5.1.

Cod	Descripción
f_1	Valida si el valor de un atributo es una URL
f_2	Valida si el valor de un atributo es una fecha
f3	Valida si el valor de un atributo es un porcentaje
f_4	Valida si el valor de un atributo es nulo
f_5	Valida si el valor de un atributo está escrito en minúscula
f_6	Valida si el valor de un atributo es un entero largo
f_7	Valida si el valor de un atributo es un número de 10 dígitos correspondiente a un código ISBN
f_8	Valida si el valor de un atributo es una dirección IP
f_9	Valida si el valor de un atributo acaba con un símbolo de puntuación
f_{10}	Valida si el valor de un atributo es una dirección de correo válida
f_{11}	Valida si el valor de un atributo es un doble
f_{12}	Valida si el valor de un atributo es un nombre de dominio
f_{13}	Valida si el valor de un atributo es una hora
f_{14}	Valida si el valor de un atributo representa una cantidad monetaria
f_{15}	Valida si el valor de un atributo es un número de tarjeta de crédito
f_{16}	Valida si el valor de un atributo comienza en mayúscula

Tabla 8.2: *Características categóricas aplicadas*

Cod	Descripción
f_{17}	Número de palabras en un atributo
f_{18}	Número de palabras en un atributo que estén escritas en mayúscula
f_{19}	Número de tokens en un atributo que estén escritos en mayúscula
f_{20}	Número de m-gramas en un atributo que estén escritos en mayúscula
f_{21}	Número de tokens que contiene un atributo
f_{22}	Número de n-gramas que contiene un atributo
f_{23}	Longitud media de las palabras contenidas en un atributo
f_{24}	Longitud media de los tokens contenidos en un atributo
f_{25}	Número de palabras contenidas en un atributo y escritas en minúscula
f_{26}	Número de tokens contenidos en un atributo y que están escritos en minúscula
f_{27}	Número de n-gramas contenidos en un atributo y que están escritos en minúscula
f_{28}	Densidad media de palabras que comiencen en mayúscula
f_{29}	Densidad media de palabras que empiecen en minúscula
f_{30}	Densidad media de tokens que comiencen en mayúscula
f_{31}	Número de tokens que comienzan en mayúscula
f_{32}	Número de n-gramas que comienzan en mayúscula
f_{33}	Densidad media de letras mayúsculas contenidas en un atributo
f_{34}	Longitud de un atributo
f_{35}	Densidad media de las letras minúsculas contenidas en un atributo
f_{36}	Densidad media de los dígitos contenidos en un atributo
f_{37}	Densidad media de las marcas HTML contenidas en un atributo
f_{38}	Densidad media de números flotantes contenidos en un atributo
f_{39}	Densidad media de los símbolos de puntuación contenidos en un atributo
f_{40}	Densidad media de los dígitos contenidos en un atributo

Tabla 8.3: *Características numéricas aplicadas*

Debemos resaltar que las características usadas no son significativas respecto a los algoritmos. Esto es, los nuevos sistemas de verificación propuestos en este trabajo parten de un conjunto de características lo suficientemente representativas y son los propios sistemas los que se encargar de elegir el subconjunto más adecuado para cada sitio y clase. Si el conjunto de características es pequeño o no caracteriza adecuadamente a los slots, los sistemas de verificación no obtendrán buenos resultados.

8.2.2. Evaluación de la experimentación

Cada vez que queramos evaluar la capacidad de verificación de cualquier sistema de verificación para un determinado sitio web, aplicamos el siguiente proceso de validación cruzada con k = 10:

1. Obtenemos todos los *slots* gracias a la ejecución del plan de recolección y los dividimos por roles. En adelante, estos roles serán considerados las clases de cada conjunto, y dicha clase será considerada la clase de interés. El resto de clases del mismo sitio web que no pertenecen al conjunto son consideradas clases de no interés.

2. Cada uno de estos subconjuntos se divide en diez particiones. Nueve de ellas se usarán para crear el modelo de verificación y, por tanto, estarán formados únicamente por elementos de la clase c. La partición restante se usará para hacer el test y estará formado por los *slots* de la clase c, no contenidos en las otras particiones, y el resto de *slots* que no pertenezca a la clase c. Para evaluar los resultados obtenidos en el test se usarán las medidas que se indican en la Sección §8.2.3.

3. Repetimos el proceso anterior diez veces más variando las particiones que se usan para entrenar y las que se usan para el test.

4. Al final de las diez iteraciones, nos quedamos con los valores medios obtenidos en la fase de test.

En la Figura §8.1 se describe este proceso suponiendo que en el sitio web existen tres clases de interés y que sólo vamos a hacer cuatro particiones en vez de diez. El resultado medio calculado nos sirve para evaluar la capacidad de verificación para cada clase de cada sitio web.

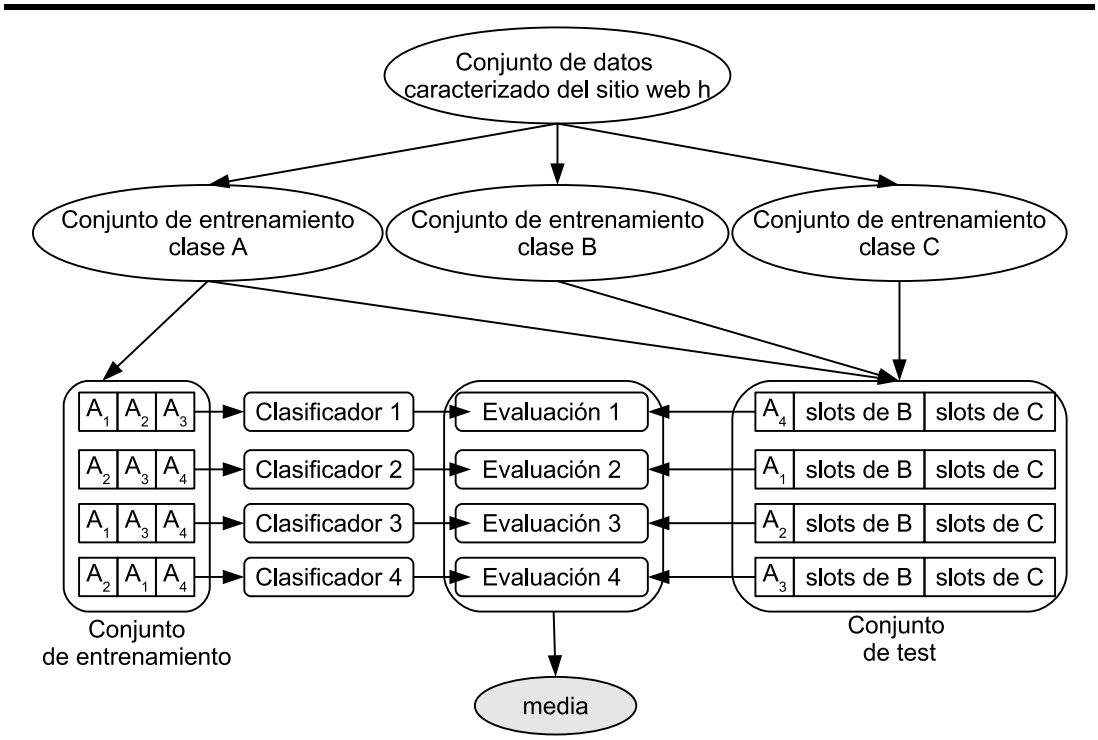


Figura 8.1: *Sistema de evaluación*

8.2.3. Medidas

Para cada ejecución del proceso visto anteriormente, calculamos una serie de medidas que nos servirán para determinar la bondad de verificación de cada sistema de verificación para cada clase y sitio web. Previo al cálculo de estas medidas es necesario obtener los valores que se muestran en la Figura §8.2 para cada una de la iteraciones de un proceso de 10 validación cruzada. Esta matriz se interpreta de la siguiente manera:

- *TA (Objetivos Aceptados)* es el número de *slots* que pertenecen a las clases de interés y han sido etiquetados correctamente por nuestro sistema.
- *TR (Objetivos Rechazados)* es el número de *slot* que pertenecen a las clases de interés pero que el sistema los detecto como de no interés y lanzo una alarma.

		Etiqueta esperada de la clase	
		Objetivo	Valor atípico
Etiqueta real de la clase	Objetivo	Objetivo Aceptado (TA)	Objetivo Rechazado (TR)
	Valor atípico	Valor atípico Aceptado (OA)	Valor atípico rechazado (OR)

Figura 8.2: *Matriz de confusión*

- *OA (Valores Atípicos Aceptados)* es el número de *slots* etiquetados como pertenecientes a una clase de no interés y que el sistema etiquetó como de interés.
- *OR (Valores Atípicos Rechazados)* es el número de *slots* etiquetados como pertenecientes a una clase de no interés y que el sistema también detectó como de no interés.

A partir de los valores calculados en la tabla de confusión, definimos las siguientes medidas:

- *Precisión* (P): $\frac{TA}{TA+OA}$
- *Cobertura* (R): $\frac{TA}{TA+TR}$
- *F-medida* (F): $2 \cdot \frac{precision \times cobertura}{precision+cobertura}$
- *Exactitud* (A): $\frac{TA+OR}{TA+OR+TR+OA}$
- *Área bajo la curva Roc* (AUC): $\frac{1+TAR-OAR}{2}$

donde:

- *TAR (Target Accepted Rate)*$= \frac{TA}{TA+TR}$ es la proporción de slots pertenecientes a una clase de interés que han sido detectados adecuadamente por el sistema.

- *OAR (Outlier Rejected Rate)*$= \frac{OA}{OA+OR}$ es el número de slots pertenecientes a una clase de no interés que no han sido adecuadamente detectados por el verificador.

El valor medio de estas medidas es el que se usará para comparar cada uno de los algoritmos estudiados.

Sin embargo, la medida de exactitud no es apropiada para nuestros conjuntos de datos caracterizados, porque el número de *slots* etiquetados como de interés es mucho menor que los *slots* clasificados de no interés y esta medida se basa en valores absolutos y no relativos. Es más, el objetivo de las medidas de precisión y cobertura suelen entrar en conflicto ya que cuando se incrementa el valor de *TA*, el número de *OA* también pueden incrementarse. Esto reduce el valor de la precisión [29]. Así pues, usaremos el valor de AUC como medida estándar para comparar la capacidad de verificación de los distintos algoritmos estudiamos.

8.2.4. Comparaciones

Para realizar las comparaciones entre algoritmos usaremos los test no paramétricos propuestos en [73, 74, 148]. Para ello primero aplicaremos el test de Kolmogorov-Smirnov para determinar si los valores de AUC a comparar no siguen una distribución normal y por tanto, no serían aplicables tests paramétricos. Este paso no es requerido ya que aunque el test nos indique que los datos siguen una distribución normal, los test no paramétricos son igualmente aplicables.

Cuando queramos hacer comparaciones uno a uno, aplicaremos el test de Wilcoxon Signed-Ranks Test [53] con un nivel de significación de 0.05.

Si, por el contrario, lo que queremos hacer son comparaciones múltiples entonces optaremos por usar el test Friedman Aligned-Ranks [90] y el post-hoc de Holm [91] con el nivel de significación de 0.05. El test de Friedman se usa cuando queremos buscar el mejor algoritmo de entre un conjunto. Este test es capaz de calcular las diferencias significativas entre algoritmos mientras que el post-hoc de Holm determinará qué algoritmos son significativamente diferentes y cuales no [74].

8.3. Algoritmos

Los algoritmos que se han estudiado en esta experimentación se enumeran en las Tablas §8.4, §8.5 y §8.6. Se han incluido los distintos grupos de sistemas de verificación estudiados a los largo de esta disertación:

1. Sistemas de verificación clásicos.
2. Sistemas de verificación basado en colonia de hormigas.
3. Sistemas de verificación basados en clasificadores de una clase. Los clasificadores de una clase seleccionados son ejemplos de cada de las categóricas de algoritmos de este tipo descritas en la Sección §4.5. Esto es, hemos aplicado tres algoritmos de densidad, tres de frontera y dos reconstructivos.
4. Sistemas de verificación basados en clasificadores de una clase sólo con características categóricas y sólo con numéricas.
5. Sistemas de verificación basados en MAVE.

8.3.1. Parámetros

Los algoritmos RAP, LER, MCC no tienen ningún parámetro específico salvo el establecimiento de umbrales. Para calcularlos, estableceremos el porcentaje de *slots* rechazados a un 1 % durante el proceso de construcción. Este parámetro determina el porcentaje de *slots* marcados como de interés que serán rechazados durante el proceso de construcción del verificador. En cambio, los sistemas basados en clasificados de una única clase descritos en el apartado anterior admiten los siguientes parámetros:

- En las distintas variantes del clasificador de Gauss fijaremos el valor de la regularización para estimar la matriz de covarianza a 0.2.
- Los clasificadores basados en los k-centros y en la kmedia usarán como prototipos todos los elementos del conjunto.
- Para los clasificadores basados en máquinas de soporte vectorial usaremos un núcleo gausiano con sigma = 2.
- Para los clasificadores PCA la fracción para explicar la varianza es 0.9.
- En los clasificadores que usan redes neuronales, el número de unidades en su capa oculta es de 5.
- El clasificador KNN calcula automáticamente el número de vecinos más cercanos a considerar y aplica el método kappa para evaluar la distancia entre vecinos.

Sistemas de verificación clásicos

#	Algoritmo	Descripción
1	LER	Propuesta de Lerman y otros [105]
2	RAP	Propuesta de Kushmerick [100]
3	MCC	Propuesta de McCann y otros [115]

Sistemas de verificación basados en colonias de hormigas (Capítulo §6)

#	Algoritmo	Descripción
4	BWAS	Sistema de verificación usando BWAS

Sistemas de verificación basados en OCC

#	Algoritmo	Descripción
5	G	Clasificador de una única clase gausiano (Sección §4.5.1.3)
6	P	Clasificador de una única clase de Parzen (Sección §4.5.1.4)
7	KC	Clasificador de una única clase basado en los K-centros (Sección §4.5.2.1)
8	KN	Clasificador de una única clase basado en los vecinos más cercanos (Sección §4.5.2.2)
9	KM	Clasificador de una única clase basado en las k-medias (Sección §4.5.3.1)
10	SVM	Clasificador de una única clase basado en SVM (Sección §4.5.2.3)
11	PCA	Clasificador de una única clase basado en análisis de componentes principales (Sección §4.5.3.2)
12	NN	Clasificador de una única clase basado en redes auto-encoder (Sección §4.5.3.3)

Variantes de sistemas de verificación basados en OCC (Capítulo §7)

13	G-ON	Clasificador de una única clase gausiano con características sólo numéricas
14	G-OC	Clasificador de una única clase gausiano con características sólo categóricas

Continúa en la página siguiente

Tabla 8.4: *Sistemas de verificación estudiados*

Variantes de sistemas de verificación basados en OCC (Capítulo §7)

15	P-ON	OCC de Parzen con características sólo numéricas
16	P-OC	OCC de Parzen con características sólo categóricas
17	KC-ON	OCC basado en los K-centros con características sólo numéricas
18	KC-OC	OCC basado en los K-centros con características sólo categóricas
19	KN-ON	OCC basado en los K-vecinos más cercanos con características sólo numéricas
20	KN-OC	OCC basado en los K-vecinos más cercanos con características sólo categóricas
21	KM-ON	OCC basado en las k-medias con características sólo numéricas
22	KM-OC	OCC basado en las k-medias con características sólo categóricas
23	SVM-ON	OCC basado en SVM con características sólo numéricas
24	SVM-OC	OCC basado en SVM con características sólo categóricas
25	PCA-ON	OCC basado en análisis de componentes principales con características sólo numéricas
26	PCA-OC	OCC basado en análisis de componentes principales con características sólo categóricas
27	NN-ON	OCC basado en redes auto-encoder con características sólo numéricas
28	NN-ON	OCC basado en redes auto-encoder con características sólo categóricas

Variantes del sistemas de verificación MAVE (Sección §7.4)

29	M-G	MAVE usando OCC Gausianos
30	M-P	MAVE usando OCC de Parzen
31	M-KC	MAVE usando OCC basados en los K-centros

Continúa en la página siguiente

Tabla 8.5: *Sistemas de verificación estudiados (continuación)*

Variantes del sistemas de verificación MAVE (Sección §7.4)

32	M-KN	MAVE usando OCC basado en los K-vecinos más cercamos
33	M-KM	MAVE usando OCC basados en las K-medias
34	M-SVM	MAVE usando OCC basados en Máquinas de soporte vectorial
35	M-PCA	MAVE usando OCC basados en análisis de componentes principales
36	M-NN	MAVE usando OCC basados en redes auto-encode

Tabla 8.6: *Sistemas de verificación estudiados (continuación)*

Estos valores no han sido especialmente elegidos para el problema de la verificación ya que uno de los objetivos de esta disertación es demostrar que este tipo de sistema son una alternativa a los ya existentes pero en ningún momento hemos buscado la mejor parametrización.

Parámetro	Dominio	Valor
número de hormigas	[3,100]	14
número de iteraciones	[5,1000]	968
τ_o	[0.0001, 0.1]	0.08
β	[0.0001, 10]	0.22
α	[0.0001, 5]	0.72
ρ	[0.01, 1]	0.16
σ	[2,7]	3
reinicio	[30 %, 70 %]	33 %
K	[1, 6]	1

Tabla 8.7: *Parámetros usados para el algoritmo BWAS*

Por último, el ajuste de parámetros para el algoritmos BWAS se ha hecho automáticamente usando iterativamente el método *F-Race* [9] sobre un conjunto de configuraciones candidatas. *F-Race* es un método de competición

que, en cada iteración, aplica cada una de las soluciones candidatas al problema combinatorio o a la función continua a optimizar. Si la configuración candidata es estadísticamente peor que las otras (usa el método de Friedman de dos vías del análisis de la varianza), es eliminado de la competición. *F-Race* finaliza cuando sólo queda una configuración candidata o ha finalizado el tiempo de la competición. Una vez obtenida la mejor configuración, se muestrea con el objetivo de obtener nuevas configuraciones candidatas alrededor de la mejor configuración obtenida hasta el momento. El proceso completo finaliza cuando se llegue al número máximo de repeticiones.

Durante el proceso de optimización de parámetros, se ha ejecutado un máximo de 2500 BWAS. El número de evaluaciones en cada ejecución fue de $2500 \times N$, donde N fue el número de clases contenidos en cada sitio web. Los rangos de los valores de cada uno de los parámetros a optimizar son los mismos definidos en [9, 64]. Las mismas páginas web descritas en [98], fueron muestreadas aleatoriamente para obtener subconjuntos de 1000 *slots*, usados como entradas en el proceso de entrenamiento.

Los valores de los parámetros tuneados para BWAS se muestran en la Tabla §8.7. La primera columna muestra el nombre del parámetro, la segunda el rango de valores muestreados y la tercera el valor elegido.

8.4. Experimentos abordados

Los experimentos que se han realizado se describen en las siguientes secciones. Se estudiaron un total de 36 algoritmos distintos para verificar los contenidos de 23416 páginas web provenientes de 27 sitios web. Además, para cada sitio web se generaron tantos modelos de verificación como clases distintas tiene cada sitio.

En total se han abordado 4 experimentos, cada uno de ellos orientado a estudiar y analizar el comportamiento de los sistemas de verificación tradicionales frente a las nuevas propuestas de verificación estudiadas a lo largo de esta disertación.

En cada experimento mostraremos la siguiente información:

1. **Objetivo:** describiremos brevemente los objetivos que perseguimos al realizar el experimento.
2. **Resultados:** Mostraremos una tabla resumen con los valores medios y varianza de la medida AUC para cada sitio web y algoritmo. Además, expondremos gráficamente estos resultados para facilitar su comprensión

3. **Análisis comparativo:** Tras exponer los resultados, compararemos cada uno de los sistemas de verificación estudiando aplicando los test no paramétricos descritos en la Sección §8.2.4.

4. **Discusión:** Para acabar cada apartado introduciremos una sección de discusión en la que analizaremos los resultados expuestos en los apartados anteriores.

Por último, y antes de pasar al análisis de cada uno de los algoritmo, indicar que todas las tablas y gráficos expuestos se han calculado a partir de las tablas de resultados incluidas en el Apéndice §A.

8.4.1. Experimento 1

El objetivo de este primer experimento es comparar las técnicas basadas en hormigas (Algoritmo BWAS) con los sistemas de verificación actuales (RAP, LER, MCC). Esto es, comprobar si podemos tratar el problema de la verificación de wrappers como un problema de optimización computacional.

Resultados y estudio comparativo

La Tabla §8.8 muestra los resultados obtenidos por cada uno de los algoritmos para cada uno de los sitios web que forman nuestra base de datos. En ella se indica el valor de AUC después de aplicar el verificador para casa sitio web. La Figura §8.3 muestran el valor medio de AUC normalizado a una escala logarítmica para apreciar mejor la diferencia entre cada verificador para cada uno de los 27 sitios web.

Si comparamos los resultados de los algoritmos sitio por sitio, vemos que el sistema de verificación basado en BWAS se comporta mejor que el resto de los sistemas de verificación clásicos salvo en los sitios 1, 15 y 26. En los dos primeros las diferencias son mínimas pero en el último, el algoritmo RAP obtiene un valor de AUC de 0.82 mientras que BWAS alcanza el 0.73. Si nos centramos en los valores medios globales, BWAS logra un valor de AUC de 0.91 frente al 0.84 del algoritmo propuesto por Lerman.

Si aplicamos los métodos estadísticos descritos en secciones anteriores para comparar los valores de AUC de la Tabla §8.8, primero debemos confirmar que estos no siguen una distribución normal. Esta premisa la comprobamos aplicando el test de Kolmogorov-Smirnov a un nivel de significación de 0.05. Los resultados de este test se pueden consultar en la Tabla §8.9).

w	BWAS	RAP	LER	MCC	w	BWAS	RAP	LER	MCC
1	0.86	0.87	0.50	0.50	**15**	0.83	0.86	0.50	0.50
2	0.94	0.76	0.50	0.50	**16**	0.92	0.89	0.50	0.50
3	0.93	0.83	0.51	0.51	**17**	0.98	0.90	0.56	0.56
4	0.90	0.82	0.63	0.63	**18**	0.98	0.96	0.63	0.63
5	0.97	0.94	0.50	0.50	**19**	0.98	0.79	0.50	0.50
6	0.58	0.50	0.50	0.50	**20**	0.84	0.83	0.50	0.50
7	0.92	0.90	0.64	0.64	**21**	1.00	0.95	0.74	0.73
8	0.96	0.83	0.63	0.63	**22**	0.99	0.88	0.50	0.50
9	0.96	0.93	0.50	0.50	**23**	0.99	0.84	0.63	0.63
10	0.93	0.92	0.67	0.67	**24**	0.96	0.92	0.50	0.50
11	0.94	0.86	0.50	0.50	**25**	0.99	0.87	0.50	0.50
12	0.86	0.81	0.63	0.63	**26**	0.73	0.82	0.50	0.50
13	0.88	0.82	0.63	0.63	**27**	0.99	0.86	0.50	0.50
14	0.77	0.63	0.50	0.50					
					Medias	0.91	0.84	0.55	0.55

Tabla 8.8: *Resultados de AUC obtenidos por el verificador BWAS y los verificadores clásicos*

Como los p-values están por debajo de 0.05, la hipótesis nula se rechaza y, por tanto, la premisa de normalidad no se satisface.

Para comprobar que, efectivamente, los valores del AUC logrados por BWAS son significativamente mejores a los del resto de algoritmos, hemos aplicado el test de Friedman (Tabla §8.10). El algoritmos que presenta un mejor ranking es, como cabría de esperar, BWAS. Al aplicar el post-hoc de Holm (columna 4) vemos que los valores están por debajo del 0.95 por lo que podemos concluir que el valor de ranking del algoritmo BWAS es significativamente mejor que el del resto de algoritmos a un nivel de significación del 0.05.

8.4.2. Experimento 2

El objetivo de este segundo experimento es comprobar si los sistemas de verificación basados en OCC mejoran a los sistemas de verificación clási-

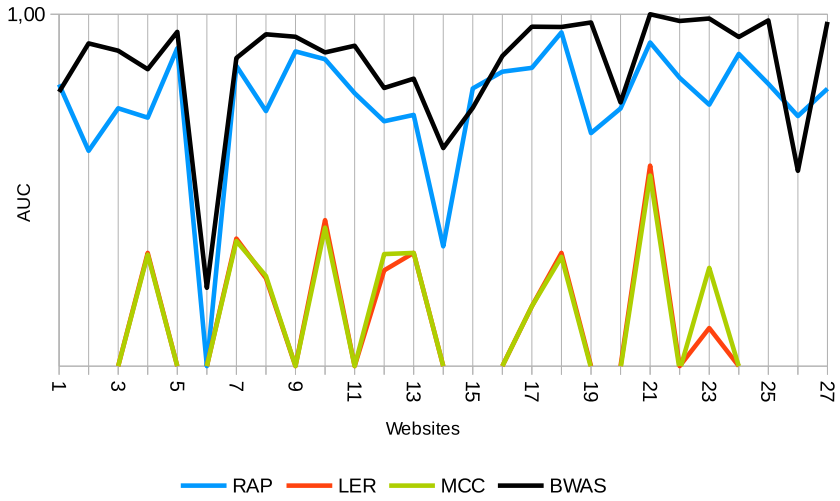


Figura 8.3: *Resultados obtenidos por BWAS y la técnicas clásicas*

Algoritmo	Estadístico	p-value
BWAS	0.43	1.74e-52
LER	0.14	9.41e-05
RAP	0.30	1.17e-23
MCC	0.47	5.84e-62

Tabla 8.9: *Resultados del test de Kolmogorov-Smirnov para BWAS y sistemas clásicos*

cos. Para ello, además de presentar los resúmenes de los datos de AUC, haremos un test comparativo por pares usando el test de Wilcoxon. Así pretendemos demostrar que cualquiera de la variantes OCC mejora a las técnicas tradicionales.

8.4.2.1. Resultados y estudio comparativo

La Tabla §8.11 muestra el resumen de los valores medios de AUC obtenidos por las distintas variantes de técnicas OCC estudiadas y los sistemas de verificación clásicos. Si comparamos los valores de AUC para los sistemas de verificación basados en OCC, vemos que el algoritmo P (basado en

Algoritmo	Ranking	pHolm
BWAS	1.43	
RAP	1.90	0
LER	3.25	0
MCC	3.43	0.012

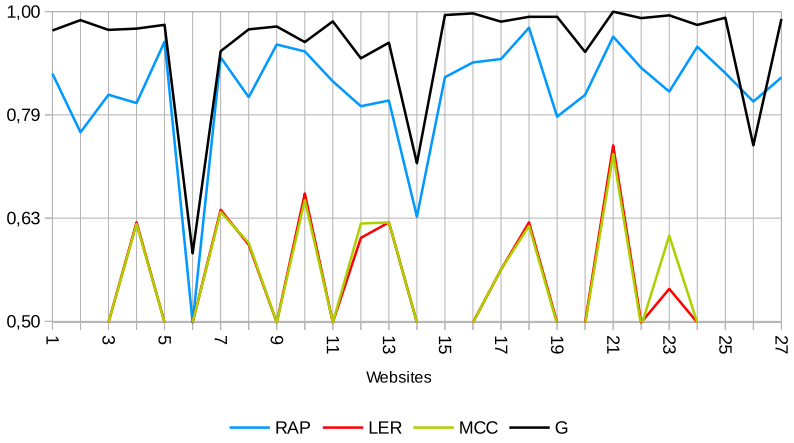
Tabla 8.10: *Resultados test Friedman para BWAS y sistemas clásicos*

Parzen) alcanza un valor de AUC de tan solo 0.73, que es incluso más bajo al alcanzado por el algoritmo RAP. El segundo peor algoritmo OCC es el SVM que logra unos resultados similares a RAP. El resto de algoritmos OCC logran unos resultados por encima del 0.92 lo que representa una mejorar de entorno a un 10 % respecto a las sistemas tradicionales.

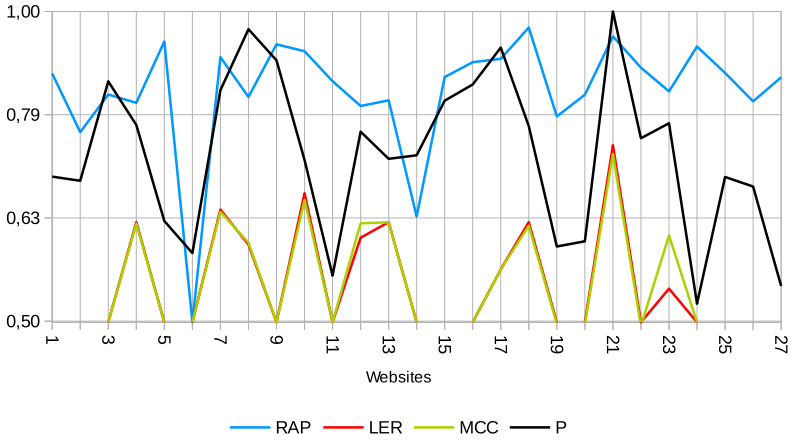
En las Figura §8.4, §8.5 y §8.6 detallamos la evolución de AUC para cada uno de los algoritmos usados y cada uno de los sitios web que forman nuestra base de datos. Salvo algún caso puntual, el del website 26, la mayoría de los sistemas de verificación basados en OCC funcionan mejor que cualquiera de los sistemas clásicos. En algunos casos, como el algoritmo P y SVM, son más los websites donde las técnicas clásicas mejoran a las basadas en OCC.

Algoritmo	AUC	Algoritmo	AUC
G	0.93	RAP	0.84
P	0.73	LER	0.54
KC	0.92	MCC	0.54
KNN	0.92		
KM	0.92		
SVM	0.82		
PCA	0.92		
NN	0.92		

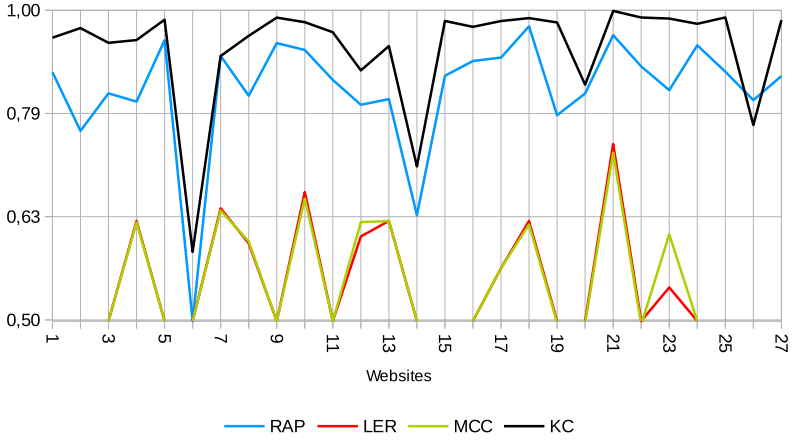
Tabla 8.11: *Valores de AUC logrados por las técnicas OCC y los sistemas de verificación clásicos*



(a) *Verificadores clásicos vs. Gauss*

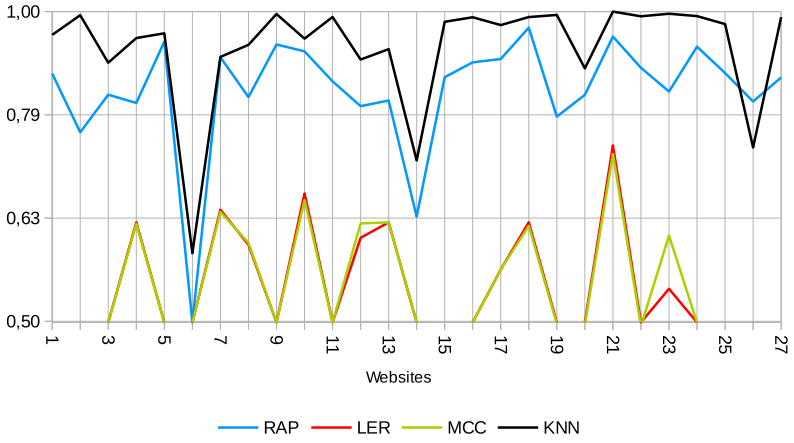


(b) *Verificadores clásicos vs. Parzen*

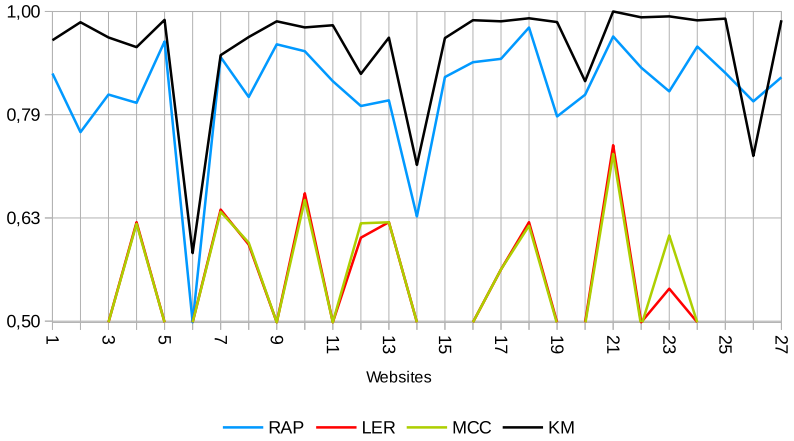


(c) *Verificadores clásicos vs. K-center*

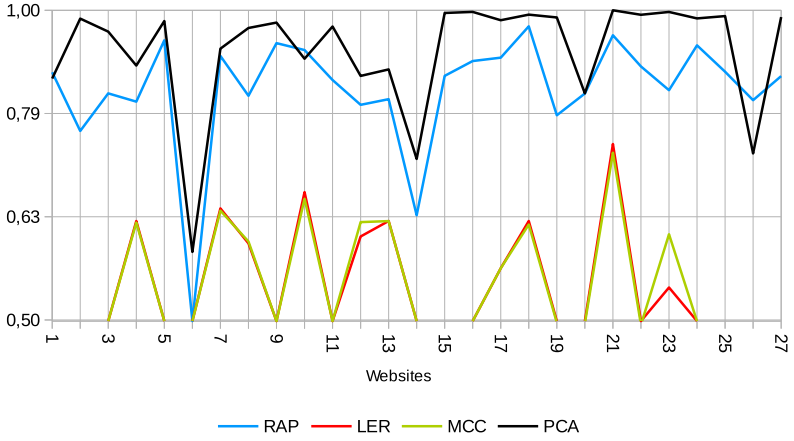
Figura 8.4: *Valores de AUC obtenidos por OCC y los sistemas de verificación clásico para cada sitio web (I)*



(a) *Verificadores clásicos vs. Knn*

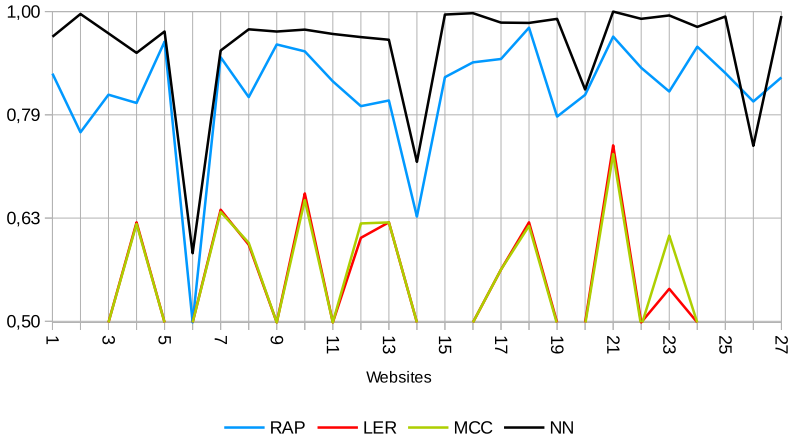


(b) *Verificadores clásicos vs. K-medias*

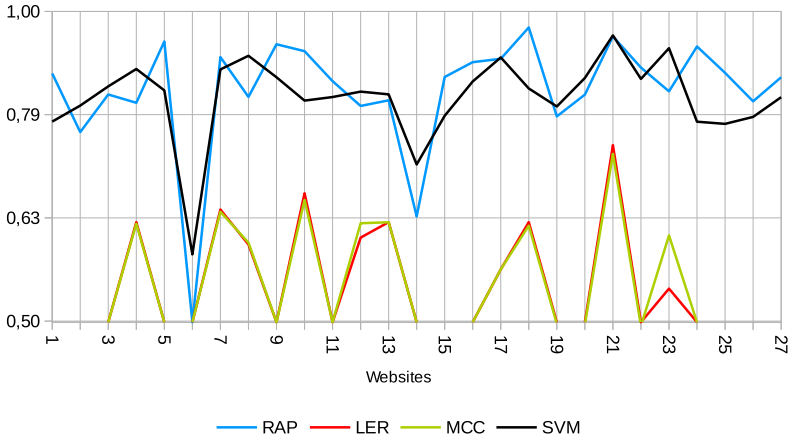


(c) *Verificadores clásicos vs. PCA*

Figura 8.5: *Valores de AUC obtenidos por OCC y los sistemas de verificación clásico para cada sitio web (II)*



(a) *Verificadores clásicos vs. Redes Neuronales*



(b) *Verificadores clásicos vs. SVM*

Figura 8.6: *Valores de AUC obtenidos por OCC y los sistemas de verificación clásico para cada sitio web (III)*

Como en la experimentación anterior, antes de aplicar los test no paramétricos para comprar los valores obtenidos por los distintos algoritmos aplicamos el test de Kolmogorov-Smirnov a un nivel de significancia de 0.05 para asegurar la no normalidad de los mismos. En la Tabla §8.12 podemos consultar los resultados de este test. Al estar los valores por debajo de 0.05 (salvo para SVM), la hipótesis nula se rechaza.

Para hacer la comparación por pares entres las técnicas clásicas y las basadas en OCC, aplicamos el test de Wilcoxon. Los resultados del test los

Algoritmo	Estadístico	p-value	Algoritmo	Estadístico	p-value
G	0.30	6.95e-25	RAP	0.13	0.00
P	0.23	4.63e-15	LER	0.50	2.87e-72
KC	0.29	6.54e-23	MCC	0.48	6.28e-68
KNN	0.28	1.02e-21			
KM	0.28	1.06e-21			
SVM	0.08	0.135			
PCA	0.33	3.33e-31			
NN	0.29	1.46e-23			

Tabla 8.12: *Resultados del test de Kolmogorov-Smirnov para los sistemas de verificación basados en OCC y las técnicas clásicas*

podemos consultar en la Tabla §8.13.

Si primero analizamos los resultados relacionados con los algoritmos LER y MCC, vemos que, en todos los casos, son peores que cualquiera de los algoritmos OCC (el valor de R^+ es superior al de R^- y el valor de $p-value$ es menor que 0.05). Si ahora nos centramos en el algoritmo RAP, vemos que sale perdedor en todas las comparativas salvo la que le enfrenta al algoritmo P (donde R^- es mayor a R^+ y el valor de $p-value$ es menor que 0.05) y con SVM (en este último caso hay un empate ya que el valor de $p-value$ es mayor que 0.05).

8.4.3. Experimento 3

El objetivo de este tercer experimento es estudiar la influencia que tiene el uso de características numéricas y/o categóricas sobre los sistemas de verificación basados en OCC. Esto es, en el apartado anterior usamos todas las características para generar un único modelo por clase sin preocuparnos por la diferencia de dominio existente entre características.

Para realiza este estudio, partiremos de tres tipos de conjuntos: conjuntos de entrenamiento caracterizados por todas las características, conjuntos de entrenamiento caracterizados por características numéricas y conjuntos de entrenamiento caracterizados por característica categóricas. Cada uno de estos pares de conjuntos será usado para construir los sistemas de verificación basados en OCC y validar su funcionamiento.

Test	R^+	R^-	Z-value	p-value	Test	R^+	R^-	Z-value	p-value
G vs RAP	3024	136	-7.0569	0	KM vs RAP	2992	168	-6.9005	0
G vs LER	3655	0	-8.0077	0	KM vs LER	3652	3	-7.9945	0
G vs MCC	3654	1	-8.0033	0	KM vs MCC	3652	3	-7.9945	0
P vs. RAP	803.5	2356.5	-3.7948	0	PCA vs RAP	2910	330	-6.1872	0
P vs. LER	3270	51	-7.578	0	PCA vs LER	3653	2	-7.9989	0
P vs. MCC	3446	40	-7.7318	0	PCA vs MCC	3652	3	-7.9945	0
KC vs RAP	2984	176	-6.8614	0	NN vs RAP	2666	574	-5.0169	0
KC vs LER	3655	0	-8.0077	0	NN vs LER	3680	61	-7.7917	0
KC vs MCC	3655	0	-8.0077	0	NN vs MCC	3680	61	-7.7917	0
KNN vs RAP	2987	173	-6.8761	0	SVM vs RAP	1564.5	2176.5	-1.3176	0.18
KNN vs LER	36541	1	-8.0033	0	SVM vs LER	3740	1	-8.0501	0
KNN vs MCC	3655	0	-8.0077	0	SVM vs MCC	3740	1	-8.0501	0

Tabla 8.13: *Resumen de los resultados del test de Wilcoxon aplicado a las técnicas OCC y las clásicas*

8.4.3.1. Resultados y estudio comparativo

En la Tabla §8.14 se muestran los resultados obtenidos por los OCC cuando el modelo de verificación se basa características numéricas y categóricas (columna segunda), sólo características numéricas *-ON (columna cuarta) y sólo características categóricas *-OC (última columna).

Si nos centramos en los valores de *-ON y *-OC, vemos que, en todos los casos, el valor de AUC obtenido por *-ON es mucho mejor salvo Parzen. Si observamos el comportamiento del algoritmo KC-*, vemos que, en este caso, la diferencia de AUC a favor de KC-ON llega a ser de un 16 %. Para todos los casos se cumple que ninguno de los algoritmos *-OC alcanza valores de AUC superiores al 0,90 por lo que podemos concluir que el uso de modelos de verificación basados sólo en características categóricas no son satisfactorios para el proceso de verificación.

Al comparar las columnas segunda y cuarta, vemos que las diferencias ya no son tan claras. Para algunos casos, como el algoritmo P, parece claro optar por un modelo formado únicamente por características categóricas. Para el resto de los casos no existe un ganador claro, lo que si es cierto es que, teniendo en cuenta los valores medios de AUC, ningún modelo basado en características numéricas y categóricas mejora al modelo basado en sólo características numéricas.

En las Figuras §8.7, §8.8 y §8.9 podemos comparar el comportamiento de los distintos sistemas de verificación basados en OCC. En cada gráfica se comparan los resultados de cada algoritmo si se usan todas las características, sólo las numéricas o sólo las categóricas. En general podemos decir que la linea vede, que es la que representa a los modelos de verificación creados únicamente con características categóricas, está por debajo del resto de líneas en casi todos los algoritmos y sitios. Sólo para el website 6 y 17, este tipo de modelos tiene un mejor comportamiento, independientemente del algoritmo utilizado. Para algunos algoritmos, como el KNN o el Kcenter, este tipo de modelos se comportan especialmente mal. Mención aparte merece el algoritmo de Parzen para el que los modelo basados en características categóricas experimentan un mejor comportamiento.

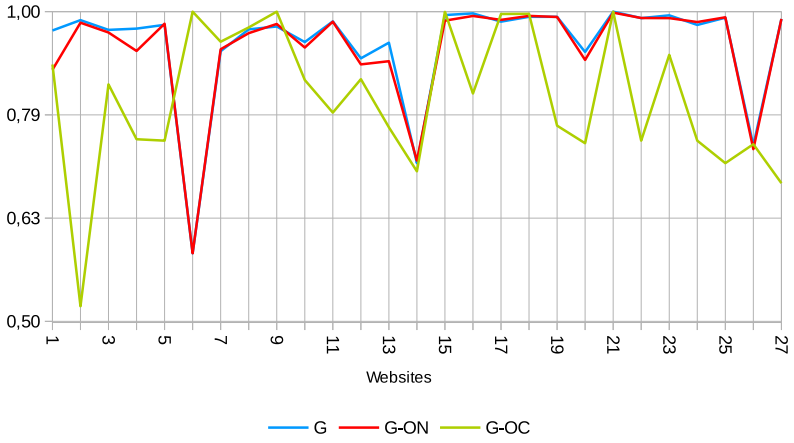
Respecto al uso de todas o sólo las características numéricas (lineas azul y roja) los resultados no son tan claros y no se aprecian diferencias entre ellos. Dependiendo del algoritmo y el sitio web evaluado, un modelo es mejor que el otro y presenta resultados prácticamente iguales.

Como en experimentos anteriores, aplicamos el test de Wilcoxon para hacer comparaciones por pares. Primero empezamos realizando el test de normalidad a los datos de AUC logrados por los distintos algoritmos (Tabla §8.15). Como en casos anteriores, la hipótesis nula se rechaza y, por tanto, procede la aplicación de test estadísticos no paramétricos.

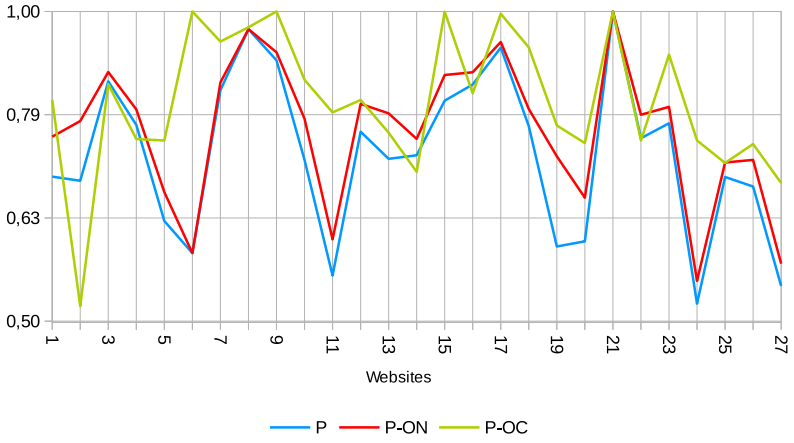
En la Tabla §8.16 mostramos los valores del test del Wilcoxon para ca-

Algoritmo	**AUC**	**Algoritmo**	**AUC**	**Algoritmo**	**AUC**
G	0.93	G-ON	0.93	G-OC	0.85
P	0.73	P-ON	0.80	P-OC	0.84
KC	0.92	KC-ON	0.92	KC-OC	0.76
KNN	0.92	KNN-ON	0.92	KNN-OC	0.75
KM	0.92	KM-ON	0.93	KM-OC	0.83
SVM	0.82	SVM-ON	0.85	SVM-OC	0.83
PCA	0.92	PCA-ON	0.92	PCA-OC	0.84
NN	0.92	NN-ON	0.92	NN-OC	0.85

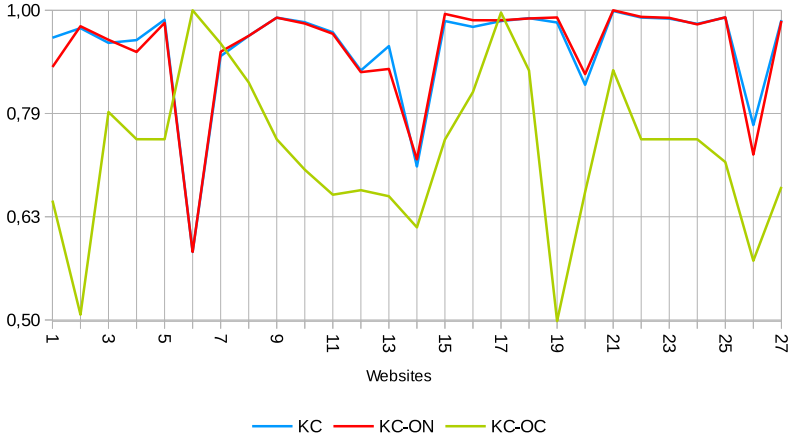
Tabla 8.14: *Valores de AUC logrados por OCC, OCC con características numéricas categóricas y OCC con características categóricas*



(a) *Gauss*

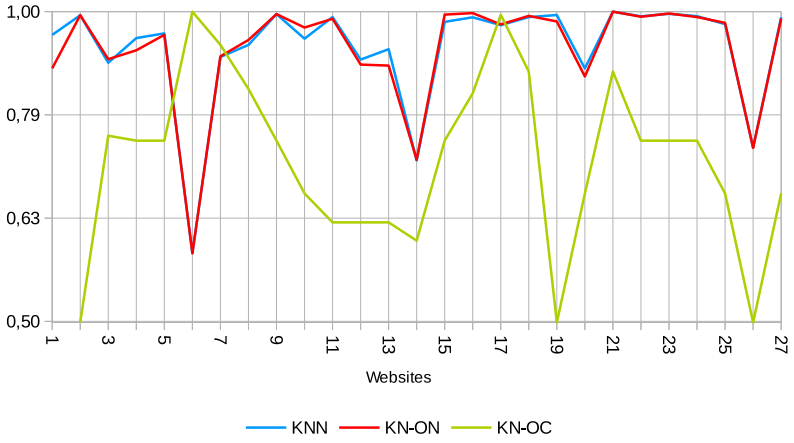


(b) *Parzen*

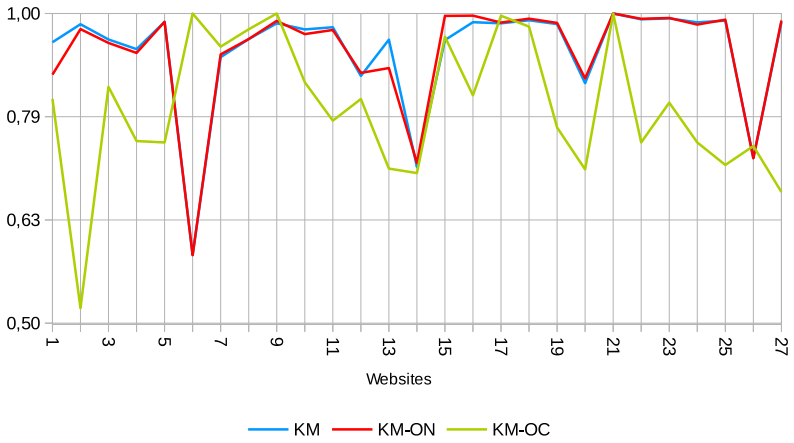


(c) *K-Center*

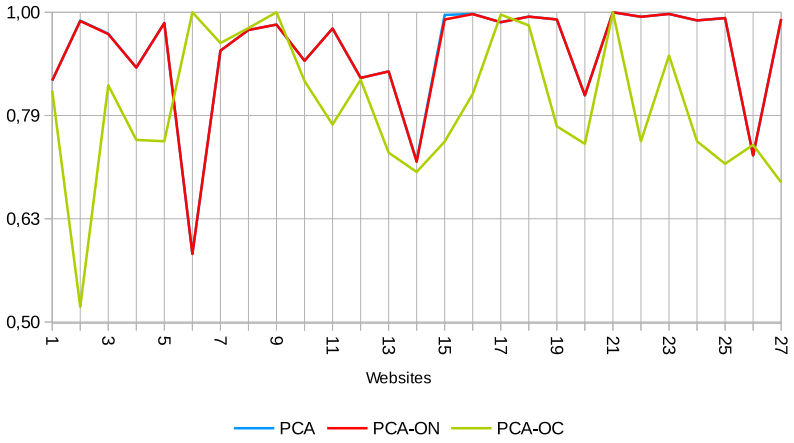
Figura 8.7: *Valores de AUC obtenidos por OCC con todas las características, sólo categóricas o sólo numéricas para cada sitio web (I)*



(a) *KNN*

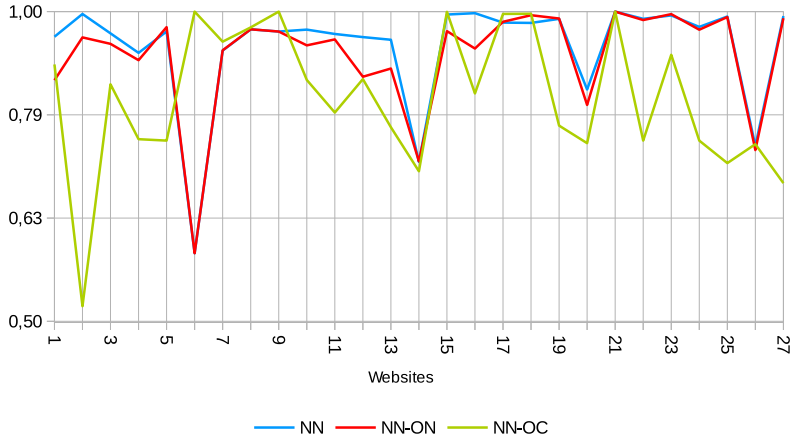


(b) *K-Medias*

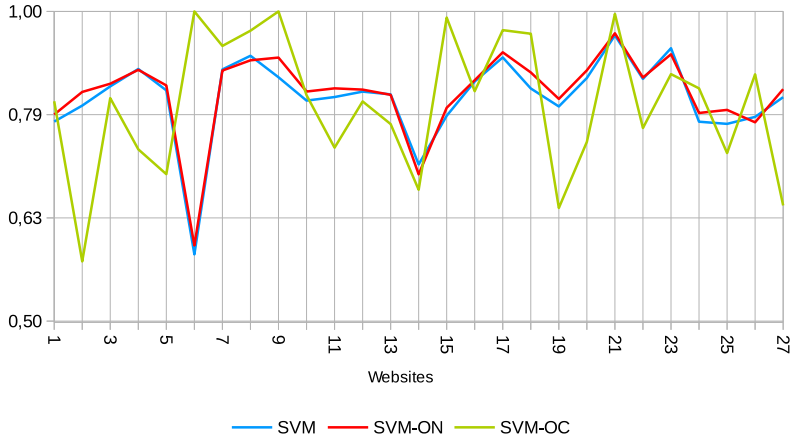


(c) *PCA*

Figura 8.8: *Valores de AUC obtenidos por OCC con todas las características, sólo categóricas o sólo numéricas para cada sitio web (II)*



(a) *Redes Neuronales*



(b) *SVM*

Figura 8.9: *Valores de AUC obtenidos por OCC con todas las características, sólo categóricas o sólo numéricas para cada sitio web (III)*

da una de las comparaciones por pares. Si comparamos los valores de R^+ y $R-$ de los test en los que comparamos *-ON con *-OC vemos que R^+ es mayor que R^- para todos los casos salvo para la comparación P-ON vs P-OC. En todos los casos, menos la comparación SVM-ON vs SVM-OC y P-ON vs P-OC, el valor de $p-value$ está por debajo de 0.05. Así pues podemos concluir que todas las variantes *-ON son mejores que *-OC salvo en el caso P-ON vs P-OC y SVM-ON vs SVM-OC donde se produce un empate.

Al comparar los ***-ON** con los modelos de verificación donde se mezclan

Alg	Estadístico	p-value	Alg	Estadístico	p-value	Alg	Estadístico	p-value
G	0.30	6.95e-25	**G-ON**	0.30	4.47e-24	**G-OC**	0.34	3.43e-32
P	0.23	4.63e-15	**P-ON**	0.22	2.90e-13	**P-OC**	0.34	1.25e-31
KC	0.29	6.54e-23	**KC-ON**	0.28	1.25e-20	**KC-OC**	0.31	3.48e-25
KNN	0.28	1.02e-21	**KNN-ON**	0.30	3.14e-24	**KNN-OC**	0.32	7.33e-28
KM	0.28	1.06e-21	**KM-ON**	0.29	6.20e-23	**KM-OC**	0.33	3.57e-30
SVM	0.08	0.07	**SVM-ON**	0.09	0.03	**SVM-OC**	0.27	4.57e-19
PCA	0.33	3.33e-31	**PCA-ON**	0.33	4.41e-29	**PCA-OC**	0.34	8.65e-31
NN	0.29	1.46e-23	**NN-ON**	0.36	1.00e-35	**NN-OC**	0.26	1.34e-18

Tabla 8.15: *Resultados del test de Kolmogorov-Smirnov para las variantes de los sistemas de verificación basados en OCC*

los dos tipos de características, vemos R^- es mayor que R^+ para las comparaciones G-ON vs G, KNN-ON vs KNN, PCA-ON vs PCA y NN-ON vs NN. En estos casos el valor de $p-value$ es menor que cero por lo que las variantes quehacen uso de todas las características mejores al otro conjunto de algoritmos. Para las camparativa P-ON vs P y SVM-ON vs SVM, R^+ es mayore que R^- y el valor de $p-value$ está próximo a cero por lo que son las variantes que sólo usan variables numéricas las que se comportan mejor. Para el resto de comparaciones, KC-ON vs KC y KM-ON vs KM, se llega a un empete entre ambas propuestas.

el valor de $p-value$ está por encima de 0.05 en todos los casos salvo en la comparación P vs P-ON. Esto indica que existe un empate en todas las comparaciones salvo en P vs P-ON para la que el valor de R^- es mucho mayor que R^+ y, por tanto, P-ON logra la victoria.

En consecuencia, la única conclusión que podemos obtener tras este experimento es que dependiendo de la instancia de la base de datos y del algoritmo aplicado el modelo de verificación que mejor resultado dará podrá ser, indistintamente, el que usa variables numéricas o bien que combine ambos tipos. Esto nos abre la posibilidad a buscar nuevos modelos de verificación que hagan una adecuada combinación de las características categóricas y no categóricas aprovechando el poder discriminador de ambas.

8.4.4. Experimento 4

Este experimento busca establecer cuál o cuales de los OCC es más adecuado para usar junto a MAVE. En particular este experimento se divide en tres fases:

Test	R^+	R^-	Z-value	p-value
G-ON vs. G-OC	1970	1033	-2.3788	0.01732
P-ON vs. P-OC	985	1571	-1.6788	0.09296
KC-ON vs KC-OC	2360.5	414.5	-5.2418	0
KNN-ON vs KNN-OC	2322	379	-5.3409	0
KM-ON vs KM-OC	1865	691	-3.3634	0.00078
PCA-ON vs PCA-OC	1857	699	-3.3176	0.0009
NN-ON vs NN-OC	1743.5	884.5	-2.4102	0.01596
SVM-ON vs SVM-OC	1999	1656	-0.7515	0.45326
G-ON vs. G	955	1746	-2.1743	0.03
P-ON vs. P	2114	164	-6.0905	0
KC-ON vs KC	1412	1144	-0.7678	0.4413
KNN-ON vs KNN	809	1606	-2.3826	0.01732
KM-ON vs KM	1091	1187	-0.2998	0.76418
PCA-ON vs PCA	843	1572	-2.1793	0.02926
NN-ON vs NN	884.5	1743.5	-2.4102	0.01596
SVM-ON vs SVM	2412.5	1242.5	-2.5633	0.01046

Tabla 8.16: *Resumen de los resultados del test de Wilcoxon aplicado OCC, OCC-ON y OCC-OC*

- Fase 1. El objetivo de esta primera fase es comprobar que MAVE es mejor que el algoritmo OCC en el cual se base. Esto es, comprobamos si la idea de tratar de forma independiente las características categóricas y no categóricas, y dar más peso a las primeras, mejora el proceso de verificación.

- Fase 2. Una vez que confirmemos que las distintas variantes de MAVE presentan un mejor comportamiento que las técnicas OCC, identificaremos qué variante de MAVE es la que mejores resultados alcanza.

- Fase 3. Finalmente, aplicaremos las técnicas tradicionales y compararemos sus resultados con los obtenidos por todas las variantes de MAVE estudiadas.

8.4.4.1. Resultados y estudio comparativo

La Tabla §8.17 muestra los resultados medios obtenidos por todos los algoritmos estudiados para cada uno de los sitios web que conforman nuestra base de datos. Las Figuras §8.10, §8.11, §8.12, §8.13 muestra el AUC medio alcanzado por cada uno de los algoritmos en cada una de las sitios web de la base de datos. Los valores han sido normalizados a una escala logarítica para apreciar mejor las diferencias entre algoritmos.

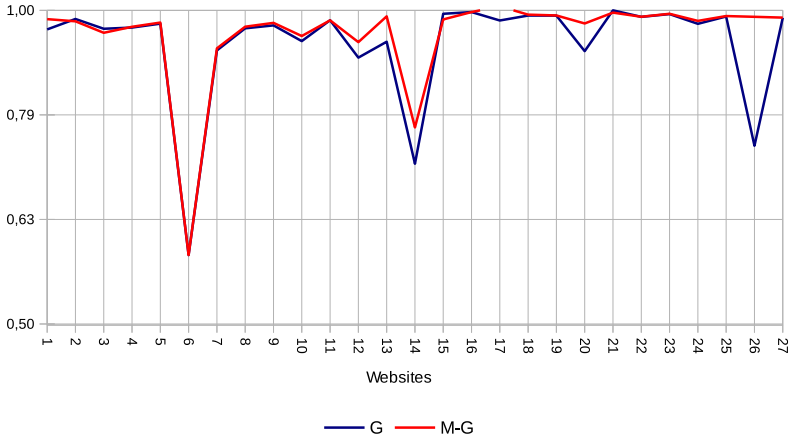
El análisis comparativo de los resultados logrados por todos los algoritmos se ha realizado usando los test estadísticos indicados en secciones anteriores. Previo a su aplicación, aplicamos el test de Kolmogorov-Smirnov a un nivel de significanción de 0.05 para asegurarnos que los valores a comparar no siguen una distribución normal, en la Tabla §8.18 podemos consultar los resultados tras la aplicación de este test. Como los p-values son menores que 0.05, la hipótesis nula se rechaza y la condición que garantiza la aplicación de test paramétricos no se cumple. Por tanto, el uso de test no paramétricos queda justificado.

Fase 1

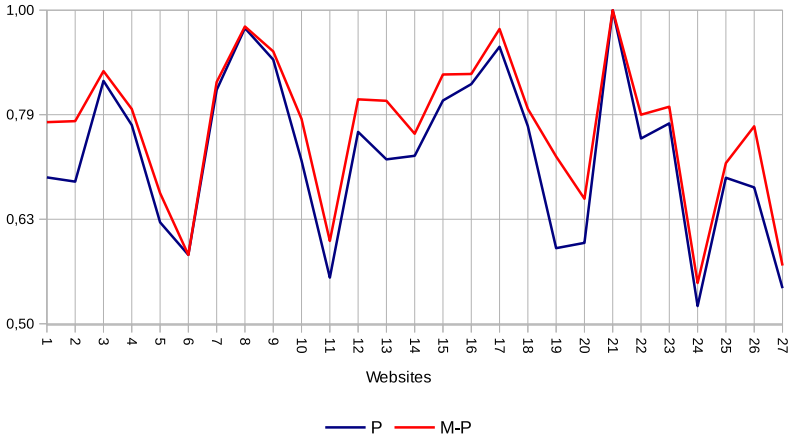
Si analizamos los datos de la Tabla §8.17 y comparamos los valores de las columnas segunda y cuarta, apreciamos que las variantes de MAVE alcanzan

Alg	AUC	Alg	AUC	Alg	AUC
G	0.93	**M-G**	0.95	**RAP**	0.84
P	0.73	**M-P**	0.78	**LER**	0.54
KC	0.92	**M-KC**	0.94	**MCC**	0.54
KNN	0.92	**M-KNN**	0.95		
KM	0.92	**M-KM**	0.95		
SVM	0.82	**M-SVM**	0.90		
PCA	0.92	**M-PCA**	0.92		
NN	0.92	**M-NN**	0.92		

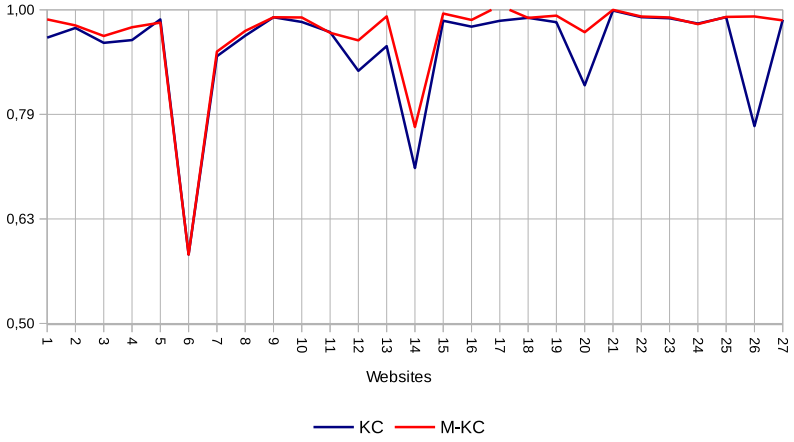
Tabla 8.17: *Valores de AUC logrados por variantes OCC, MAVE y los sistemas de verificación clásicos*



(a) *G vs M-G*



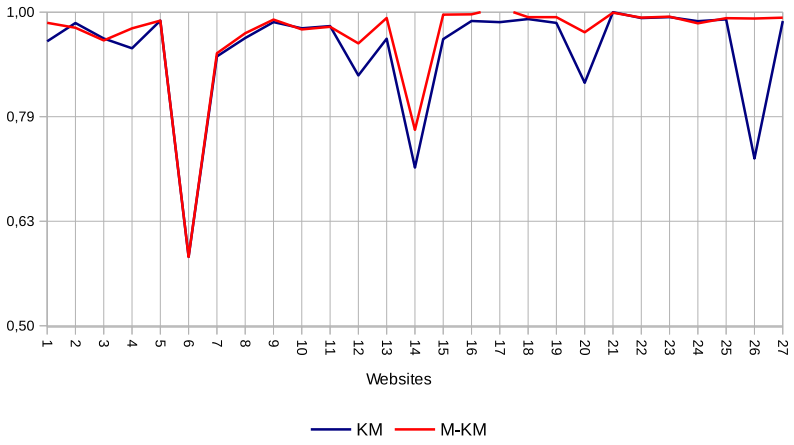
(b) *Mejor variante MAVE contra técnicas OCC*



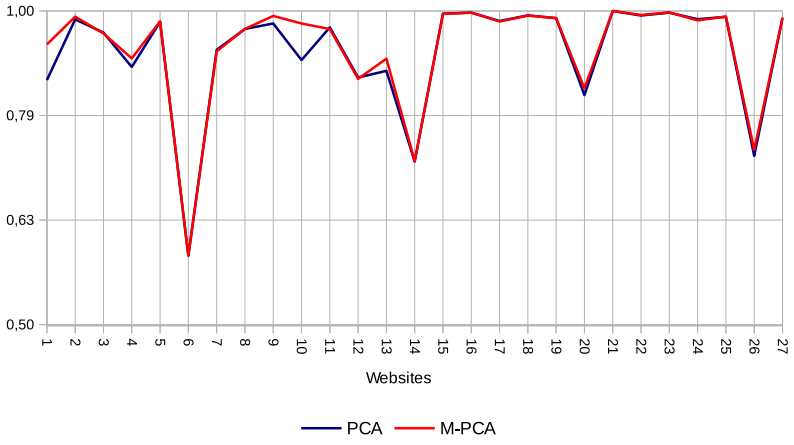
(c) *KC vs M-KC*

Figura 8.10: *Valores de AUC obtenidos por OCC y MAVE por sitio web (I)*

(a) *KNN vs M-KNN*

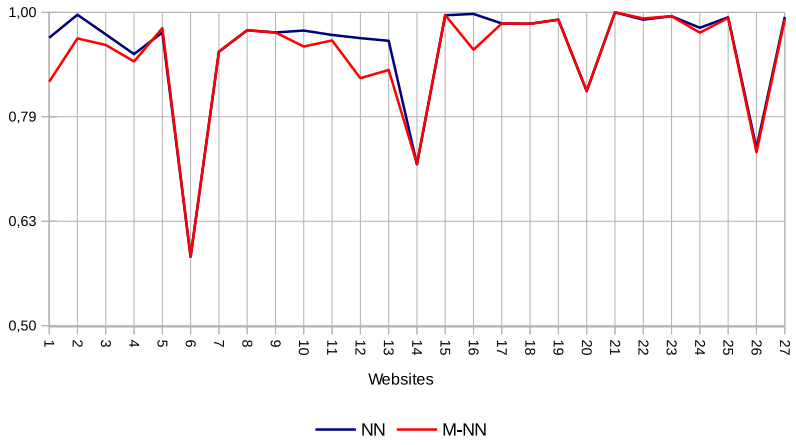


(b) *KM vs M-KM*

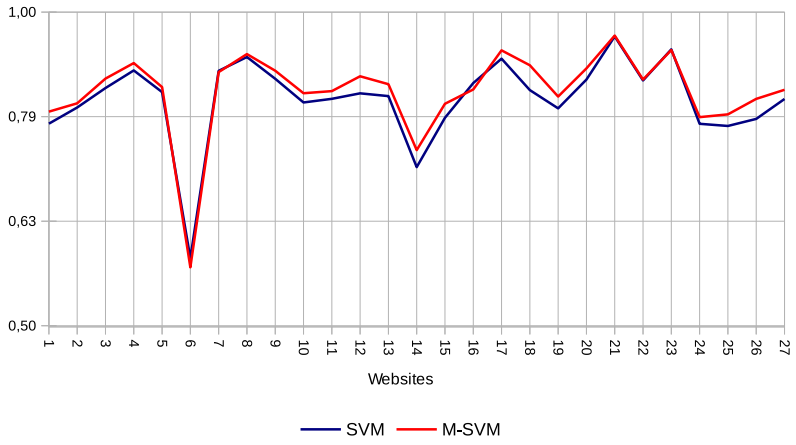


(c) *PCA vs M-PCA*

Figura 8.11: *Valores de AUC obtenidos por OCC y MAVE por sitio web (II)*



(a) *NN vs M-NN*

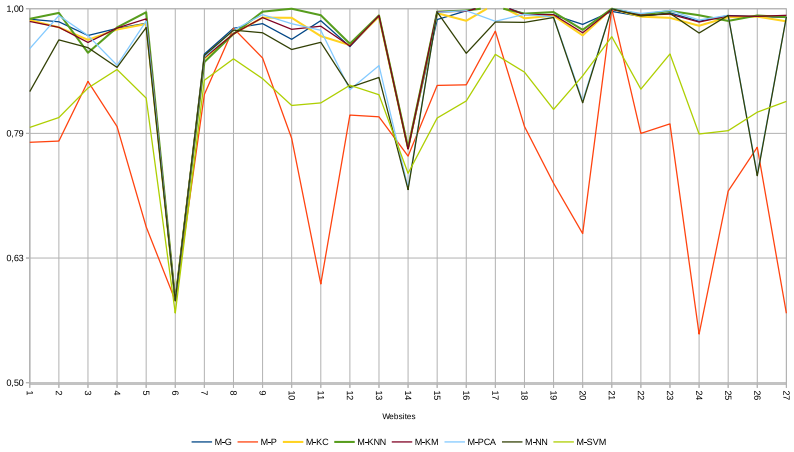


(b) *SVM vs M-SVM*

Figura 8.12: *Valores de AUC obtenidos por OCC y MAVE por sitio web (IIII)*

unos resultados mejores que los logrados por las técnicas OCC. Por ejemplo, si aplicamos el algoritmo P, el valor de AUC alcanza- do es 0.73 mientras que el algoritmo M-P logra superar esta cifra logrando un 0.78 de AUC. Este comportamiento se mantiene en mayor o menor medida para todas las variantes de MAVE y las técnicas de OCC.

Las Figuras §8.10, §8.11, §8.12 muestra la comparativa, para cada sitio WEB, de cada uno de los algoritmos OCC con respecto a la variante de MAVE que lo emplea. Puede observarse que las variantes de MAVE superan lo logrado por su homólogo OCC para cada sitio web. Por tanto, el primer ni-



(a) *Variantes MAVE*

Figura 8.13: *Valores de AUC obtenidos por las variantes de MAVE, OCC y los sistemas de verificación clásico para cada sitio web*

vel introducido por MAVE, previo a la aplicación de las técnicas OCC, incrementa considerablemente el poder de verificación de MAVE.

Todo esto queda corroborado tras la aplicación del test Wilcoxon Signed-Ranks Test. La Tabla §8.19 muestra los resultados tras aplicar este test a los valores de AUC logrados por los variantes MAVE y las técnicas de OCC. Dado que, el $p - value \leq 0.05$, podemos concluir que los resultados son significativos y, por tanto, los variantes de MAVE son mejores que su homólogo OCC salvo el caso de PCA donde existe un empate al obtener un p-value superior al nivel de significación.

Fase 2

Si volvemos a centrarnos en los valores de la Tabla §8.17 y comparamos los valores de la cuarta columna, apreciamos que los valores de AUC logrados por las distintas variantes de MAVE son bastante altos (superiores o iguales al 0.90), salvo para el algoritmo M-P, que obtienen un valor de 0.78. Concretamente, los algoritmos M-G, M-KNN y M-KM llegan a valores de AUC de 0.95.

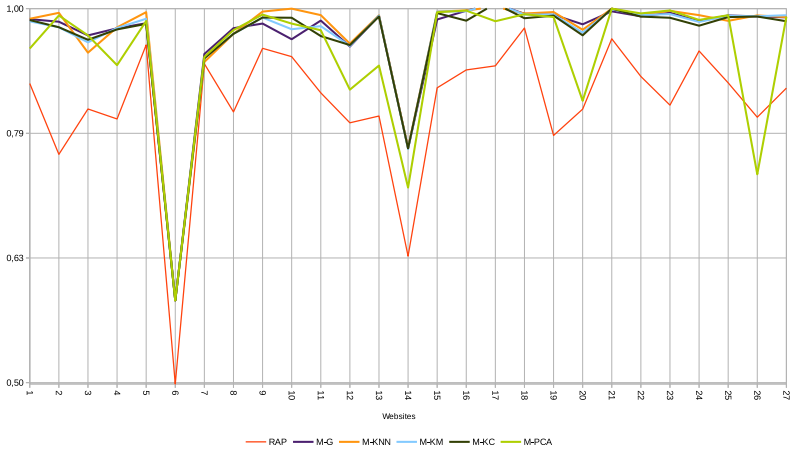
En la Figura §8.13b podemos ver la evolución de las distintas variantes de MAVE para cada uno de los sitios web que forman nuestra base de datos. Vemos que el comportamiento global antes descrito también se puede apreciar

Alg	Estadístico	p-value	Alg	Estadístico	p-value
G	0.30	6.95e-25	M-G	0.31	4.57e-27
P	0.23	4.63e-15	M-P	0.22	2.91e-13
KC	0.29	6.54e-23	M-KC	0.29	2.02e-24
KNN	0.28	1.02e-21	M-KNN	0.33	3.86e-30
KM	0.28	1.06e-21	M-KM	0.30	7.01-26
SVM	0.08	0.07	M-SVM	0.26	1.60e-18
PCA	0.33	3.33e-31	M-PCA	0.33	6.21e-31
NN	0.29	1.46e-23	M-NN	0.28	1.01e-22
RAP	0.13	0.00			
LER	0.50	2.87e-72			
MCC	0.48	6.28e-68			

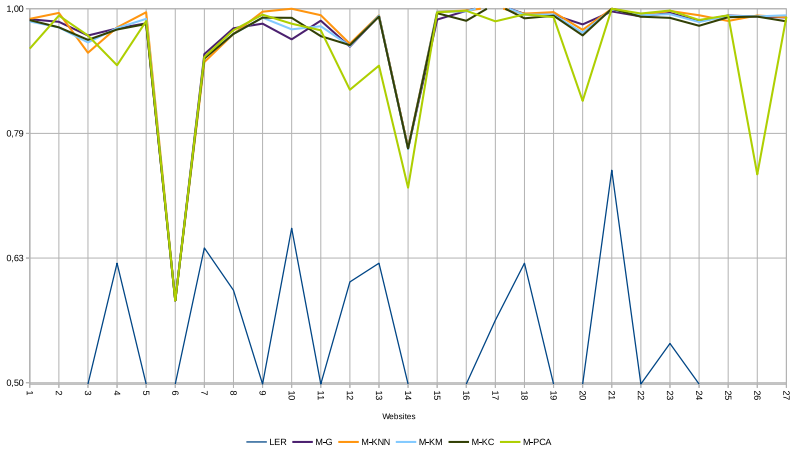
Tabla 8.18: *Resultados del test de Kolmogorov-Smirnov para variantes MAVE, técnicas OCC y clásicas*

en el caso particular de cada sitio. Esto es, todas las líneas se entrelazan salvo las correspondientes a M-P y M-SVM. Independiente del sitio web que analicemos, los valores de AUC se muestran muy prometedores salvo para los sitios 6 y 14, en donde apenas superan el 0.5. La razón de este mal comportamiento la encontramos en que el conjunto de características usadas no es capaz de discriminar adecuadamente cada uno de los slots y a un mal etiquetado de los elementos que formaron nuestro conjunto de entrenamiento.

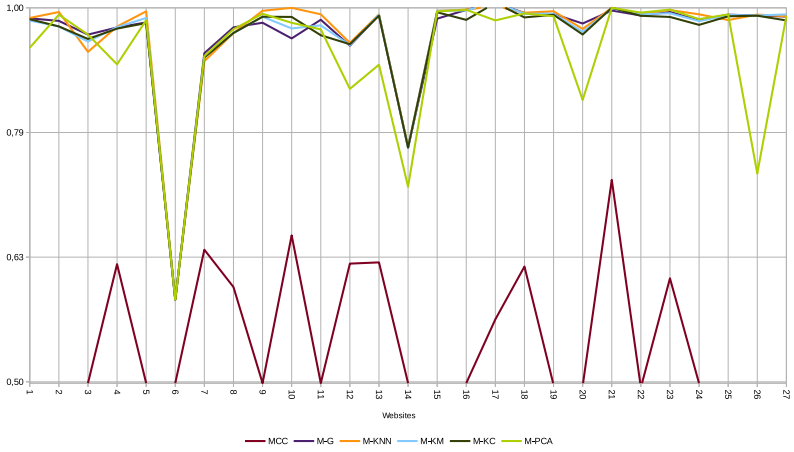
La Tabla §8.20 muestra los resultados devueltos tras la aplicación del test de Friedman. Este test se uso para detectar diferencias significativas de rendimiento entre las distintas variantes de MAVE. La segunda columna muestra el valor del ranking y la tercera el valor devuelto por el post-test de Holm. Si interpretamos los resultados de este test, vemos que M-G es el que mejor ranking obtiene, pero este no es significativo respecto a los cuatro siguientes. Por tanto, existen distintos algoritmo OCC que puede usar MAVE en su segundo nivel sin que el rendimiento se vea afectado.



(a) *RAP*



(b) *LER*



(c) *MCC*

Figura 8.14: *Valores de AUC obtenidos por las técnicas tradicionales y las variantes de MAVE*

Test	R^+	R^-	Z-value	p-value
M-G vs. G	1065	261	3.7681	0.00016
M-P vs. P	1812	79	-6.2239	0
M-KC vs KC	1661	230	-5.1393	0
M-KNN vs KNN	1551	102	-5.7563	0
M-KM vs KM	1359	237	-4.5761	0
M-PCA vs PCA	679.5	355.5	-1.8286	0.06724
M-NN vs NN	1466	756	-2.3029	0.02144
M-SVM vs SVM	256	3399	-6.8859	0

Tabla 8.19: *Resumen de los resultados del test de Wilcoxon aplicado a variantes MAVE y técnicas OCC*

Algoritmo	Ranking	pHolm
M-G	584.93	-
M-KNN	592.21	0.92
M-KM	596.55	0.87
M-KC	615.15	0.69
M-PCA	724.54	0.06
M-SVM	793.30	0.00
M-NN	825.20	0.00
M-P	1260.58	0

Tabla 8.20: *Resumen de los resultados del test de Friedman aplicados a las variantes de MAVE*

Fase 3

Observando ahora los valores de la cuarta y sexta columna de la Tabla §8.17, vemos que el valor medio de AUC alcanzado por las todas las variantes de MAVE mejora a los logrados con los sistemas de verificación clásicos. El mejor valor de AUC alcanzado por estas últimas no llega al 0.85 lo

que contrasta con el valor de 0.95 logrado por las mejores variantes de MAVE.

Si pasamos a analizar sitio por sitio Figura §8.14, los valores de MCC y LER son muy malos para todos los sitios web. En el caso del algoritmo RAP, su comportamiento está por debajo de las mejores variantes de MAVE en todos los sitios web salvo el 26. Al compararlo con el resto de variantes MAVE los valores de AUC se emparejan pero siguen siendo un poco peor.

Por último, si comparamos los resultados de todas las variantes MAVE respecto a los sistemas de verificación clásicos, de nuevo usando el test de Wilcoxon Signed-Ranks Test (Tabla §8.21), vemos que salvo en el caso de M-P vs RAP los resultados de las primeras son siempre mejores que la de los sistemas de verificación clásicos.

8.5. Resumen

En este capítulo, hemos descrito cómo se han realizado los experimentos para poder comparar los distintos algoritmos estudiamos en este trabajo de investigación. Hemos definido la base de datos, las características y las medidas a usadas además de detallar cada uno de los pasos llevados a cabo durante el experimento y exponer los resultados alcanzados.

	R^+	R^-	Z-value	p-value
M-G	3142	18	-7.6336	0
M-P	1148.5	2011.5	-2.1088	0.03486
M-KC	3066	15	-7.5981	0
M-KNN	3137	23	-7.6092	0
M-KM	3152	8	-7.6825	0
M-PCA	2979	181	-6.837	0
M-NN	2791	369	-5.9182	0
M-SVM	3142.5	512.5	-5.762	0

(a) *Variantes MAVE vs RAP*

	R^+	R^-	Z-value	p-value
M-G	3655	0	-8.0077	0
M-P	3651	4	-7.9901	0
M-KC	3655	0	-8.0077	0
M-KNN	3655	0	-8.0077	0
M-KM	3655	0	-8.0077	0
M-PCA	3654	1	-8.0033	0
M-NN	3653	2	-7.9989	0
M-SVM	3734	7	-8.0242	0

(b) *Variantes MAVE vs MCC*

	R^+	R^-	Z-value	p-value
M-G	3655	0	-8.0077	0
M-P	3480	6	-7.8862	0
M-KC	3655	0	-8.0077	0
M-KNN	3655	0	-8.0077	0
M-KM	3654	1	-8.0033	0
M-PCA	3654	1	-8.0033	0
M-NN	3654	1	-8.0033	0
M-SVM	3734	7	-8.0242	0

(c) *Variantes MAVE vs LER*

Tabla 8.21: *Resultados del test de Wilcoxon aplicado a las variantes de MAVE y los sistemas de verificación clásicos*

Parte IV

Comentarios finales

Capítulo 9

Conclusiones y trabajos futuros

This is the end Beautiful friend This is the end My only friend, the end.

The End, Canción (1966)

En esta disertación, hemos presentados dos nuevos enfoques a la hora de tratar el problema de la verificación de información extraídas por wrappers web. La primera propuesta plantea enfocar el problema de la verificación desde el punto de vista de la optimización computacional mientras que la segunda lo enfoca desde el campo de la clasificación.

Nuestra primera propuesta permite usar cualquier algoritmo aplicado en la optimización computacional. En concreto se optó por aplicar las metaheurísticas ACO para solucionar el problema. De todas estas metaherísticas, se aplicó BWAS que ya había demostrado resultados muy prometedores en comparación con otros metaheurísticas ACO. Este sistema de verificación, establece una analogía entre el problema del QAP y el de la verificación de información. De esta forma los slots son considerados entidades que se asocian a localizaciones que, en el problema de la verificación, son los roles al que puede pertenecer un slot. Lo que se busca es minimizar el coste de asociación teniendo en cuenta que si una entidad es asociada adecuadamente a una localización el coste es cero.

La segunda propuesta hace uso de un tipo especial de clasificador denominados clasificadores de una única clase. El sistema de verificación propuesto se denomina MAVE y parte de la premisa de ponderar la capacidad discriminatoria de las características aplicadas a la hora de crear el modelo de

verificación. En concreto, MAVE define dos niveles de verificación: en el primer nivel se usan sólo características categóricas mientras que el segundo nivel sólo se aplican las numéricas. El segundo nivel sólo se usa para casos en los que el primer nivel no haya sido capaz de detectar si el dato no es correcto.

Si bien los estudios comparativos que se han realizado entre los sistemas propuestos y los ya existentes, reflejan que las nuevas técnicas superan en valor de AUC a las clásicas, todavía quedan distintas líneas de investigación a implantar. Así pues, en los sistemas de verificación basados en BWAS sería deseable que el número de puntos que forman nuestro modelo de verificación se pudiera calcular y modificar dinámicamente, durante la ejecución del algoritmo. También habría que hacer un estudio con distintas funciones de fitness y/o búsqueda local para lograr alcanzar resultado más óptimos. En el algoritmo MAVE queda abierta la puerta a hacer un ajuste más específico para cada sitio web en base a su dominio.

Parte V

Apéndices

Apéndice A

Tablas de resultados

En esta disertación se han estudiado 36 sistemas de verificación. Estos se han clasificado en los siguiente grupos: técnicas tradicionales, basados en colonias de hormigas, basados en OCC y jerárquicos. En este apéndice incluimos las tablas completas de resultados obtenidas por cada una de estas técnicas para cada uno de los experimentos tratados. Este apéndice se ha organizado de la siguiente manera: la Sección describe la estructura de los datos y cómo se han obtenido;la Sección §A.2 presenta los resultados obtenidos por las técnicas tradicionales; la Sección §A.3 describe los resultados alcanzados por BWAS; la Sección §A.4 contiene las tablas de recultados logrados por las variantes OCC; en la Sección §A.5 se encuentran los resultados obtenidos por MAVE y todas sus variantes usadas.

A.1. Introducción

En este apéndice mostramos los resultados de AUC obtenidos por todos los sistemas de verificación. Las tablas que se exponen en los apartados siguientes son el resultados de aplicar el procedimiento de de validación cruzada descrito en la Sección §8.2 para cada uno de los experimentos diseñados en la Sección §8.4.

En cada sección se incluyen dos tablas. La primera representa los datos medio de AUC por sitio y etiqueta y tiene la siguiente estructura: la primera columna (**Sitio**) indica el sitio web tratado, la segunda (**Eti**) el datos de interés que estamos verificando, el resto de columnas se corresponden con cada uno de los algoritmos analizados. Cada celda contiene el dato de AUC logrado por cada algoritmo para cada sitio y dato de interés. La segunda tabla calcula los datos medios de AUC por sitio.

A.2. Técnicas tradicionales

Sitio	Eti	RAP	LER	MCC	Sitio	Eti	RAP	LER	MCC
1	1	0.88	0.50	0.50	9	1	1.00	0.50	0.50
1	2	0.87	0.50	0.50	9	3	0.86	0.50	0.50
1	4	0.86	0.50	0.50	10	1	0.86	0.50	0.50
2	1	0.74	0.50	0.50	10	2	0.88	0.50	0.50
2	3	0.73	0.50	0.50	10	4	1.00	1.00	1.00
2	4	0.82	0.50	0.50	11	1	0.88	0.50	0.50
3	1	0.83	0.50	0.50	11	2	0.83	0.50	0.50
3	2	0.84	0.56	0.52	11	3	0.79	0.50	0.50
3	3	0.89	0.50	0.50	11	5	0.92	0.50	0.50
3	5	1.00	0.50	0.50	12	1	0.87	0.50	0.50
3	6	0.65	0.50	0.50	12	2	0.72	0.50	0.50
3	7	0.88	0.50	0.50	12	3	0.65	0.50	0.50
3	8	0.66	0.50	0.50	12	5	1.00	1.00	1.00
3	9	0.91	0.50	0.56	13	2	1.00	1.00	1.00
4	1	0.88	0.50	0.50	13	3	0.69	0.50	0.50
4	2	0.78	0.50	0.50	13	4	0.75	0.50	0.50
4	4	1.00	1.00	1.00	13	5	0.84	0.50	0.50
4	5	0.61	0.50	0.50	14	1	0.46	0.50	0.50
5	1	0.94	0.50	0.50	14	2	0.89	0.50	0.50
5	3	0.93	0.50	0.50	14	3	0.66	0.50	0.50
6	1	0.50	0.50	0.50	14	4	0.59	0.50	0.50
6	2	0.50	0.50	0.50	14	5	0.56	0.50	0.50
7	1	1.00	1.00	1.00	15	2	0.89	0.50	0.50
7	2	1.00	0.50	0.50	15	3	0.83	0.50	0.50
7	3	1.00	0.50	0.50	16	1	0.93	0.50	0.50
7	4	0.66	0.50	0.50	16	2	0.87	0.50	0.50
7	5	1.00	1.00	1.00	16	4	0.88	0.50	0.50
7	6	0.66	0.50	0.50	17	1	0.86	0.50	0.50
7	7	1.00	0.50	0.50	17	2	1.00	1.00	1.00

Sitio	Eti	RAP	LER	MCC	Sitio	Eti	RAP	LER	MCC
17	3	0.93	0.50	0.50	21	5	0.86	0.50	0.50
17	4	0.93	0.50	0.50	22	1	0.96	0.50	0.50
17	5	0.88	0.50	0.50	22	3	0.80	0.50	0.50
17	6	0.82	0.50	0.50	23	1	0.88	0.50	0.50
17	8	0.82	0.50	0.50	23	2	1.00	1.00	1.00
17	9	0.94	0.50	0.50	23	4	0.62	0.50	0.50
18	1	1.00	1.00	1.00	23	5	0.84	0.50	0.50
18	2	0.95	0.50	0.50	24	2	0.93	0.50	0.50
18	3	1.00	0.50	0.50	24	3	0.92	0.50	0.50
18	4	0.91	0.50	0.50	25	2	0.74	0.50	0.50
19	2	0.69	0.50	0.50	25	3	0.91	0.50	0.50
19	3	0.89	0.50	0.50	25	4	0.96	0.50	0.50
20	1	0.78	0.50	0.50	26	2	0.85	0.50	0.50
20	2	0.80	0.50	0.50	26	3	0.79	0.50	0.50
20	3	0.91	0.50	0.50	27	1	0.74	0.50	0.50
21	1	1.00	1.00	1.00	27	2	0.95	0.50	0.50
21	3	1.00	1.00	1.00	27	4	0.91	0.50	0.50
4	5	0.61	0.50	0.50	14	1	0.46	0.50	0.50
5	1	0.94	0.50	0.50	14	2	0.89	0.50	0.50
5	3	0.93	0.50	0.50	14	3	0.66	0.50	0.50
6	1	0.50	0.50	0.50	14	4	0.59	0.50	0.50
6	2	0.50	0.50	0.50	14	5	0.56	0.50	0.50
7	1	1.00	1.00	1.00	15	2	0.89	0.50	0.50
7	2	1.00	0.50	0.50	15	3	0.83	0.50	0.50
7	3	1.00	0.50	0.50	16	1	0.93	0.50	0.50
7	4	0.66	0.50	0.50	16	2	0.87	0.50	0.50
7	5	1.00	1.00	1.00	16	4	0.88	0.50	0.50
7	6	0.66	0.50	0.50	17	1	0.86	0.50	0.50
7	7	1.00	0.50	0.50	17	2	1.00	1.00	1.00

Sitio	RAP	LER	MCC
1	0,87	0,50	0,50
2	0,76	0,50	0,50
3	0,83	0,50	0,50
4	0,82	0,63	0,62
5	0,94	0,50	0,50
6	0,50	0,50	0,50
7	0,90	0,64	0,64
8	0,83	0,59	0,60
9	0,93	0,50	0,50
10	0,92	0,67	0,66
11	0,86	0,50	0,50
12	0,81	0,60	0,62
13	0,82	0,63	0,62
14	0,63	0,50	0,50
15	0,86	0,50	0,50
16	0,89	0,50	0,50
17	0,90	0,56	0,56
18	0,96	0,63	0,62
19	0,79	0,50	0,50
20	0,83	0,50	0,50
21	0,95	0,74	0,73
22	0,88	0,50	0,50
23	0,84	0,54	0,61
24	0,92	0,50	0,50
25	0,87	0,50	0,50
26	0,82	0,50	0,50
27	0,86	0,50	0,50

A.3. BWAS

Sitio	Eti	BWAS	Sitio	Eti	BWAS
1	1	0.86	9	1	1.00
1	2	0.74	9	3	0.91
1	4	0.98	10	1	0.90
2	1	0.98	10	2	0.88
2	3	1.00	10	4	1.00
2	4	0.85	11	1	0.85
3	1	0.98	11	2	0.99
3	2	0.99	11	3	0.96
3	3	0.95	11	5	0.96
3	5	1.00	12	1	0.99
3	6	0.67	12	2	0.71
3	7	0.86	12	3	0.76
3	8	1.00	12	5	1.00
3	9	1.00	13	2	1.00
4	1	1.00	13	3	0.83
4	2	0.77	13	4	0.72
4	4	1.00	13	5	0.98
4	5	0.82	14	1	0.56
5	1	0.95	14	2	1.00
5	3	0.98	14	3	1.00
6	1	0.50	14	4	0.52
6	2	0.67	14	5	0.51
7	1	1.00	15	2	0.99
7	2	1.00	15	3	1.00
7	3	1.00	16	1	1.00
7	4	0.60	16	2	0.76
7	5	1.00	16	4	1.00
7	6	0.82	17	1	0.98
7	7	1.00	17	2	1.00

Sitio	Eti	BWAS	Sitio	Eti	BWAS
17	3	0.86	21	5	1.00
17	4	0.74	22	1	0.98
17	5	0.98	22	3	0.99
17	6	0.98	23	1	0.98
17	8	1.00	23	2	1.00
17	9	0.85	23	4	1.00
18	1	0.98	23	5	0.99
18	2	0.99	24	2	0.99
18	3	0.95	24	3	0.92
18	4	1.00	25	2	0.98
19	2	0.67	25	3	1.00
19	3	0.86	25	4	0.99
20	1	1.00	26	2	0.97
20	2	1.00	26	3	0.49
20	3	1.00	27	1	0.98
21	1	0.77	27	2	0.99
21	3	1.00	27	4	0.98
21	4	0.82			

Sitio	Eti	BWAS	RAP	LER	MCC	Sitio	Eti	BWAS	RAP	LER	MCC
1	**1**	0.86	0.88	0.50	0.50	**9**	**1**	1.00	1.00	0.50	0.50
1	**2**	0.74	0.87	0.50	0.50	**9**	**3**	0.91	0.86	0.50	0.50
1	**4**	0.98	0.86	0.50	0.50	**10**	**1**	0.90	0.86	0.50	0.50
2	**1**	0.98	0.74	0.50	0.50	**10**	**2**	0.88	0.88	0.50	0.50
2	**3**	1.00	0.73	0.50	0.50	**10**	**4**	1.00	1.00	1.00	1.00
2	**4**	0.85	0.82	0.50	0.50	**11**	**1**	0.85	0.88	0.50	0.50
3	**1**	0.98	0.83	0.50	0.50	**11**	**2**	0.99	0.83	0.50	0.50
3	**2**	0.99	0.84	0.56	0.52	**11**	**3**	0.96	0.79	0.50	0.50
3	**3**	0.95	0.89	0.50	0.50	**11**	**5**	0.96	0.92	0.50	0.50
3	**5**	1.00	1.00	0.50	0.50	**12**	**1**	0.99	0.87	0.50	0.50
3	**6**	0.67	0.65	0.50	0.50	**12**	**2**	0.71	0.72	0.50	0.50
3	**7**	0.86	0.88	0.50	0.50	**12**	**3**	0.76	0.65	0.50	0.50
3	**8**	1.00	0.66	0.50	0.50	**12**	**5**	1.00	1.00	1.00	1.00
3	**9**	1.00	0.91	0.50	0.56	**13**	**2**	1.00	1.00	1.00	1.00
4	**1**	1.00	0.88	0.50	0.50	**13**	**3**	0.83	0.69	0.50	0.50
4	**2**	0.77	0.78	0.50	0.50	**13**	**4**	0.72	0.75	0.50	0.50
4	**4**	1.00	1.00	1.00	1.00	**13**	**5**	0.98	0.84	0.50	0.50
4	**5**	0.82	0.61	0.50	0.50	**14**	**1**	0.56	0.46	0.50	0.50
5	**1**	0.95	0.94	0.50	0.50	**14**	**2**	1.00	0.89	0.50	0.50
5	**3**	0.98	0.93	0.50	0.50	**14**	**3**	1.00	0.66	0.50	0.50
6	**1**	0.50	0.50	0.50	0.50	**14**	**4**	0.52	0.59	0.50	0.50
6	**2**	0.67	0.50	0.50	0.50	**14**	**5**	0.51	0.56	0.50	0.50
7	**1**	1.00	1.00	1.00	1.00	**15**	**2**	0.99	0.89	0.50	0.50
7	**2**	1.00	1.00	0.50	0.50	**15**	**3**	1.00	0.83	0.50	0.50
7	**3**	1.00	1.00	0.50	0.50	**16**	**1**	1.00	0.93	0.50	0.50
7	**4**	0.60	0.66	0.50	0.50	**16**	**2**	0.76	0.87	0.50	0.50
7	**5**	1.00	1.00	1.00	1.00	**16**	**4**	1.00	0.88	0.50	0.50
7	**6**	0.82	0.66	0.50	0.50	**17**	**1**	0.98	0.86	0.50	0.50
7	**7**	1.00	1.00	0.50	0.50	**17**	**2**	1.00	1.00	1.00	1.00

Continúa en la página siguiente

Sitio	Eti	BWAS	RAP	LER	MCC	Sitio	Eti	BWAS	RAP	LER	MCC
17	**3**	0.88	0.93	0.50	0.50	**21**	**5**	1.00	0.86	0.50	0.50
17	**4**	0.99	0.93	0.50	0.50	**22**	**1**	0.98	0.96	0.50	0.50
17	**5**	0.98	0.88	0.50	0.50	**22**	**3**	0.99	0.80	0.50	0.50
17	**6**	0.99	0.82	0.50	0.50	**23**	**1**	0.98	0.88	0.50	0.50
17	**8**	0.99	0.82	0.50	0.50	**23**	**2**	1.00	1.00	1.00	1.00
17	**9**	1.00	0.94	0.50	0.50	**23**	**4**	1.00	0.62	0.50	0.50
18	**1**	1.00	1.00	1.00	1.00	**23**	**5**	0.99	0.84	0.50	0.50
18	**2**	0.99	0.95	0.50	0.50	**24**	**2**	0.99	0.93	0.50	0.50
18	**3**	1.00	1.00	0.50	0.50	**24**	**3**	0.92	0.92	0.50	0.50
18	**4**	0.91	0.91	0.50	0.50	**25**	**2**	0.98	0.74	0.50	0.50
19	**2**	0.98	0.69	0.50	0.50	**25**	**3**	1.00	0.91	0.50	0.50
19	**3**	0.98	0.89	0.50	0.50	**25**	**4**	0.99	0.96	0.50	0.50
20	**1**	0.81	0.78	0.50	0.50	**26**	**2**	0.97	0.85	0.50	0.50
20	**2**	0.74	0.80	0.50	0.50	**26**	**3**	0.49	0.79	0.50	0.50
20	**3**	0.97	0.91	0.50	0.50	**27**	**1**	0.98	0.74	0.50	0.50
21	**1**	1.00	1.00	1.00	1.00	**27**	**2**	0.99	0.95	0.50	0.50
21	**3**	1.00	1.00	1.00	1.00	**27**	**4**	0.98	0.91	0.50	0.50
21	**4**	1.00	0.92	0.50	0.50						

Sitio	BWAS
1	0,86
2	0,94
3	0,93
4	0,90
5	0,97
6	0,58
7	0,92
8	0,96
9	0,96
10	0,93
11	0,94
12	0,86
13	0,88
14	0,77
15	0,83
16	0,92
17	0,98
18	0,98
19	0,98
20	0,84
21	1,00
22	0,99
23	0,99
24	0,96
25	0,99
26	0,73
27	0,99

A.4. Verificadores OCC

Site	Cla	G	P	KC	KN	KM	PCA	NN	SVM
1	**1**	0.92	0.62	0.92	0.95	0.93	0.89	0,89	0.77
1	**2**	0.96	0.72	0.91	0.91	0.91	0.91	0,96	0.75
1	**4**	0.99	0.73	0.98	0.99	0.97	0.98	0,99	0.82
2	**1**	0.99	0.56	0.97	0.99	0.99	0.98	0,98	0.77
2	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.88
2	**4**	0.95	0.50	0.91	0.98	0.94	0.98	1,00	0.79
3	**1**	0.97	0.98	0.96	0.87	0.95	0.94	0,97	0.85
3	**2**	0.97	0.99	0.96	0.93	0.98	0.97	0,99	0.89
3	**3**	0.92	0.77	0.94	0.96	0.94	0.94	0,93	0.74
3	**5**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.98
3	**6**	0.87	0.52	0.72	0.64	0.77	0.79	0,80	0.72
3	**7**	0.95	0.59	0.95	0.92	0.92	0.96	0,93	0.78
3	**8**	1.00	1.00	0.94	0.88	1.00	1.00	1,00	0.91
3	**9**	1.00	1.00	0.97	0.94	0.99	1.00	1,00	0.91
4	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
4	**2**	0.93	0.50	0.86	0.85	0.82	0.77	0,77	0.72
4	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
4	**5**	0.92	0.61	0.88	0.92	0.87	0.83	0,87	0.80
5	**1**	0.96	0.70	0.98	0.91	0.98	0.97	0,93	0.91
5	**3**	0.98	0.56	0.98	0.99	0.98	0.99	0,99	0.77
6	**1**	0.50	0.50	0.50	0.50	0.50	0.50	0,50	0.50
6	**2**	0.67	0.67	0.67	0.67	0.67	0.67	0,67	0.67
7	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
7	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.95
7	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.89
7	**4**	0.58	0.39	0.60	0.62	0.61	0.58	0,61	0.70
7	**5**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
7	**6**	0.83	0.48	0.72	0.70	0.74	0.83	0,81	0.68
7	**7**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.93
8	**1**	0.97	0.97	0.90	0.87	0.90	0.97	0,97	0.92
8	**2**	0.92	0.92	0.92	0.92	0.92	0.92	0,92	0.83
8	**3**	0.96	0.96	0.96	0.93	0.96	0.96	0,96	0.87
8	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00

Continúa en la página siguiente

Site	Cla	G	P	KC	KN	KM	PCA	NN	SVM
9	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.93
9	**3**	0.93	0.79	0.97	0.99	0.96	0.98	0,91	0.79
10	**1**	0.95	0.60	0.97	0.83	0.94	0.95	0,94	0.74
10	**2**	0.85	0.56	0.96	0.99	0.96	0.97	0,94	0.72
10	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
11	**1**	0.96	0.50	0.90	0.98	0.94	0.92	0,90	0.87
11	**2**	0.99	0.50	0.98	0.99	0.99	0.97	0,99	0.76
11	**3**	0.98	0.60	0.95	0.99	0.96	0.97	0,94	0.87
11	**5**	0.99	0.62	0.97	0.99	0.99	0.98	0,98	0.81
12	**1**	0.99	0.84	0.96	0.97	0.95	0.98	0,99	0.89
12	**2**	0.86	0.64	0.78	0.88	0.77	0.71	0,80	0.81
12	**3**	0.76	0.58	0.76	0.74	0.76	0.76	1,00	0.65
12	**5**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
13	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
13	**3**	0.90	0.57	0.82	0.85	0.89	0.86	0,89	0.76
13	**4**	0.85	0.66	0.89	0.84	0.90	0.75	0,88	0.77
13	**5**	0.99	0.64	0.98	0.99	0.98	0.99	0,99	0.79
14	**1**	0.54	0.51	0.52	0.57	0.53	0.56	0,54	0.60
14	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.97
14	**3**	1.00	1.00	0.97	1.00	1.00	0.98	1,00	0.83
14	**4**	0.52	0.50	0.52	0.51	0.52	0.52	0,52	0.55
14	**5**	0.50	0.62	0.52	0.50	0.50	0.51	0,52	0.60
15	**2**	0.98	0.64	0.95	0.96	0.88	0.99	0,99	0.79
15	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.79
16	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.96
16	**2**	0.99	0.55	0.89	0.96	0.94	0.99	0,99	0.78
16	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.83
17	**1**	0.98	0.98	0.98	0.98	0.98	0.98	0,98	0.95
17	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
17	**3**	0.91	0.48	0.90	0.86	0.90	0.89	0,88	0.82
17	**4**	0.99	0.99	0.99	0.99	0.99	0.99	0,99	0.87
17	**5**	0.98	0.98	0.98	0.98	0.98	0.97	0,98	0.85
17	**6**	0.99	0.99	0.98	0.98	0.99	0.99	0,99	0.95
17	**8**	0.99	0.99	0.98	0.98	0.99	0.99	0,99	0.87
17	**9**	0.98	0.97	1.00	0.99	1.00	0.99	1,00	0.90

Continúa en la página siguiente

Site	Cla	G	P	KC	KN	KM	PCA	NN	SVM
18	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
18	**2**	0.98	0.50	0.96	0.99	0.97	0.98	0,99	0.80
18	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.83
18	**4**	0.97	0.59	0.97	0.97	0.97	0.98	0,91	0.74
19	**2**	0.99	0.53	0.99	0.99	0.99	0.99	0,98	0.72
19	**3**	0.99	0.65	0.96	0.99	0.96	0.98	0,98	0.90
20	**1**	0.85	0.57	0.71	0.75	0.72	0.80	0,81	0.85
20	**2**	0.90	0.50	0.84	0.90	0.86	0.73	0,74	0.77
20	**3**	0.99	0.72	0.99	1.00	0.99	1.00	0,97	0.96
21	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
21	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
21	**4**	1.00	1.00	0.99	1.00	1.00	1.00	1,00	0.95
21	**5**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	0.84
22	**1**	0.98	0.95	0.98	0.99	0.98	0.99	0,98	0.97
22	**3**	0.99	0.56	0.99	0.99	0.99	0.99	0,99	0.75
23	**1**	0.98	0.51	0.95	1.00	0.97	1.00	0,98	0.96
23	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1,00	1.00
23	**4**	1.00	1.00	0.99	0.99	1.00	1.00	1,00	0.98
23	**5**	0.99	0.61	0.99	0.99	0.99	0.99	0,99	0.75
24	**2**	0.98	0.54	0.98	0.99	0.99	0.98	0,98	0.76
24	**3**	0.96	0.50	0.96	0.99	0.97	0.97	0,95	0.80
25	**2**	0.98	0.55	0.98	0.98	0.98	0.98	1,00	0.75
25	**3**	1.00	1.00	1.00	0.95	0.99	1.00	0,99	0.82
25	**4**	0.99	0.52	0.98	0.99	0.99	0.99	0,98	0.77
26	**2**	0.98	0.80	0.97	0.98	0.95	0.98	0,49	0.80
26	**3**	0.51	0.55	0.58	0.50	0.49	0.50	0,99	0.78
27	**1**	0.98	0.56	0.97	0.98	0.97	0.98	0,99	0.85
27	**2**	0.99	0.50	0.99	0.99	0.99	0.99	0,98	0.82
27	**4**	0.98	0.56	0.98	0.99	0.98	0.99	1,00	0.81

Sitio	G	P	KC	KN	KM	PCA	NN	SVM
1	0,96	0,69	0,94	0,95	0,94	0,86	0,95	0,78
2	0,98	0,69	0,96	0,99	0,98	0,98	0,99	0,81
3	0,96	0,86	0,93	0,89	0,94	0,95	0,95	0,85
4	0,96	0,78	0,94	0,94	0,92	0,88	0,91	0,88
5	0,97	0,63	0,98	0,95	0,98	0,98	0,96	0,84
6	0,58	0,58	0,58	0,58	0,58	0,58	0,58	0,58
7	0,92	0,84	0,90	0,90	0,91	0,92	0,92	0,88
8	0,96	0,96	0,94	0,93	0,94	0,96	0,96	0,91
9	0,97	0,90	0,98	0,99	0,98	0,97	0,96	0,86
10	0,93	0,72	0,97	0,94	0,96	0,90	0,96	0,82
11	0,98	0,55	0,95	0,99	0,97	0,96	0,95	0,83
12	0,90	0,76	0,87	0,90	0,87	0,86	0,94	0,84
13	0,93	0,72	0,92	0,92	0,94	0,88	0,94	0,83
14	0,71	0,73	0,71	0,72	0,71	0,72	0,72	0,71
15	0,99	0,82	0,98	0,98	0,94	0,99	0,99	0,79
16	1,00	0,85	0,96	0,99	0,98	1,00	1,00	0,86
17	0,98	0,92	0,98	0,97	0,98	0,98	0,98	0,90
18	0,99	0,77	0,98	0,99	0,98	0,99	0,98	0,84
19	0,99	0,59	0,97	0,99	0,98	0,98	0,98	0,81
20	0,91	0,60	0,85	0,88	0,86	0,83	0,84	0,86
21	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,95
22	0,99	0,75	0,98	0,99	0,99	0,99	0,98	0,86
23	0,99	0,78	0,98	1,00	0,99	1,00	0,99	0,92
24	0,97	0,52	0,97	0,99	0,98	0,98	0,97	0,78
25	0,99	0,69	0,98	0,97	0,98	0,99	0,99	0,78
26	0,74	0,68	0,77	0,74	0,72	0,73	0,74	0,79
27	0,98	0,54	0,98	0,99	0,98	0,98	0,99	0,83

A.5. MAVE

Site	Cla	M-G	M-P	M-KC	M-K	M-KM	M-PCA	M-NN	M-SVM
1	**1**	0.99	0.68	0.99	0.99	0.98	0.89	0.86	0.96
1	**2**	0.97	0.82	0.96	0.96	0.96	0.91	0.74	0.94
1	**4**	0.99	0.84	0.99	0.99	0.99	0.98	0.98	0.97
2	**1**	0.99	0.85	0.99	0.99	0.99	0.98	0.98	0.66
2	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.97
2	**4**	0.94	0.50	0.91	0.98	0.91	0.98	0.85	1.00
3	**1**	0.96	0.98	0.96	0.89	0.96	0.94	0.98	0.95
3	**2**	0.97	0.98	0.96	0.94	0.97	0.98	0.99	0.96
3	**3**	0.94	0.83	0.91	0.96	0.92	0.94	0.95	0.99
3	**5**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
3	**6**	0.87	0.61	0.80	0.68	0.75	0.79	0.67	0.81
3	**7**	0.96	0.59	0.96	0.96	0.93	0.96	0.86	0.98
3	**8**	0.92	1.00	1.00	1.00	1.00	1.00	1.00	1.00
3	**9**	1.00	1.00	0.97	0.95	0.99	1.00	1.00	0.99
4	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
4	**2**	0.95	0.53	0.93	0.93	0.95	0.77	0.77	0.76
4	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
4	**5**	0.91	0.69	0.92	0.93	0.91	0.83	0.82	0.97
5	**1**	0.96	0.70	0.96	1.00	0.98	0.97	0.95	0.99
5	**3**	0.99	0.64	0.99	0.99	0.98	0.99	0.98	0.80
6	**1**	0.50	0.50	0.50	0.50	0.50	0.50	0.50	0.33
6	**2**	0.67	0.67	0.67	0.67	0.67	0.67	0.67	0.46
7	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
7	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
7	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
7	**4**	0.61	0.48	0.63	0.63	0.66	0.58	0.60	0.69
7	**5**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
7	**6**	0.83	0.50	0.76	0.71	0.74	0.83	0.82	0.92
7	**7**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
8	**1**	0.98	0.98	0.94	0.94	0.94	0.97	0.97	0.92
8	**2**	0.92	0.92	0.92	0.92	0.92	0.92	0.92	0.87
8	**3**	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.93
8	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
9	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.98
9	**3**	0.95	0.83	0.97	0.99	0.97	0.98	0.91	0.96
10	**1**	0.98	0.71	0.99	1.00	0.98	0.95	0.90	0.85
10	**2**	0.85	0.65	0.96	1.00	0.91	0.97	0.88	0.96
10	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
11	**1**	0.96	0.50	0.88	0.98	0.94	0.92	0.85	1.00
11	**2**	0.99	0.53	0.99	0.99	0.99	0.97	0.99	0.87

Site	Cla	M-G	M-P	M-KC	M-KN	M-KM	M-PCA	M-NN	M-SVM
11	**3**	0.98	0.67	0.94	0.99	0.96	0.97	0.96	1.00
11	**5**	0.99	0.71	0.99	0.99	0.98	0.98	0.96	0.98
12	**1**	0.99	0.88	0.98	0.99	0.99	0.98	0.99	0.98
12	**2**	0.91	0.70	0.93	0.93	0.92	0.71	0.71	0.94
12	**3**	0.83	0.71	0.83	0.82	0.83	0.76	0.76	0.82
12	**5**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
13	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
13	**3**	0.99	0.69	0.99	0.99	0.99	0.86	0.83	0.93
13	**4**	0.97	0.82	0.97	0.97	0.97	0.75	0.72	0.93
13	**5**	0.99	0.77	0.99	0.99	0.99	0.99	0.98	0.94
14	**1**	0.63	0.60	0.62	0.63	0.62	0.56	0.56	0.63
14	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
14	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
14	**4**	0.64	0.54	0.64	0.64	0.64	0.52	0.52	0.64
14	**5**	0.60	0.67	0.60	0.60	0.59	0.51	0.51	0.70
15	**2**	0.99	0.74	0.98	0.99	0.99	0.99	0.99	0.78
15	**3**	0.97	1.00	1.00	1.00	1.00	1.00	1.00	0.99
16	**1**	1.00	0.99	1.00	1.00	1.00	1.00	1.00	0.99
16	**2**	0.99	0.62	0.93	0.99	0.99	0.99	0.76	0.86
16	**4**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	0.99
17	**1**	0.98	0.98	0.98	0.98	0.98	0.98	0.98	0.96
17	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
17	**3**	0.92	0.55	0.90	0.86	0.90	0.89	0.88	0.92
17	**4**	1.00	1.00	1.00	1.00	1.00	0.99	0.99	1.00
17	**5**	0.98	0.94	0.98	0.98	0.98	0.97	0.98	0.92
17	**6**	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.97
17	**8**	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98
17	**9**	0.97	0.99	1.00	0.99	1.00	1.00	1.00	0.94
18	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
18	**2**	0.98	0.59	0.96	0.99	0.97	0.98	0.99	0.96
18	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
18	**4**	0.98	0.63	0.97	0.98	0.98	0.98	0.91	0.88
19	**2**	0.99	0.69	0.99	0.99	0.99	0.99	0.98	0.71
19	**3**	0.99	0.76	0.98	1.00	0.99	0.98	0.98	1.00
20	**1**	0.98	0.65	0.92	0.93	0.92	0.80	0.81	0.96
20	**2**	0.95	0.53	0.95	0.96	0.95	0.73	0.74	0.95
20	**3**	0.99	0.80	0.98	1.00	0.99	1.00	0.97	1.00
21	**1**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
21	**3**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
21	**4**	1.00	1.00	1.00	1.00	0.99	1.00	1.00	0.98

Site	Cla	M-G	M-P	M-KC	M-KN	M-KM	M-PCA	M-NN	M-SVM
21	**5**	0.98	1.00	1.00	1.00	1.00	1.00	1.00	0.98
22	**1**	0.98	0.95	0.98	0.99	0.99	0.99	0.98	1.00
22	**3**	0.99	0.64	0.99	0.99	0.99	0.99	0.99	0.77
23	**1**	0.98	0.53	0.95	1.00	0.97	1.00	0.98	1.00
23	**2**	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
23	**4**	1.00	1.00	1.00	0.99	1.00	1.00	1.00	1.00
23	**5**	0.99	0.70	0.99	0.99	0.99	0.99	0.99	0.64
24	**2**	0.99	0.58	0.99	0.98	0.99	0.98	0.99	0.75
24	**3**	0.97	0.52	0.95	0.99	0.97	0.97	0.92	0.99
25	**2**	0.98	0.62	0.97	0.98	0.98	0.98	0.98	0.87
25	**3**	1.00	1.00	1.00	0.97	0.99	1.00	1.00	0.98
25	**4**	0.99	0.52	0.98	0.99	0.99	0.99	0.99	0.95
26	**2**	0.98	0.85	0.98	0.98	0.98	0.98	0.97	0.94
26	**3**	0.99	0.69	0.99	0.99	0.99	0.50	0.49	0.93
27	**1**	0.98	0.54	0.97	0.98	0.99	0.98	0.98	0.99
27	**2**	0.99	0.50	0.99	0.99	0.99	0.99	0.99	0.89
27	**4**	0.98	0.67	0.97	0.98	0.99	0.99	0.98	0.99

Sitio	M-G	M-P	M-KC	M-KNN	M-KM	M-PCA	M-NN	M-SVM
1	0,98	0,78	0,98	0,98	0,98	0,93	0,86	0,80
2	0,98	0,78	0,97	0,99	0,97	0,99	0,94	0,82
3	0,95	0,87	0,94	0,92	0,94	0,95	0,93	0,86
4	0,96	0,80	0,96	0,97	0,96	0,90	0,90	0,89
5	0,97	0,67	0,97	0,99	0,98	0,98	0,97	0,85
6	0,58	0,58	0,58	0,58	0,58	0,58	0,58	0,57
7	0,92	0,85	0,91	0,91	0,91	0,91	0,92	0,88
8	0,96	0,96	0,95	0,95	0,95	0,96	0,96	0,91
9	0,97	0,91	0,98	0,99	0,98	0,99	0,96	0,88
10	0,95	0,79	0,98	1,00	0,96	0,97	0,93	0,84
11	0,98	0,60	0,95	0,99	0,97	0,96	0,94	0,84
12	0,93	0,82	0,93	0,94	0,93	0,86	0,86	0,87
13	0,99	0,82	0,99	0,99	0,99	0,90	0,88	0,85
14	0,77	0,76	0,77	0,77	0,77	0,72	0,72	0,74
15	0,98	0,87	0,99	0,99	0,99	0,99	0,99	0,82
16	1,00	0,87	0,98	1,00	1,00	1,00	0,92	0,84
17	1,01	0,96	1,01	1,00	1,01	0,98	0,98	0,92
18	0,99	0,80	0,98	0,99	0,99	0,99	0,98	0,89
19	0,99	0,72	0,99	0,99	0,99	0,98	0,98	0,83
20	0,97	0,66	0,95	0,96	0,96	0,84	0,84	0,88
21	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,95
22	0,99	0,79	0,99	0,99	0,99	0,99	0,99	0,86
23	0,99	0,81	0,98	1,00	0,99	1,00	0,99	0,92
24	0,98	0,55	0,97	0,99	0,98	0,98	0,96	0,79
25	0,99	0,71	0,98	0,98	0,99	0,99	0,99	0,80
26	0,99	0,77	0,99	0,99	0,99	0,74	0,73	0,83
27	0,98	0,57	0,98	0,98	0,99	0,99	0,99	0,84

Bibliografía

[1] J. Alcalá-fdez, L. Sánchez, S. García, M. J. D. Jesus, S. Ventura, J. M. Garrell, J. Otero, J. Bacardit, V. M. Rivas, J. C. Fernández y F. Herrera. *Keel: A software tool to assess evolutionary algorithms for data mining problems*. *Soft Computing*, 3(14):307–318, 2009.

[2] G. Alonso, F. Casati, H. Kuno y V. Machiraju. *Web services: Concepts, architectures and applications*. Springer Berlin Heidelberg, 2004.

[3] F. Angiulli, G. Ianni y L. Palopoli. *On the complexity of mining association rules*. En *National Symposium on Advanced Database Systems*, página 34–40, 2001.

[4] A. Arning, R. Agrawal y P. Raghavan. *A Linear Method for Deviation Detection in Large Databases*. En *Knowledge Discovery and Data Mining*, páginas 164–169, 1996.

[5] P. Baldi y K. Hornik. *Neural networks and principal component analysis: learning from examples without local minima*. *Neural Networks*, 2:53–58, 1989.

[6] M. Bernard, L. Boyera, A. Habrardb y M. Sebban. *Learning probabilistic models of tree edit distance*. *Pattern Recognition*, 41:2611–2629, 2008.

[7] P. A. Bernstein y L. M. Haas. *Information integration in the enterprise*. *Communication ACM*, 51(9):72–79, 2008.

[8] P. Bille. *A survey on tree edit distance and related problems*. *Theoretical Computer Science*, 337:217–239, 2005.

[9] M. Birattari, Z. Yuan, P. Balaprakash y T. Stützle. *F-race and iterated f-race: An overview*. En T. Bartz-Beielstein, M. Chiarandini, L. Paquete y M. Preuss, editores, *Experimental Methods for the Analysis of Optimization Algorithms*, páginas 311–336. Springer Berlin Heidelberg, 2010.

[10] C. M. Bishop. *Novelty detection and neural network validation*. En *IEEE Proceedings on Vision and Image Signal Processing*, volumen 141, páginas 217–222, 1994.

[11] C. M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, 1995.

[12] C. Blum, J. Puchinger, G. R. Raidl y A. Roli. *Hybrid metaheuristics in combinatorial optimization: A survey*. *Applied Soft Computing*, 11(6):4135–4151, 2011.

[13] J. Blythe, D. Kapoor, C. Knoblock y K. Lerman. *Information integration for the masses*. *Journal of Universal Computer Science*, 14:1811–1837, 2008.

[14] E. Bonabeau, M. Dorigo y G. Theraulaz. *Swarm intelligence: From natural to artificial systems*. Oxford University Press, 1999.

[15] R. G. Brereton. *One-class classifiers*. *Journal of Chemometrics*, página 225–246, 2011.

[16] M. Bugalho y A. Oliveira. *Inference of regular languages using state merging algorithms with search*. *Pattern Recognition*, 38:1457–1467, 2005.

[17] B. Bullnheimer, R. F. Hartl y C. Strauss. *A new rank-based version of the ant system: A computational study*. *Central European Journal for Operations Research and Economics*, páginas 25–38, 1999.

[18] B. Bullnheimer, R. F. Hartl y C. Strauss. *A new rank based version of the ant system - a computational study*. *Central European Journal for Operations Research and Economics*, 7:25–38, 1997.

[19] G. Carpenter, S. Grossberg y D. Rosen. *Art 2-a: an adaptive resonance algorithm for rapid category learning and recognition*. *Neural Networks*, 4:493–504, 1991.

[20] R. Carrasco y J. Oncina. *Learning stochastic regular grammars by means of a state merging method*. En *International Colloquium on Grammatical Inference and Applications*, página 139–152, 1994.

[21] R. Carrasco, J. Oncina y J. Calera-Rubio. *Stochastic inference of regular tree languages. machine learning*. *Machine Learning*, 44:185–197, 2001.

[22] R. Carrasco y J. Rico-Juan. *A similarity between probabilistic tree languages: application to xml document families*. *Pattern Recognition*, 36:2197–2199, 2003.

[23] J. Casillas, O. Cordón, I. F. de Viana y F. Herrera. *Learning cooperative linguistic fuzzy rules using the Best-Worst Ant System algorithm: Research Articles*. *International Journal of Intelligent System*, 20(4):433–452, 2005.

[24] J. Casillas, O. Cordón, I. F. de Viana y F. Herrera. *Learning cooperative linguistic fuzzy rules using the Best-Worst Ant System algorithm: Research Articles*. *International Journal of Intelligent System*, 20(4):433–452, 2005.

[25] V. Chandola, A. Banerjee y V. Kumar. *Anomaly detection: A survey*. *ACM Computing Surveys*, 41(3), 2009.

[26] C.-H. Chang, M. Kayed, M. R. Girgis y K. F. Shaalan. *A survey of web information extraction systems*. *IEEE Trans. on Knowl. and Data Eng.*, 18:1411–1428, 2006.

[27] C.-H. Chang, Y.-L. Lin, K.-C. Lin y M. Kayed. *Page-level wrapper verification for unsupervised web data extraction*. En *WISE*, volumen 1, páginas 454–467, 2013.

[28] S. Chawathe. *Comparing hierarchical data in external memory*. En *International Conference on Very Large Data Bases*, páginas 90–101, 1999.

[29] N. V. Chawla. *Data mining and knowledge discovery handbook*, capítulo Data Mining for Imbalanced Datasets: An Overview, páginas 875–886. Springer, 2010.

[30] S. Chebrolua y A. Abrahama. *Feature deduction and ensemble design of intrusion detection systems*. *Computers & Security*, 24:295–307, 2005.

[31] O. Cordón, I. F. de Viana y F. Herrera. *Analysis of the Best-Worst Ant System and its variants on the QAP*. En *International Workshop on Ant Algorithms*, páginas 228–234, 2002.

[32] O. Cordón, I. F. de Viana y F. Herrera. *Analysis of the Best-Worst Ant System and its variants on the QAP*. En *International Workshop on Ant Algorithms*, páginas 228–234, 2002.

[33] O. Cordón, I. F. de Viana y F. Herrera. *Analysis of the Best-Worst Ant System ant its variants on the TSP*. *Mathware Soft Computing*, 1:177–192, 2002.

[34] O. Cordón, I. F. de Viana y F. Herrera. *Analysis of the Best-Worst Ant System ant its variants on the TSP*. *Mathware Soft Computing*, 1:177–192, 2002.

[35] O. Cordón, F. Herrera y T. Stützle. *A review on the ant colony optimization metaheuristic: Basis, models and new trends*. *Mathware & Soft Computing*, 9:141–175, 2002.

[36] O. Cordón, F. Herrera y T. Stützle. *A review on the ant colony optimization metaheuristic: Basis, models and new trends*. *Mathware & Soft Computing*, 9:141–175, 2002.

[37] O. Cordón, F. H. Herrera y L. Moreno. *Integración de conceptos de computación evolutiva en un nuevo modelo de colonias de hormiga*. En *VIII Conferencia de la Asociación Española para la Inteligencia Artificial*, volumen 2, páginas 98–105, 1999.

[38] O. Cordón, I. naaki Fernández de Viana, F. Herrera y L. Moreno. *A new aco model integrating evolutionary computation concepts: The best-worst ant system*. En *Abstract proceedings of ANTS2000 - From Ant Colonies to Artificial Ants: A series of International Workshops on Ant Algorithms*, páginas 22–29, 2000.

[39] O. Cordón, I. naki Fernández de Viana y F. Herrera. *Analysis of the best-worst ant system and its variants on the qap. Lecture Notes in Computer Science*, 2463:228–234, 2002.

[40] O. Cordón, I. F. de Viana, F. Herrera y L. Moreno. *A new aco model integrating evolutionary computation concepts: The best-worst ant system*. En *From Ant Colonies to Artificial Ants*, páginas 22–29, 2000.

[41] M. Dash y H. Liu. *Feature selection for classification. Intelligent Data Analysis*, 1:131–156, 1997.

[42] O. de Oliveira y A. da Silva. *Automatic verification of data extracted from the web*. En *Brazilian Symposium on Databases*, página 56–71, 2003.

[43] I. F. de Viana, P. J. Abad, J. L. Álvarez y J. L. Arjona. *Applying one class classifier techniques to reduce maintenance costs of eai*. En *International Joint Conference on Software Technologies*, páginas 41–46, 2011.

[44] I. F. de Viana, P. J. Abad, J. L. Arjona y J. L. Álvarez. *Toward one class classifier techniques applied to verifier information*. En *6th Iberian Conference on Information Systems and Technologies (CISTI)*, 2011.

[45] I. F. de Viana, P. J. Abad, J. L. Arjona y J. L. Álvarez. *Optimizating one-class techniques applied to verify information extractors*. En *Information Systems and Technologies (CISTI), 2012 7th Iberian Conference on Publication*, página 16, 2012.

[46] I. F. de Viana, P. J. Abad, J. L. Arjona y J. L. Álvarez. *Verificación de la información extraída por wrappers web usando algoritmos basados en colonias de hormigas*. En *15th Conference on Software Engineering and Databases*, página •, 2012.

[47] I. F. de Viana, J. L. Arjona y J. L. Álvarez. *Verificación de wrappers web: Nuevas ideas*. En *Conference on Software Engineering and Databases*, 2010.

[48] I. F. de Viana, J. L. Arjona y J. L. Álvarez. *Selección de características para mejorar la información del comprobador de modelos de eai. IEEE América Latina*, 8(2), 2011.

[49] I. F. de Viana, J. L. Arjona, J. L. Álvarez y P. Abad. *Selección de características para mejorar los modelos de verificación de información en EAI*. En *Conference on Software Engineering and Databases*, páginas 21–32, 2009.

[50] I. F. de Viana, I. Hernandez, P. Jiménez, C. R. Rivero y H. A. Sleiman. *Integrating deep-web information sources*. En *8th International Conference on Practical Applications of Agents and Multiagent Systems*, 2010.

[51] I. F. de Viana, I. Hernández, P. Jiménez, C. R. Rivero y H. A. Sleiman. *Integrating deep-web information sources*. Informe técnico, The Distributed Group, 2010.

[52] I. F. de Viana, J. L. Arjona, J. L. Álvarez y P. J. Abad. *Feature selection to improve information verifier models in eai). Revista IEEE América Latina*, 8:158–163, 2010.

[53] J. Demsar. *Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research*, 7:1–30, 2006.

[54] W. Dixon. *Analysis of extreme values. The Annals of Mathematical Statistics*, 4(21):488–506, 1950.

[55] M. Dorigo y T. Stützle. *The ant colony optimization metaheuristic: Algorithms, applications, and advances*. En F. Glover y G. Kochenberger, editores, *Handbook of Metaheuristics*, volumen 57 de *International Series in Operations Research & Management Science*, capítulo 9, páginas 250–285. Springer, 2003.

[56] M. Dorigo, G. Di Caro y L. M. Gambardella. *Ant algorithms for discrete optimization. Artificial Life*, 5(2): 137–172, 1999.

[57] M. Dorigo y G. Di Caro. *New ideas in optimization*, capítulo The ant colony optimization meta-heuristic, páginas 11–32. McGraw-Hill Ltd., UK, 1999.

[58] M. Dorigo y G. Di Caro. *New ideas in optimization*, capítulo The Ant Colony Optimization meta-heuristic, páginas 11–32. McGraw Hill, 1999.

[59] M. Dorigo y L. M. Gambardella. *Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation*, páginas 53–66, 1997.

[60] M. Dorigo y L. Gambardella. *Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation*, 1:53–66, 1997.

[61] M. Dorigo, V. Maniezzo y A. Colorni. *The ant system: Optimization by a colony of cooperating agents. IEEE Tansactions on systems, Man, and Cybernetics*, páginas 29–41, 1996.

[62] M. Dorigo y T. Stützle. *Ant colony optimization*. Bradford Company, Scituate, MA, USA, 2004.

[63] Q. Duan, T. W. Liao y H. Yi. *A comparative study of different local search application strategies in hybrid metaheuristics. Applied Soft Computing*, 2012.

[64] J. Dubois-Lacoste, M. López-Ibáñez y T. Stützle. *Automatic configuration of state-of-the-art multi-objective optimizers using the tp+pls framework*. En *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, páginas 2019–2026, 2011.

[65] R. B. Emilio Ferrara. *Automatic wrapper adaptation by tree edit distance matchy. Combinations of Intelligent Methods and Application*, 20:41–53, 2011.

[66] M. Ernst, J. C. nd W. Griswold y D. Notkin. *Dynamically discovering likely program invariants to support program evolution. IEEE Transactions on Software Engineering*, 27(2), 2001.

[67] M. Ernst, J. Perkins, P. Guo, S. McCamant, C. Pacheco, M. Tschantz y C. Xiao. *The daikon system for dynamic detection of likely invariants. Science of Computer Programming*, 6(1-3), 2007.

[68] E. Ferrara y R. Baumgartner. *Design of automatically adaptable web wrappers. CoRR*, abs/1103.1254, 2011.

[69] E. Ferrara, P. D. Meo, G. Fiumara y R. Baumgartner. *Web data extraction, applications and techniques: A survey. Knowledge-Based Systems*, 70:301–323, 2014.

[70] R. Z. Frantz, R. Corchuelo y C. Molina-Jiménez. *A proposal to detect errors in enterprise application integration solutions. Journal of Systems and Software*, 85(3):480 – 497, 2012.

[71] S. S. Fu, J. Yang, J. Laredo, Y. Huang, H. Chang, S. Kumaran, J.-Y. Chung y Y. Kosov". *"solution templates tool for enterprise business applications integration". "Sensor Networks, Ubiquitous, and Trustworthy Computing, International Conference on"*, 0:314–319, 2008.

[72] K. Fukunaga. *Introduction to statistical pattern recognition*. Morgan Kaufmann, 1995.

[73] S. García, A. Fernández, J. Luengo y F. Herrera. *A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Computing*, 13:959–977, 2009.

[74] S. García, A. Fernández, J. Luengo y F. Herrera. *Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Information Sciences*, 180(10):2044–2064, 2010.

[75] M. R. Garey y D. S. Johnso. *Computers and intractability: A guide to the theory of np-completeness*. W.H. Freeman, 1979.

[76] F. Glover y G. Kochenberger. *Handbook of metaheuristics*. Kluwer Academic Publishers, 2003.

[77] T. Goan, N. Benson y O. Etzioni. *A grammar inference algorithm for the world wide web*. En *AAAI Spring Symposium on Machine Learning in Information Access*, 1996.

[78] S. Goss, S. Aron, J. Deneubourg y J. Pasteels. *Self-organized shortcuts in the argentine ant*. *Naturwissenschaften*, 76(12):579–581, 1989.

[79] M. Grochowski y N. Jankowski. *Comparison of instance selection algorithms ii. results and comments*. En *International Conference on Artificial Intelligence and Soft Computing*, páginas 580–585, 2004.

[80] F. Grubbs. *Sample criteria for testing outlying observations*. *The Annals of Mathematical Statistics*, 1(21): 27–58, 1950.

[81] I. Guyon y A. Elisseeff. *An introduction to variable and feature selection*. *Journal of Machine Learning Research Homepage*, 3:1157–1182, 2003.

[82] M. A. Hall y G. Holmes. *Benchmarking attribute selection techniques for discrete class data mining*. *Transactions on Knowledge and Data Engineering*, 15(6):1437–1447, 2003.

[83] S. Harmeling, G. Dornhege, D. Tax, F. Meinecke y K.-R. Müller. *From outliers to prototypes: Ordering data*. *Neurocomputing*, páginas 1608–1618, 2006.

[84] X. He, D. Cai y P. Niyogi. *Laplacian score for feature selection*. En *Neural Information Processing Systems*, páginas 67–74. MIT Press, 2005.

[85] X. He, D. Cai y P. Niyogi. *Locality preserving projections*. En *Neural Information Processing Systems*, páginas 67–77. MIT Press, 2005.

[86] J. Hertz, A. Krogh y R. Palmer. *Introduction to the theory of neural computation*. Addison Wesley, 1991.

[87] M. Hilario y A. Kalousis. *Characterizing learning models and algorithms for classification*. Informe técnico, University of Geneva, 2006.

[88] J. Hippa, U. Günntzer y G. Nakhaeizadeh. *Algorithms for association rule mining: A general survey and comparison*. En *International Conference on Knowledge Discovery and Data Mining*, página 58–64, 2000.

[89] V. J. Hodge y J. Austin. *A survey of outlier detection methodologies*. *Artificial Intelligence Review*, 22: 2004, 2004.

[90] J. Hodges y E. Lehmann. *Ranks methods for combination of independent experiments in analysis of variance*. *Annals of Mathematical Statistics*, 33:482–497, 1962.

[91] S. Holm. *A simple sequentially rejective multiple test procedure*. *Scandinavian Journal of Statistics*, 6(2): 65–70, 1979.

[92] J. Doak. *An evaluation of feature selection methods and their application to computer security*. Informe técnico, Univ. of California at Davis, Dept. Computer Science, 1992.

[93] N. Japkowicz. *Concept learning in the absence of counterexamples: an autoassociation-based approach to classification*. Tesis doctoral, Rutgers University, New Brunswick, NJ, USA, 1999.

[94] N. Japkowicz, C. Myers y M. Gluck. *A novelty detection approach to classification*. En *International Joint Conferences on Artificial Intelligence*, páginas 518–523, 1995.

[95] P. Juszczak. *A study on one-class classification and active learning*. Tesis doctoral, Delft University of Technology, 2006.

[96] P. Juszczaka, D. M. Tax, E. Pe¸kalska y R. P. Duina. *Minimum spanning tree based one-class classifier. Neurocomputing*, página 225–246, 2009.

[97] T. Kohonen. *Self-organizing maps*. Springer-Verlag, Heidelberg, 1995.

[98] N. Kushmerick. *Regression testing for wrapper maintenance*. En *AAAI '99/IAAI '99: Proceedings of the sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference innovative applications of artificial intelligence*, páginas 74–79, 1999.

[99] N. Kushmerick. *Wrapper induction: Efficiency and expressiveness. Artificial Intelligence*, 118(1-2):15–68, 2000.

[100] N. Kushmerick. *Wrapper verification*. En *Proceedings of the 9th international conference on World Wide Web*, páginas 79–94, 2000.

[101] T. Lane. *Machine learning techniques for the computer security domain of anomaly detection*. Tesis doctoral, Purdue University, 2000.

[102] M. Lauer. *A mixture approach to novelty detection using training data with outliers*. En *European Conference on Machine Learning*, página 300–311, 2001.

[103] T. Lee y Y. Yang. *Constraint-based wrapper specification and verification for cooperative information systems. Information Systems*, 29:617–636, 2004.

[104] K. Lerman y C. A. Knoblock. *Wrapper maintenance*. En *Encyclopedia of Database Systems*. Springer, 2009.

[105] K. Lerman, S. N. Minton y C. A. Knoblock. *Wrapper maintenance: A machine learning approach. Journal of Artificial Intelligence Research*, 18(0):2003, 2003.

[106] T. W. Liao, E. Hu y T. R. Tiersch. *Metaheuristic approaches to grouping problems in high-throughput cryopreservation operations for fish sperm. Applied Soft Computing*, 12(8):2040–2052, 2012.

[107] D. Linthicum. *Enterprise application integration*. Addison-Wesley Professional, 2000.

[108] H. Liu y H. Motoda. *Feature selection for knowledge discovery and data mining*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[109] H. Liu y L. Yu. *Toward integrating feature selection algorithms for classification and clustering. IEEE Transactions on Knowledge and data engineering*, 17:491–502, 2005.

[110] J. Madhavan, L. Afanasiev, L. Antova y A. Y. Halevy. *Harnessing the deep web: Present and future*. En *CIDR*, 2009.

[111] J. Madhavan, S. Cohen, A. Y. Halevy, S. R. Jeffery, X. L. Dong, D. Ko y C. Yu. *Web-scale data integration: You can afford to pay as you go*. En *CIDR*, páginas 342–350, 2007.

[112] L. M. Manevitz y M. Yousef. *One-class svms for document classification. Journal of Machine Learning Research*, 2:139–154, 2002.

[113] M. Markou y S. Singh. *Novelty detection: a review part 1: statistical approaches. Signal Processing*, 83: 2481–2497, 2003.

[114] O. Mazhelis. *One-class classifiers: A review and analysis of suitability in the context of mobile-masquerader detection. Advances in end-user data-mining techniques*, páginas 29–48, 2006.

[115] R. McCann, B. AlShebli, Q. Le, H. Nguyen, L. Vu y A. Doan. *Mapping maintenance for data integration systems*. En *VLDB '05: Proceedings of the 31st international conference on Very large data bases*, páginas 1018–1029, 2005.

[116] R. McCann, B. AlShebli, Q. Le, H. Nguyen, L. Vu y A. Doan. *Mapping maintenance for data integration systems*. En *International Conference on Very Large Databases*, páginas 1018–1029, 2005.

[117] R. Mitkov. *The oxford handbook of computational linguistics (oxford handbooks in linguistics s.)*. Oxford University Press, 2003.

[118] M. Mohri, A. Rostamizadeh y A. Talwalkar. *Foundations of machine learning*. The MIT Press, 2012.

[119] P. M. Narendra y K. Fukunaga. *A branch and bound algorithm for feature subset selection*. *Transactions on Computers*, 26(9):917–922, 1977.

[120] B. C. nd B. Roustant y M. Brette. *Documentum eci self-repairing wrappers: performance analysis*. En *ACM SIGMOD International Conference on Management of Data*, página 708–717, 2006.

[121] R. M. Neal y G. E. Hinton. *A view of the em algorithm that justifies incremental, sparse, and other variants*, página 355–368. MIT Press, 1999.

[122] I. H. Osman y J. P. Kelly. *Meta-heuristics: Theory and applications*. Kluwer Academic Publishers, 1996.

[123] A. Papadopoulos. *Nearest neighbor search: A database perspective*. Springer Verlag, 2004.

[124] E. Parzen. *On estimation of a probability density function and mode*. *Annals of Mathenatical Statistics*, 33:1065–1076, 1962.

[125] J. M. Pasteels, J. L. Deneubourg y S. Goss. *Self-organization mechanisms in ant societies (i):trail recruitment to newly discovered food sources. From individual to collective behavior in social insects les Treilles Workshop*, 54:155–175, 1987.

[126] E.-H. Pek, X. Li y Y. Liu. *Web wrapper validation*. volumen 1, páginas 388–393. 2003.

[127] G. Piatetsky-Shapiro. *Discovery, analysis and presentation of strong rules*. En *Knowledge Discovery in Databases*, páginas 229–248. AAAI Press, 1991.

[128] V. Poosala, P. J. Haas, Y. E. Ioannidis y E. J. Shekita. *Improved histograms for selectivity estimation of range predicates*. En *International conference on Management of data*, páginas 294–305, 1996.

[129] X. Qin y W. Lee. *Statistical causality analysis of infosec alert data*. En *International Symposium on Recent Advances in Intrusion Detection*, página 73–93, 2003.

[130] D. R y P. Hart. *Pattern classification and scene analysis*. Wiley and Sons, 1973.

[131] P. E. H. R. O. Duda y D. G. Stork. *Pattern classification*. John Wily & Son, 2000.

[132] L. Rabiner. *A tutorial on hidden markov models and selected applications in speech recognition*. En *Proceedings of the IEEE*, páginas 257–286, 1989.

[133] O. Raz, P. Koopman y M. Shaw. *Semantic anomaly detection in on-line data sources*. En *International Conference on Software Engineering*, páginas 302–312, 2002.

[134] D. Rorabacher. *Statistical treatment for rejection of deviant values: Critical values of dixon q parameter and related subrange ratios at the 95 percent confidence level. Analytical Chemistry*, 2(83):139–146, 1991.

[135] T. Z. S. Lange y S. Zilles. *Learning indexed families of recursive languages from positive data: A survey*. *Theoretical Computer Science*, 397:194–232, 2008.

[136] G. Salton. *Automatic text processing: The transformation, analysis, and retrieval of information by computer*. Addison-Wesley, 1989.

[137] I. Sanz, M. Mesiti, G. Guerrini y R. Berlanga. *Fragment-based approximate retrieval in highly heterogeneous xml collections. Data and Knowledge Engineering*, 64(1):266–293, 2008.

[138] S. Sarawagi. *Information extraction. Foundations and Trends in Databases*, 1(3):261–377, 2008.

[139] B. Scholkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola y R. C. Williamson. *Estimating the support of a high-dimensional distributio. Neural Computation*, 13:1443–1471, 2001.

[140] R. H. Shumway y D. S. Stoffer. *Time series analysis and its applications*. Springer, 2000.

[141] J. V. B. Soares, J. J. G. Le, R. M. Cesar, H. F. Jelinek, M. J. Cree y S. Member. *Retinal vessel segmentation using the 2-d gabor wavelet and supervised classification. IEEE Transaction on Medical Imaging*, 25: 1214–1222, 2006.

[142] T. Stützle y H. H. Hoos. *Max-min ant system. Future Gener. Comput. Syst.*, páginas 889–914, 2000.

[143] T. Stützle y H. H. Hoos. *MAX-MIN Ant System. Journal Future Generation Computer Systems*, 16(9): 889–914, 2000.

[144] T. Sukchotrata, S. B. Kimb y F. Tsungc. *One-class classification-based control charts for multivariate process monitoring. IEEE Transactions*, 42:107–120, 2009.

[145] D. M. J. Tax. *One-class classification, concept learning in the absence of counter example*. Tesis doctoral, Delft University of Technology, 2001.

[146] J. Tekli, R. Chbeir y K. Yétongnon. *A fine-grained xml structural comparison approach*. En *ER International Conference on Conceptual Modeling*, páginas 582–598, 2007.

[147] F. Thollard, P. Dupont y C. de la Higuera. *Probabilistic dfa inference using kullback- leibler divergence and minimalit*. En *International Conference on Machine Learning*, 2001.

[148] I. Triguero, S. García y F. Herrera. *Differential evolution for optimizing the positioning of prototypes in nearest neighbor classification. Pattern Recognition*, 44:901–916, 2011.

[149] C. E. Tsourakakis y G. Paliourast. *Wewra: An algorithm for wrapper verification*. Informe técnico CMU-ML-09-100, Carnegie Mellon University, 2009.

[150] C. E. Tsourakakis y G. Paliourast. *Wewra: An algorithm for wrapper verification*. Technical report, Carnegie Mellon University, marzo 2009.

[151] N. Ullman. *Elementary statistics, an applied approach*. Wiley and Sons, 1978.

[152] S. D. Villalba y P. Cunningham. *An evaluation of dimension reduction techniques for one-class classification. Artificial Intelligence Review*, 27(4):273–294, 2007.

[153] S. D. Villalba y P. Cunningham. *An evaluation of dimension reduction techniques for one-class classification. Artificial Intelligence Review*, 27(4):273–294, 2007.

[154] S. Voss, S. Martello, I. Osman y C. Roucairol. *Meta-heuristics: Advances and trends in local search paradigms for optimization*. Kluwer Academic Publishers, 1999.

[155] J. Weiss. *Aligning relationships: Optimizing the value of strategic outsourcing*. Informe técnico, IBM, 2005.

[156] D. Wilson y T. Martinez. *Reduction techniques for instance-based learning algorithms. Machine Learning*, 38:257–286, 2000.

[157] C. W. Y. Wong, K. hung Lai y T. C. E. Cheng. *Value of information integration to supply chain management: Roles of internal and external contingencies. J. of Management Information Systems*, 28(3):161–200, 2012.

[158] T. Wong y W. Lam. *Adapting web information extraction knowledge via mining site-invariant and site-dependent features. ACM Transactions on Internet Technology*, 7(1), 2007.

[159] T. Wong, W. Lam y W. Wang. *Beyond supervised learning of wrappers for extracting information from unseen web sites*. En *International Conference on Intelligent Data Engineering and Automated Learning*, página 725–733, 2003.

[160] J.-M. Yang, R. Cai, Y. Wang, J. Zhu, L. Zhang y W.-Y. Ma. *Incorporating site-level knowledge to extract structured data from web forums*. En *International Conference on World Wide Web*, páginas 181–190, 2009.

[161] A. Ypma y R. Duin. *Support objects for domain approximation*. En *International Conference on Artificial Neural Networks*, páginas 2–4, 1998.

This document was typeset on November 13, 2015 at 17:30 using class RG-BOOK α2.14 for LaTeX$_{2\varepsilon}$. As of the time of writing this document, this class is not publicly available since it is in alpha version. Only members of The Distributed Group are using it to typeset their documents. Should you be interested in giving forthcoming public versions a try, please, do contact us at contact@tdg-seville.info. Thanks!