UNIVERSIDAD AUTÓNOMA DE CHIHUAHUA

FACULTAD DE INGENIERÍA

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

UN NUEVO ALGORITMO BASADO EN APRENDIZAJE DE ENSAMBLE PARA EL PROBLEMA DE DETECCIÓN DE ANOMALÍAS

POR:

ING. LUIS ANGEL PEREYRA VILLANUEVA

TESIS PRESENTADA COMO EQUIVALENTE PARA OBTENER EL GRADO DE MAESTRO EN INGENIERÍA EN COMPUTACIÓN

CHIHUAHUA, CHIH., MÉXICO **MARZO DE 2019**

Un nuevo algoritmo basado en aprendizaje de ensamble para el problema de detección de anomalías. Tesis presentada por Ing. Luis Angel Pereyra Villanueva como requisito parcial para obtener el grado de Maestría En Ingeniería En Computación, ha sido aprobada y aceptada por

M.I. Javier González Cantú
Director de la Facultad de Ingeniería

Dr. Alejandro Villalobos Aragón
Secretario de Investigación y Posgrado

M.S.I. Karina Rocío Requena Yáñez
Coordinadora Académica

Dr. Luis Carlos González Gurrola
Director de Tesis

Fecha
Febrero de 2019

Comité:
Dr. Luis Carlos González Gurrola
Dr. Raúl Monroy Borja
Dra. Graciela Ramírez Alonso
Dr. Luis Fernando Gaxiola Orduño

Luis Angel Pereyra Villanueva
ESCORZA 9003, COLONIA CENTRO
CHIHUAHUA, MÉXICO

UNIVERSIDAD AUTÓNOMA DE
CHIHUAHUA

28 de febrero de 2019

ING. LUIS ANGEL PEREYRA VILLANUEVA

Presente

En atención a su solicitud relativa al trabajo de tesis para obtener el grado de Maestro en Ingeniería en Computación, nos es grato transcribirle el tema aprobado por esta Dirección, propuesto y dirigido por el director **Dr. Luis Carlos González Gurrola** para que lo desarrolle como tesis, con el título: **"UN NUEVO ALGORITMO BASADO EN APRENDIZAJE DE ENSAMBLE PARA EL PROBLEMA DE DETECCIÓN DE ANOMALÍAS".**

ÍNDICE

Índice de Contenido

FACULTAD DE INGENIERÍA
Circuito No.1, Campus Universitario 2
Chihuahua, Chih., México. C.P. 31125
Tel. (614) 442-95-00
www.fing.uach.mx

UNIVERSIDAD AUTÓNOMA DE
CHIHUAHUA

4. **Escenario 2: Detección de riesgos personales mediante el procesamiento de señales de bio-sensado**
   - 4.1. Revisión de literatura
   - 4.2. Introducción a un enfoque de cómputo distribuido para abordar problemas de detección de anomalías
   - 4.3. Implementación de Bagging-RandomMiner en Apache Spark
   - 4.4. Implementación de One-Class K-means with Randomly-projected features Algorithm (OCKRA) en Apache Spark
   - 4.5. Experimentos y resultados sobre PRIDE
5. **Conclusiones y trabajo futuro**
   - 5.1. Conclusiones
   - 5.2. Trabajo futuro
6. **Publicaciones ligadas a la tesis**
   - 6.1. Revista
   - 6.2. Congreso

Referencias

A. Evaluación de Bagging-RandomMiner en otros dominios de aplicación
   - A.1. Resultados

Solicitamos a Usted tomar nota de que el título del trabajo se imprima en lugar visible de los ejemplares de las tesis.

A T E N T A M E N T E
"Naturam subiecit aliis"

EL DIRECTOR

M.I. JAVIER GONZÁLEZ CANTÚ

FACULTAD DE INGENIERÍA U.A.CH. DIRECCIÓN

EL SECRETARIO DE INVESTIGACIÓN Y POSGRADO

DR. ALEJANDRO VILLALOBOS ARAGÓN

FACULTAD DE INGENIERÍA
Circuito No.1, Campus Universitario 2
Chihuahua, Chih., México. C.P. 31125
Tel. (614) 442-95-00
www.fing.uach.mx

Índice de figuras

Índice de tablas

Lista de algoritmos

Capítulo 1

Introducción. ¿Por qué estudiar este problema?

Vivimos en una sociedad dependiente de la tecnología, donde el uso de dispositivos electrónicos y sistemas informáticos ha formado parte de la vida cotidiana de millones de personas. La interacción con estos artefactos ha logrado facilitar una gran cantidad de actividades laborales y de carácter común para los usuarios, pero, según la tercera ley de *Murphy*, si creemos que algo no puede fallar, lo hará a pesar de todo. Lo que nos indica que, aunque los dispositivo electrónicos y sistemas informáticos tengan un excelente diseño y hallan sido probados en miles de ocasiones, no están exentos de vulnerabilidades que repercuten en ellos y en consecuencia a los usuarios. Tomando esto como punto de partida, dentro de la comunidad científica se originó un área de interés para detectar estos fallos, que en la mayoría de las ocasiones se les denomina anomalías. La detección de anomalías se refiere a discernir las desviaciones de un conjunto de patrones esperados [5], estos pueden ser desde pequeñas fluctuaciones en una serie de datos hasta situaciones donde se comprometa todo el sistema. Por lo tanto, la detección oportuna de una anomalía puede ayudar a prevenir pérdidas en las que se puede incurrir cuando surge un problema.

Éstas fallas han sido un problema relevante en distintos escenarios de aplicación como: intrusión en sesiones de computadora [6], detección de fraudes en tarjetas de crédito [7], detección de riesgos personales [8], entre muchos mas. En este trabajo de investigación se optó por abordar dos escenarios en específico, ambos con un gran impacto científico y de aplicabilidad en situaciones reales. El primero hace referencia a la detección de intrusos en sesiones de computadora [6]. Estos po-

sibles infractores del sistema, al violar la seguridad podrían adquirir información confidencial del usuario provocando pérdidas financiares o documentos personales. Se considera que, cada persona tiene un patrón de movimiento al navegar en un sistema informático, lo que permite construir un perfil de comportamiento normal y utilizarlo para detectar cualquier singularidad que se presente por un atacante al no obedecer dicho comportamiento normal.

El segundo escenario es la detección de riesgos personales mediante el procesamiento de señales de bio-sensado [8], basado en detectar cuando una persona se encuentra en una crisis de salud, un asalto o alguna situación que pueda dañar su integridad física. Esto se puede lograr mediante la lectura de indicadores de salud, como el ritmo cardiaco, actividad eléctrica muscular o la temperatura de la piel. Al recolectar información del perfil del usuario se establece un patrón de comportamiento que nos permite identificar anomalías, éstas nos indicarán cuando la persona necesite de asistencia inmediata.

Una técnica que ha demostrado aumentar significativamente el rendimiento de los clasificadores es la metodología por *ensambles* [9–12], ya que uno de los elementos críticos en un clasificador es la diversidad. Para mitigar esta deficiencia, se crea una combinación de clasificadores variando la selección de atributos o instancias en cada uno de ello. Esto mejora el sesgo que se presenta al utilizar un solo clasificador y reduce la varianza.

Dentro del aprendizaje máquina, una estrategia para mejorar el rendimiento de los clasificadores es haciendo una combinación de ellos. Las técnicas para la combinación de clasificadores, llamadas *ensambles*, aumentan el rendimiento de clasificación porque pueden reducir la varianza y el efecto que tiene el sesgo del clasificador individual [9–12]. Uno de los elementos más críticos en los *ensambles* de clasificadores es la diversidad, esto se puede aumentar variando los objetos que cada clasificador usa para entrenar o los atributos utilizados en cada uno de ellos. Estas dos alternativas y sus combinaciones se han probado con éxito en el contexto de Detección de anomalías. Esto nos motivó a encauzar la investigación en una nueva propuesta basada en *ensambles* para la detección de anomalías oportuna en los dos enfoques mencionados anteriormente.

1.1. Antecedentes

La detección de anomalías es considerado un problema de clasificación de una clase (*OCC* por sus siglas en inglés, *One-Class-Classification*). Los *OCC* son clasificadores que tratan de identificar objetos de una clase específica entre todas las demás, aprendiendo principalmente de un conjunto de entrenamiento que contiene solo los objetos de esa clase. Aunque es considerada una tarea más difícil que el problema de clasificación tradicional, este esquema ha demostrado ser muy eficiente en distintos dominios de aplicación. A continuación, se presentan una serie de investigaciones que muestran la relevancia de los *OCC* en distintos contextos:

Según Khan et al. [13] la mejor forma de clasificar problemas cuya naturaleza es des-balanceada es aplicar algoritmos de *OCC*, estos autores abordaron problemáticas de clasificación de texto, entre ellos, *Authorship Verification*, *Handwritten Digit Recognition*, *Face Recognition Applications*, *Spam Detection* y *Machine Fault Detection*. Comparando técnicas de clasificación de una sola clase y multiclase, les permitió concluir que un algoritmo de *OCC*, particularmente Máquinas de Vectores de Soporte de una clase (*OCSVM* por sus siglas en inglés, *One-Class-Support-Vector-Machine*) obtiene los mejores resultados al compararse con técnicas de aprendizaje máquina basadas en la clasificación múltiple.

Los autores en [7] abordaron un problema basado en el uso de las tarjetas de crédito, debido a la afluencia de transacciones generadas diariamente por los consumidores, están propensos a ser atacados por estafadores que roban dinero de las cuentas de los usuarios haciendo compras no autorizadas. Esta situación es de gran interés para las instituciones bancarias, debido a que es un problema de seguridad que siempre estará presente. El conjunto de datos seleccionado para las experimentaciones se denomina *ccFraud*, esta base de datos consta de 10 millones de instancias para las transacciones genuinas y 50 mil ejemplos para los robos a usuarios, sin duda pertenece a la rama de conjuntos des-balanceados. Los autores implementaron una red neuronal recurrente denominada *PSOAANN* que logró excelentes resultados, alcanzando un desempeño mínimo del $85\,\%$ y un valor máximo del $95\,\%$.

La industria espacial no se queda exenta de la aplicabilidad de los algoritmos de clasificación

de una clase para la detección de anomalías, dado que, el diagnóstico de fallas para naves espaciales es un problema crítico y de carácter primordial, porque el entorno espacial es duro, distante e incierto. Los autores Fujimaki et. al. [14] propusieron un método de detección de anomalías que no necesita de conocimiento previo, este construye un modelo a partir de los datos de telemetría. Estos autores, confirmaron la efectividad del método de detección de anomalías aplicándolo a los datos de telemetría obtenidos de un simulador de un vehículo de transferencia orbital, diseñado para realizar maniobras de encuentro con la Estación Espacial Internacional.

Los autores de [15] basaron su investigación en los sistemas de reconocimiento biométrico, el cual busca verificar la autenticidad de una persona mediante el uso de sus características anatómicas o de comportamiento. Esto se basa en quién es una persona o qué hace una persona, en lugar de lo que una persona sabe (por ejemplo, contraseña). El utilizar sistemas biométricos garantiza una mayor conformidad y seguridad para los usuarios. Específicamente, en este trabajo se analizó el sistema de verificación de firmas, el cual reconoce un individuo analizando la actividad de la firma, esto es, el orden de los trazos, la presión aplicada por el bolígrafo o la velocidad. Después de un análisis, se determinó que la combinación de clasificadores (ensamble) de una clase aunado a una selección de características es la mejor opción para la identificación de los usuarios mediante la firma.

La clasificación de las proteínas (por ejemplo, tipos de dominio, clases estructurales, familias de proteínas) es un tema crucial en la anotación del genoma. La literatura indica que los métodos de clasificación más simples se basan en la comparación por pares de proteínas, sin embargo, los grupos de proteínas conocidos poseen características que ocasionan que la aplicación de algoritmos de clasificación más comunes, por ejemplo, Máquinas de Vectores de Soporte (*SVM*) no desempeñen un buen funcionamiento. Los autores Bánhalmi et al. [16] experimentaron con dos conjuntos de datos de clasificación de proteínas (*COG* y *SCOP40*) tomados del repositorio [17], realizando una comparativa entre clasificadores de una sola clase y multiclase, concluyendo que usar *OCC* mejora un poco respecto a la clasificación, pero disminuye significativamente el tiempo de ejecución.

La seguridad cibernética es un área de interés para múltiples empresas y consumidores, debido

a la importancia que representa la información confidencial de los usuarios, esta puede se utilizada para sobornar, amenazar y/o obtener dinero de manera ilegal. Es por esto que incontables investigadores se han esforzado por detectar intrusos en sesiones de computadora desde distintas perspectivas, como lo es, mediante una secuencia de comandos de UNIX [18–25], los movimientos dinámicos del ratón [26–28], a través del uso del teclado [29–31] y el sistema de navegación de archivos [32–34]. Todos con el objetivo de poder detectar un ataque temprano utilizando el enfoque *OCC*.

Por último, se presenta el contexto para la detección de riesgos personales mediante el procesamiento de señales de bio-sensado, es decir, detectar cuando un usuario se encuentra en una situación que propicie un riesgo y que requiera de ayuda inmediata. Los autores de [8] diseñaron un experimento, el cual tiene la intención de simular escenarios de riesgo que pueden presentarse en la vida cotidiana de los usuarios, como, huir de una situación peligrosa, ser atacados por un agresor, estar inmersos en un sismo, presentar algún cuadro clínico que requiera asistencia inmediata y algunos otros escenarios de riesgo personal. Detectar estas situaciones de peligro ha sido todo un reto, sin embargo, los investigadores de [8,35,36] proponen el uso de *OCC* como la mejor herramienta para detectar anomalías.

Después de analizar algunos de los contextos de aplicación de los *OCC* para la detección de anomalías, podemos deducir que es un tema de alta relevancia y que la propuesta de un algoritmo que mejore el estado del arte, tendrá un alto impacto en múltiples escenarios.

1.2. Definición del problema

La detección de anomalías mediante algoritmos *OCC* resulta ser una estrategia altamente versátil en múltiples escenarios de aplicación. La recurrencia de utilizar estas técnicas se debe a la facilidad de extrapolarlos a escenarios reales, donde se ven beneficiados los intereses de muchos usuarios y empresas. Esto explica el esfuerzo de los investigadores por diseñar algoritmos de *OCC* más rápidos y eficientes a la hora detectar una anomalía. Al analizar los trabajos en el marco de antecedentes podemos resumir algunas cuestiones de gran interés. Se vuelve todo un reto poder

balancear la rapidez y la eficiencia de un algoritmo, además, los diseños de *OCC* más efectivos como *OCSVM* requieren de ciertas habilidades técnicas para su comprensión, que resultan difícil de entender para las personas que se encargarán de administrar estas herramientas.
El interés de la Detección de anomalías en distintos contextos y su aplicación en escenarios reales ha motivado la presente investigación, la cual pretende resolver las problemáticas más recurrentes en este tipo de clasificadores. Debido a que los autores de [9–12] indican que el esquema de ensamble mostró una mejoría significativa en el desempeño, se optó que el diseño del algoritmo de clasificación propuesto en esta investigación sea con este enfoque. Para mitigar el tiempo de ejecución, se pretende motivarse en las técnicas con el mejor desempeño de *OCC* y reducir drásticamente su complejidad computacional, proponiendo nuevas estrategias que no afecten el desempeño del clasificador. Por tanto, en esta investigación se diseñará un algoritmo de *OCC* para la detección de anomalías basado en ensamble y será evaluado en dos contextos de aplicación.

1.3. Preguntas de investigación

1. ¿Es posible diseñar un algoritmo para la detección de anomalías basado en ensamble que sea competitivo respecto al estado del arte?
2. ¿Qué técnica de ensamble será la apropiada para el clasificador?
3. ¿Cuantos clasificadores serán los ideales para construir el ensamble?
4. ¿Es posible reducir la complejidad computacional sin disminuir el desempeño del algoritmo?
5. ¿El algoritmo será eficiente en conjuntos de datos grandes?
6. ¿El algoritmo puede diseñarse en una metodología distribuida?
7. ¿El nuevo algoritmo será competitivo en múltiples escenarios de aplicación?

1.4. Justificación

La detección de anomalías es un campo con mucho potencial, donde la comunidad científica ha dedicado años de investigación, debido a los múltiples escenarios en los que puede ser aplicada. En

los trabajos mencionados en la sección 1.1 podemos inferir que existen una serie de algoritmos para la detección de anomalías, sin embargo, se sopesa que los resultados obtenidos hasta el momento no son los deseables, no existe un equilibrio entre precisión y rapidez. Por lo tanto, el diseñar una estrategia que supere los algoritmos del estado del arte mediante la metodología de *ensambles* sería una aportación muy significativa a la comunidad científica y a futuras aplicaciones.

1.5. Objetivos

1.5.1. Objetivo general

Diseñar un algoritmo de clasificación de una clase (*OCC*) basado en un esquema de *ensambles* para la detección de anomalías.

1.5.2. Objetivos particulares

- Reducir la complejidad computacional respecto al estado del arte.
- Aumentar el desempeño de predicción.
- Seleccionar dos contextos de aplicación para evaluar el algoritmo.
- Diseñar el algoritmo desde una perspectiva de computo distribuido.
- Comparar el modelo respecto al estado del arte en relación a métricas de evaluación establecidas.

1.6. Organización del documento

La presente investigación se estructuró en 6 capítulos, a continuación, se especifican los detalles del contenido de cada uno de ellos.

- **Capítulo 2. Introducción al área de Detección de Anomalías (Anomaly Detection).**
  Inicia con una introducción al área de detección de anomalías, después, un análisis de los métodos tradicionales y cómo se ha abordado esta problemática y por último, se abordan

dos escenarios: Detección de Intrusos en sesiones de computadoras y Detección de riesgos personales mediante el procesamiento de señales de bio-sensado.

- **Capítulo 3. Escenario 1 - Detección de Intrusos en sesiones de computadoras.**
  Se comienza con una revisión del estado del arte respecto a la Detección de Intrusos a sesiones computadoras, debido a esto, se presenta el algoritmo *Bagging-RandomMiner* con su respectivo análisis y diagramas de flujo, así como las regiones de decisión y resultados obtenidos sobre el conjunto de datos establecido.

- **Capítulo 4. Escenario 2 - Detección de riesgos personales mediante el procesamiento de señales de bio-sensado.**
  Al inicio, se realiza una revisión de la literatura para poner en contexto la Detección de riesgos personales mediante el procesamiento de señales de bio-sensado, después, se describe el algoritmo más sofisticado para esta tarea según la literatura (*OCKRA*). Consecuentemente, se explica la metodología de cómputo distribuido, específicamente *MapReduce* y *Apache Spark*. Al tener en claro el funcionamiento distribuido se implementan algunos algoritmos bajo esta perspectiva y finalmente se realiza la comparación de los resultados con la respectiva prueba estadística más pertinente.

- **Capítulo 5. Conclusiones y trabajo futuro**
  Se concluye el trabajo de investigación de forma crítica y analítica, así como futuras propuestas de trabajo y extensión de la investigación.

- **Capítulo 6. Publicaciones ligadas a la tesis**
  Los artículos publicados, gracias al trabajo de investigación realizado en esta tesis. El primero se denomina *Bagging-RandomMiner: a one-class classifier for file access-based masquerade detection* publicado en la revista *Machine Vision and Applications*. El segundo llevar por nombre *Bagging-RandomMiner - Un Algoritmo en MapReduce para Detección de Anomalías en Big Data* publicado en la *18th Conference of the Spanish Association for Artificial*

Capítulo 2

Introducción al área de Detección de Anomalías (Anomaly Detection)

Resumen: En este capítulo se da una introducción al área de Detección de anomalías, mostrando mediante un análisis de la literatura la importancia y relevancia que tiene esta área en distintos dominios de aplicación. También se definen los dos problemas seleccionados para evaluar el algoritmo propuesto en esta investigación.

2.1. ¿En qué consiste la detección de anomalías?

La detección de anomalías se refiere al problema de encontrar patrones en los datos que no se ajustan al comportamiento esperado, a menudo se denominan, valores atípicos, valores anormales, o contaminantes en diferentes dominios de aplicación [5]. Comúnmente, anomalías o valores atípicos son términos intercambiables al referirse a este hilo de investigación. Existe una vasta variedad de aplicaciones al detectar una anomalía tal como la detección de fraudes en tarjetas de crédito [7, 37], detección de intrusiones en sistemas cibernéticos [6, 19–24, 26–32, 34], la detección de fallas en sistemas críticos de seguridad [14] y la vigilancia militar para actividades enemigas.

La detección de anomalías o valores atípicos es una linea de investigación que se ha estudiado desde el siglo XIX [38] y el primer clasificador de este tipo fue creado por Minter [39], con el paso del tiempo debido a su amplio dominio de aplicación, han surgido escenarios que nos permiten

crear modelos para la identificación de esta información discordante.

Los problemas de detección de anomalías se abordan con algoritmos de clasificación de una sola clase (*OCC*), estos son muy distintos a los relacionados con la clasificación binaria/multiclase convencional. En los *OCC* la clase positiva no se encuentra presente y el aprendizaje se obtiene de la clase negativa. La idea de clasificar los eventos positivos en ausencia de los negativos ha ganado una atención creciente en los últimos años. La idea de los algoritmos en *OCC* es hacer una descripción del conjunto de objetos de prueba y detectar cuáles de ellos se parecen al conjunto de entrenamiento. La diferencia con la clasificación convencional es que solo las instancias de una clase están disponibles. Los objetos de esta clase suelen llamarse objetos de entrenamiento, todos los demás son considerados datos atípicos. En la figura 2.1 se muestra un ejemplo en 2 dimensiones. El conjunto de datos tiene dos regiones normales, N_1 y N_2. Los puntos $o1$, $o2$ y $o3$ están muy alejados de las regiones normales y son considerados datos atípicos.

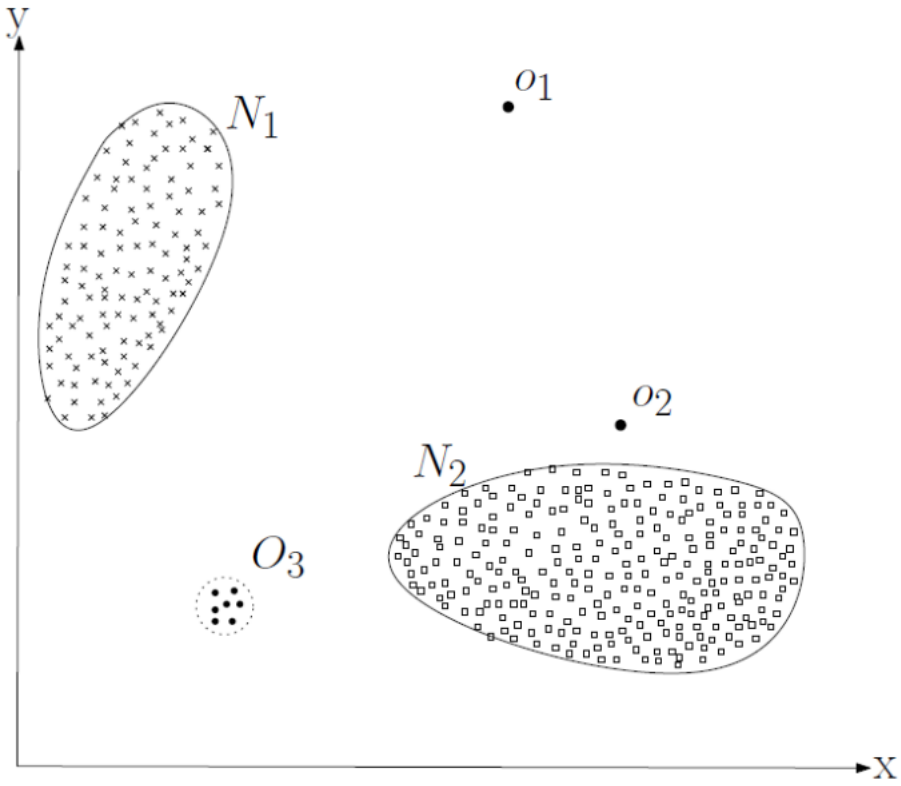


Figura 2.1: Ejemplo de anomalías en una distribución de datos en dos dimensiones

2.2. Métodos tradicionales - ¿Cómo se ha abordado este problema?

Debido a que el objetivo principal de esta tesis es proponer un clasificador para la detección de anomalías, es necesario hacer un análisis del estado del arte para identificar las ventajas y desventajas de los distintos algoritmos propuestos hasta el momento. A continuación, se definen los modelos más destacados según la literatura, utilizando la taxonomía propuesta en [40], la cual divide la clasificación de una sola clase en tres enfoques principales, los métodos de densidad, los métodos de límites y los métodos de reconstrucción.

2.2.1. Métodos de Densidad (Density Methods)

Los clasificadores de una clase basados en la densidad utilizan funciones que calculan la probabilidad de que una variable continua esté entre un rango de valores. Conocidas como funciones de densidad de probabilidad, estas funciones se utilizan para determinar la probabilidad de que un objeto x pertenezca a la clase normal. Si el valor obtenido está por encima de un umbral, el objeto se clasifica como normal y, por lado contrario, anormal. A continuación se muestran los algoritmos *OCC* más representativos de esta categoría:

Gaussian Density

El modelo más simple de *OCC* es *Gaussian Density* propuesto por *Bishop* en [41], basado en la distribución Gaussiana o normal, siendo las colas las regiones de rechazo y el volumen la región de aceptación, esto se ilustra en la figura 2.2.

Según el teorema del límite central [42], cuando se supone que los objetos de una clase se originan en un prototipo y se ven afectados por un gran número de pequeñas perturbaciones independientes, entonces este modelo es correcto. El clasificador de *Gaussian Density* calcula la probabilidad de que un objeto pertenezca a la clase normal mediante la ecuación 2.1.

$$P_N(\mathbf{z};\ \boldsymbol{\mu}, \Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{z}-\boldsymbol{\mu})\right\} \tag{2.1}$$

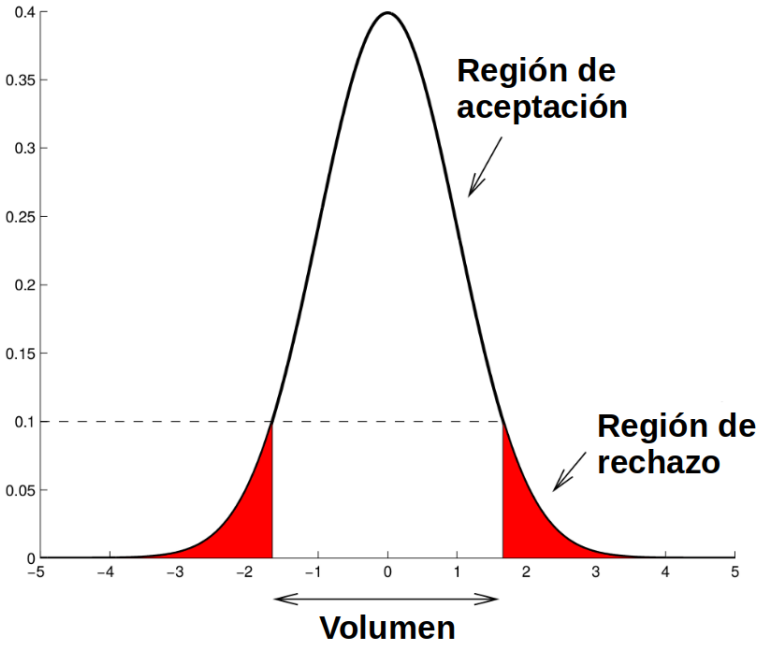


Figura 2.2: Límites de la distribución Gaussiana en 1-dimensión.

donde $\boldsymbol{\mu}$ es la media aritmética y Σ la matriz de covarianza. Este clasificador tiene una complejidad de entrenamiento de $O(n)$.

Parzen-window density estimation

Este método se define como una extensión de *Gaussian Density* y fue propuesto por Parzen en [43]. La intención de este clasificador, propicia la colaboración de todos los objetos de entrenamiento en T al construir la estimación de densidad, esta se construye en base a su distancia a cada objeto x. Para valorar la aportación de cada elemento se utiliza una ventana de tamaño h, donde cuanto más lejos esté un elemento del centro de la ventana menos contribuye. La fórmula para calcular la estimación de densidad de un nuevo objeto x se muestra en la ecuación 2.2

$$Pr(x,\ T) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{h}PDF\left\{\frac{x - x_i}{h}\right\} \tag{2.2}$$

donde PDF (por sus siglas en ingles, *Probability Density Function*) es la función de probabilidad a utilizar.

Parzen-window density es un clasificador con una varianza muy baja debido a la aportación

de cada objeto. Esto implica, que pequeñas variaciones en el conjunto de entrenamiento no repercute significativamente al modelo. El valor de h es la anchura en ambos sentidos, asumiendo una igualdad al ponderar las características y esto muestra la importancia del escalamiento de los datos. El costo computacional en la fase de entrenamiento puede considerarse lineal, sin embargo, para clasificar un nuevo objeto es necesario calcular la distancia a todos los datos de entrenamiento, aumentando significativamente el tiempo de ejecución. Este aspecto limita severamente el clasificador al tratar conjuntos de datos grandes y de altas dimensionalidades.

Naive Bayes

El algoritmo de *Naive Bayes* es una metodología muy simple que usa el teorema de Bayes para la clasificación. Aunque no es el único método de aprendizaje automático que utiliza métodos bayesianos, es el más común, en particular para la clasificación de texto, donde se ha convertido en el estándar de facto [1]. Existen algunas versiones del algoritmo, sin embargo, mencionaremos la propuesta en [44] por su aplicación en la detección de anomalías y clasificación de texto. Los modelos bayesianos son bien conocidos por su inherente capacidad de trabajar con el ruido y además lo realiza en un tiempo lineal $O(n)$, siendo n el número de ejemplos. El algoritmo define algunas características como:

- C**:** el número de veces que ha aparecido un objeto en el conjunto de entrenamiento.
- α**:** es un factor de regularización.
- N**:** El número de objetos totales en el conjunto de entrenamiento.
- A**:** El número de objetos distintos en el conjunto de entrenamiento.

Para determinar la probabilidad de ocurrencia de un objeto se define la ecuación 2.3.

$$Pr(x) = \frac{C + \alpha}{N + \alpha \cdot A} \tag{2.3}$$

El valor obtenido por la ecuación 2.3 es la probabilidad de pertenecer al comportamiento genuino o clase mayoritaria.

2.2.2. Métodos de Frontera (Boundary Methods)

Los clasificadores de una clase (*OCC*) descritos en esta sección tienen como objetivo construir una frontera de decisión que delimita el comportamiento normal o la clase mayoritaria. Lo que simplifica la clasificación de un nuevo objeto, siendo considerados como comportamiento normal aquellos que se encuentren dentro de esta frontera y anormales en caso contrario.

One-Class Support Vector Machine

Support Vector Machine (*SVM*) es un algoritmo propuesto originalmente para la clasificación de dos o más clases, sin embargo, es posible utilizar *SVM* para un tipo restringido de aprendizaje, es decir, para estimar regiones de densidad de datos que nos permiten construir un límite para separar regiones de alta y baja densidad. Dicho límite puede ser utilizado para la detección de anomalías [45]. En la formulación de dos clases, la idea básica es mapear vectores de características a un espacio dimensional alto y calcular un hiper-plano que no solo separa los vectores de entrenamiento de diferentes clases, sino que también maximiza esta separación al hacer el margen lo más grande posible [22].

Los autores de [46] propusieron un método para adaptar *SVM* a la detección de anomalías denominado *OCSVM* (por sus siglas en inglés, *One-Class Support Vector Machine*) utilizando ejemplos de una sola clase para el entrenamiento. El algoritmo *OCSVM* asigna los datos de entrada a un espacio de características de alta dimensión, transformados mediante una función de *kernel* que trata el origen como la única instancia de la otra clase. El objetivo es, buscar iterativamente el hiper-plano de margen máximo que mejor separe los datos de entrenamiento del origen.

Considerando que nuestro conjunto de entrenamiento sea $x_1, x_2, x_3 \in X$, Φ es la función de mapeo $X \to F$ a un espacio de alta dimensión, podemos definir la función de *kernel* como:

$$k(x,\ y) = (\Phi(x) \cdot \Phi(y)) \tag{2.4}$$

Al usar las funciones del *kernel*, los vectores de características no necesitan ser computados explícitamente, esto permite mejorar considerablemente la eficiencia computacional ya que los valores del *kernel* pueden ser calculados directamente. Algunas funciones *kernel* utilizadas son las siguientes:

$$\begin{aligned} \text{Kernel lineal:} \quad & k(x,\ y) = (x \cdot y) \\ \text{Kernel polinomial de grado p-th:} \quad & k(x,\ y) = (x \cdot y + 1)^p \\ \text{Kernel rbf:} \quad & k(x,\ y) = e^{\frac{-||x-y||^2}{2\sigma^2}} \end{aligned}$$

Ahora, resolver el problema *OCSVM* es equivalente al de la programación cuadrática dual (*QP*):

$$\begin{gathered} \min_{\alpha} \frac{1}{2} \sum_{ij} \alpha_i \alpha_j (x_i,\ x_j) \\ \text{Sujeto a } 0 \leq \alpha_i \leq \frac{1}{v^l}, \sum_i \alpha_i = 1 \end{gathered}$$

donde α_i es un multiplicador de *Lagrange*, que se puede considerar como un peso en el ejemplo x_i, y v es un parámetro que controla la compensación entre maximizar el número de puntos de datos contenidos por el hiper-plano y la distancia del hiper-plano desde el origen.
Después de resolver para α_i, podemos usar una función de decisión para clasificar los datos. La función de decisión se define como:

$$f(x) = sgn\left(\sum_i \alpha_i k(x_i,\ x) - p\right) \tag{2.5}$$

donde el el valor de p puede ser recuperado por

$$p = \sum_j \alpha_j k(x_j,\ x_i) \tag{2.6}$$

El algoritmo de *OCSVM* tiene la ventaja de proyectar cualquier conjunto de datos en múltiples dimensiones usando el *kernel* correcto, sin embargo, posee la gran desventaja de pertenecer a la rama de los $O(n^3)$ lo que hace muy poco efectivo utilizarlo en conjuntos de datos grandes.

2.2.3. Métodos de Reconstrucción (Reconstruction Methods)

Los clasificadores pertenecientes a esta clase, se basan en métodos especializados en describir el conjunto de datos a través de prototipos. Esta característica permite discernir si un elemento a clasificar pertenece a una clase dada, a través de la cercanía entre los objetos.

k-Nearest Neighbors

El algoritmo de *k-Nearest Neighbors (kNN)* es muy popular en el área de aprendizaje máquina, debido a la simplicidad y efectividad del método. Algunas fortalezas de *kNN* son que no hace suposiciones sobre la distribución de los datos y además carece de la fase de entrenamiento. Por otro lado, no produce un modelo y esto limita la capacidad de encontrar nuevas perspectivas en las relaciones entre las características. La fase de clasificación es muy lenta y si el conjunto de datos aumenta el requerimiento de memoria puede convertirse en un problema [1].

El funcionamiento de *kNN* se lleva a cabo en la fase de clasificación, esta etapa hace una clara justificación a la definición de vecinos cercanos, siendo el concepto de vecindad lo que predomina en el algoritmo. Al inicio, se identifica el conjunto de entrenamiento con sus respectivas etiquetas como se observa en la figura 2.3, aquí se muestra la representación en 2-dimensiones de un conjunto de datos con 3 clases; vegetales, frutas y proteínas.

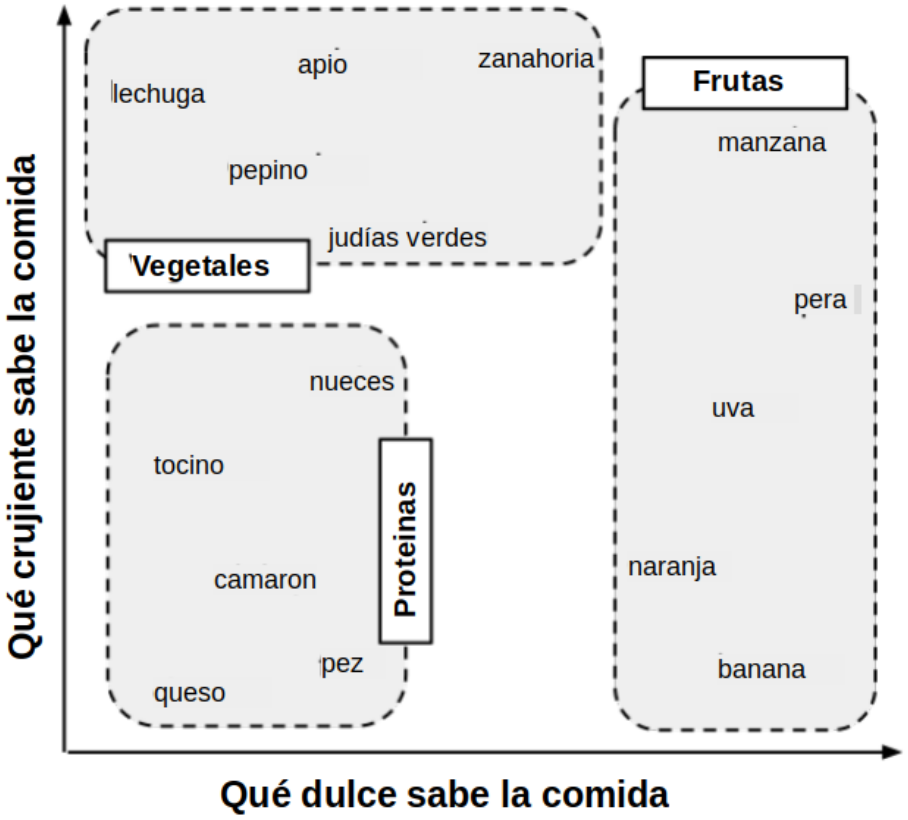


Figura 2.3: Ejemplo en 2-dimensiones de un conjunto de datos con 3 clases tomada de la referencia [1].

Supongamos que tenemos un conjunto de datos de prueba que contiene ejemplos no etiquetados y tienen las mismas características que los datos de entrenamiento. Para cada registro en el conjunto de datos de prueba, *kNN* identifica k (número entero definido anteriormente) registros en los datos de entrenamiento más cercanos al objeto de prueba, regularmente la cercanía se define con la función de distancia euclídea, sin embargo, es una pregunta abierta a la experimentación. Al objeto de prueba se le asigna la clase de la mayoría de los k vecinos más cercanos, esto se puede observar en la figura 2.4. En este ejemplo se analiza la instancia tomate con un número de vecinos $k = 4$, donde 1 es de la clase vegetales, 2 de la clase frutas y 1 de la clase proteínas, esto implica que el tomate será etiquetado como fruta.

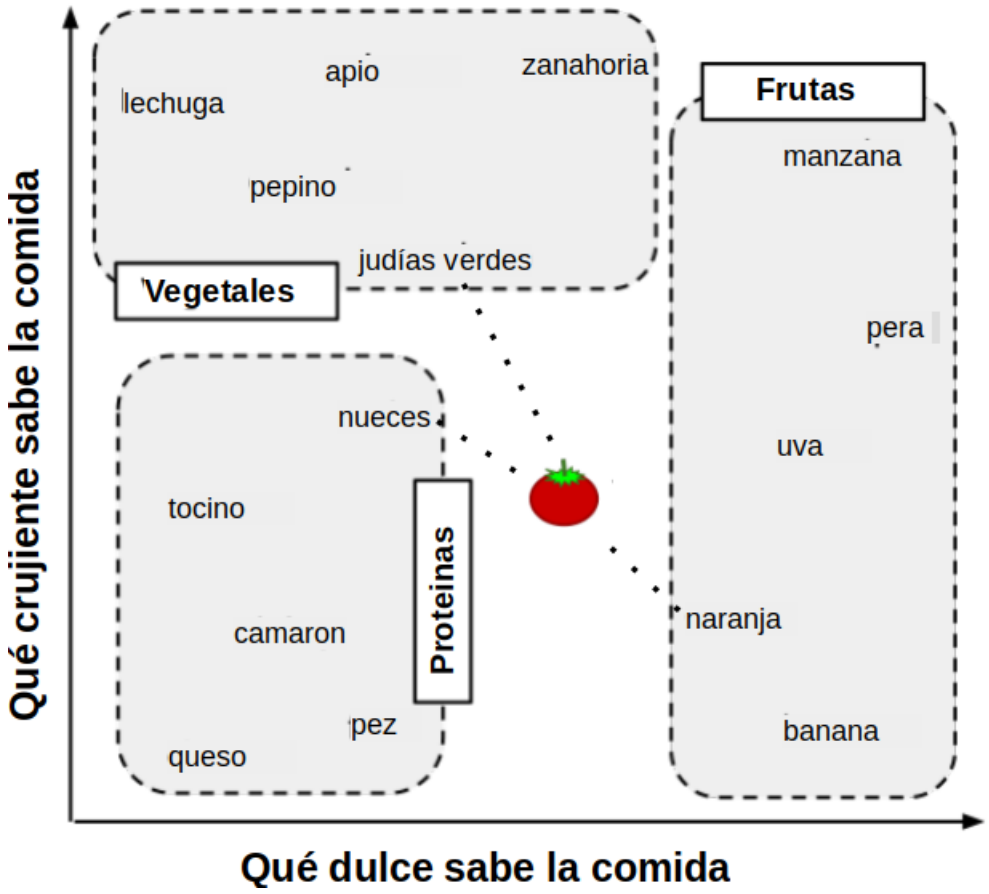


Figura 2.4: Ejemplo de clasificación del objeto tomate con un valor de $k = 4$ tomada de la referencia [1].

Existen varias versiones de *kNN* para la clasificación de una clase [47–49], sin embargo, la

versión propuesta en [49] ha mostrado excelentes resultados al detectar anomalías según *Goldstein y Uchida* en [50]. *One-Class k-Nearest Neighbors (OCkNN)* se puede simplificar en el algoritmo 2.1

Algoritmo 2.1 One-Class k-Nearest Neighbors

Entrada: D_{tr}: conjunto de entrenamiento; D_{te}: conjunto de prueba; k: número de vecinos, th: umbral de decisión.

Salida: C_{te}: conjunto de datos de prueba etiquetados.

1: **para** $I \in D_{te}$ **hacer**
2: $N \leftarrow$ La distancia entre I y cada objeto en D_{tr}
3: $D_1 \leftarrow$ La distancia más cercana de I en el conjunto N
4: $D_2 \leftarrow$ El promedio de las distancias de I a los k objetos más cercanos en N
5: **si** $\frac{D_1}{D_2} > th$ **entonces**
6: C_{te}.añadirClase(Desconocida)
7: **si no**
8: C_{te}.añadirClase(ClaseObjetivo)
9: **fin si**
10: **fin para**
11: **devolver** C_{te}

Se puedo observar que el algoritmo 2.1 sigue el comportamiento habitual de *kNN*, las únicas diferencias radican en las siguientes cuestiones. El conjunto de entrenamiento está constituido específicamente por la clase negativa, esto implica que al comparar una instancia de prueba con los k vecinos más cercanos nos dará la información de la cercanía al conjunto de datos, es decir la distancia promedio (D_2) a la que se encuentra con los k vecinos (paso 4). Para etiquetarlo se calcula la relación entre D_1 y D_2 y se compara respecto al umbral th (media armónica de las distancias del conjunto D_{tr}), si resulta ser inferior se etiqueta como la clase objetivo, en caso contrario se clasifica como una clase desconocida.

k-Means

El clasificador de *k-Means* se basa en el muy reconocido algoritmo de agrupamiento no supervisado *k-Means* [51]. La naturaleza del método radica en la obtención de k centroides definidos por el método. Este algoritmo de agrupamiento puede simplificarse en tres simples pasos, comúnmente se inicializan los k centroides de manera aleatoria, aunque existen otras versiones para hacerlo [52]. Después de la inicialización, estos k centroides se convierten en los centros de cada grupo, esto nos permite asignarle un grupo a cada uno de los elementos del conjunto de entrenamiento a través de la cercanía en función de la distancia. En el segundo paso, se recalculan los nuevos centroides aplicando la media aritmética entre los miembros de cada grupo. Por último, en el paso 3 se recalculan las distancias entre los objetos de entrenamiento y los nuevos k centroides para verificar si es necesario reasignarlos a un nuevo grupo. El proceso es iterativo hasta que los elementos del conjunto de entrenamiento convergen a un solo grupo. Al finalizar el proceso los últimos k centroides serán los objetos representativos de cada grupo.

El algoritmo de *k-Means* puede ser utilizado para la clasificación de una clase, aunque existen distintos modelos de su aplicabilidad [47, 53, 54] nos centraremos en el propuesto por [47]. Esta metodología aplica el algoritmo de agrupamiento *k-Means* para determinar k centroides, después, se calcula la distancia del objeto de prueba $\mathbf{z}$ a cada uno de los centros k de la distribución ω_t y se considera únicamente la distancia mínima como se observa en la ecuación 2.7

$$d(\mathbf{z},\ \omega_t) = \underset{i=1...k}{min} ||\mathbf{z} - \mu_i|| \tag{2.7}$$

donde μ_i representa el ith centro de cluster.

Si la distancia mínima obtenida es menor que un umbral θ el objeto de prueba se etiqueta como la clase mayoritaria, en caso contrario se rechaza. Por último, la complejidad de *k-Means* está determinada por $O(i \cdot k \cdot n)$, donde i es el número de iteraciones, k el número de centroides y n el número de ejemplos.

k-Centers

El algoritmo de *K-centers* fue propuesto por *Ypma* y *Duin* en [55] inspirados en el trabajo de números de cobertura de *Kolmogorov* [56]. Ambos siguen la naturaleza de un modelo sin conoci-

miento a priori de un conjunto de datos (aprendizaje no supervisado). El objetivo de este clasificador es encontrar una colección de k centros que colectivamente encierren todos los puntos del conjunto de entrenamiento, de modo que el radio del circulo más grande cubra toda la superficie posible. Para lograr esto, se debe minimizar el error de la ecuación 2.8.

$$\mathcal{E}_{k-centr} = \underset{i}{max}\left(\underset{k}{min}||x_i - \mu_k||^2\right) \tag{2.8}$$

La inicialización del método comienza con la asignación aleatoria del primer k centro. Por medio de la distancia euclidiana se asigna cada instancia de prueba a su centro más cercano, donde el radio de cada grupo será la distancia entre el centro y el objeto más alejado de su grupo. La elección de los k centros se realiza uno a la vez, en cada iteración, se determina el punto más alejado de los centros k para ser el siguiente k centro como se muestra en la figura 2.5

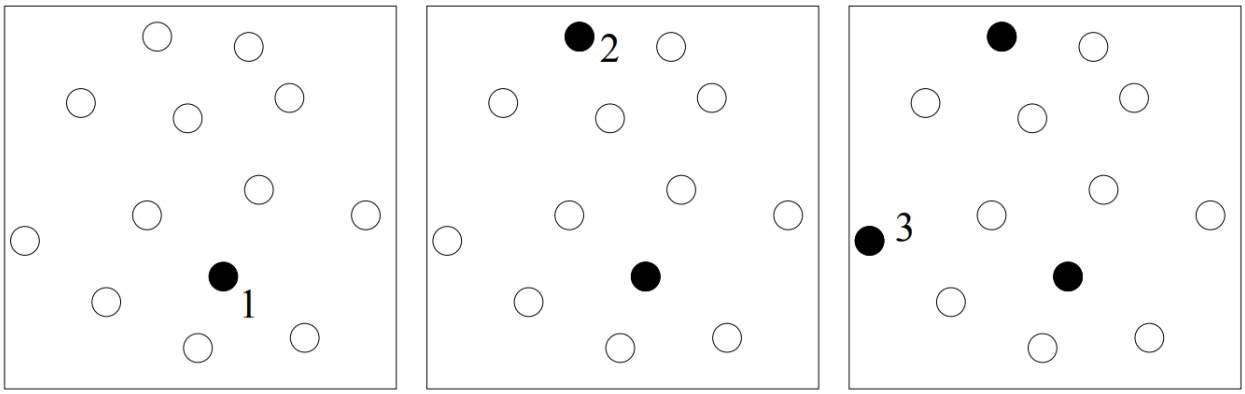


Figura 2.5: Las primeras 5 iteraciones del algoritmo *k-Centers*.

La clasificación de un objeto es análoga al algoritmo de *k-Means* para una clase. La complejidad computacional del algoritmo *k-Centers* es $O(nk)$, donde n es el número de elementos en el conjunto de entrenamiento y k el número de centros.

Para tener una perspectiva más clara de los algoritmos de clasificación de una clase se construyó la tabla 2.1 que resume los clasificadores por el método al cual pertenecen.

Tabla 2.1: Resumen de clasificadores de una clase más populares, definiendo su tipo de método y atributo.

Método	**Clasificador**	**Tipo de atributo**
Densidad	Gaussian Density	Numérico
	Parzen-window density estimation	Numérico
	Naive Bayes	Secuencia de símbolos
Frontera	One-Class Support Vector Machine	Numérico
Reconstrucción	k-Nearest Neighbors	Numérico
	k-Means	Numérico
	k-Centers	Numérico

2.3. Dos escenarios para la detección de anomalías

El aporte primordial de este trabajo de investigación es proponer un algoritmo de clasificación de una clase que sea competitivo respecto al estado del arte y que sea capaz de resolver dos problemas de la vida real. Para cumplir con esto, es necesario e indispensable seleccionar los escenarios de aplicación sobre los cuales será evaluada la aportación de esta tesis. Después de un análisis de los posibles candidatos, se decidió por la detección de intrusos en sesiones de computadora en el conjunto de datos *WUIL* [6] y la detección de riesgos personales mediante el procesamiento de señales de bio-sensado, específicamente en la base de datos *PRIDE* [8]. A continuación, se explicará detalladamente ambos conjuntos.

2.3.1. Detección de intrusos en sesiones de computadora

El conjunto de datos *WUIL* (por sus siglas en inglés, *Windows-Users and Intruder simulations Logs*) fue propuesto por Camiña et al. [6]. Este conjunto de datos fue creado con la hipótesis de que, la forma en que un usuario navega por la estructura de su sistema de archivos puede separar claramente el ataque de un intruso. Este enfoque, basado en la navegación del sistema de archivos, proporciona un medio más rico, y un nivel más alto de abstracción, para construir modelos nove-

dosos para la detección de intrusos. En su primera versión, *WUIL* incluyó 20 usuarios [6], pero a lo largo del tiempo se enriqueció al incluir 50 usuarios más, logrando la cifra de 70 [33]. Los registros de *WUIL* se recopilaron utilizando la herramienta de Windows denominada *Audit*. Cada entrada en los registros contiene información necesaria para interpretar la navegación de los usuarios en el sistema de archivos de su computador, los parámetros a considerar son: la hora, la fecha de acceso y la ruta del archivo al que se accede. Cada usuario proporcionó registros de distintas longitudes de tiempo, algunos de ellos solamente un día, mientras otros más excedieron los 60 días. Estas mediciones corresponden al comportamiento genuino de cada usuario, en la tabla 2.2 se muestran los perfiles de los primeros 20 usuarios.

Para simular la intrusión de un agente malicioso en una sesión de computadora, se construyeron tres escenarios que definen un ataque en distintos niveles, básico, intermedio y avanzado.

Ataque básico

El ataque básico modela un intruso ocasional, es decir, alguien que tiene una oportunidad de robar información sin ninguna planeación. Las acciones que esta persona realizará son muy simples y se puede asumir que no cuenta con algún dispositivo de almacenamiento (unidad flash USB). Por tanto, las acciones del intruso respecto al sistema de navegación de archivos se limitan a cerrar o abrir un documento, siendo el correo electrónico el único medio para extraer información. Los autores [6] definieron una metodología para la captura de un ataque básico que todos los intrusos tuvieron que respetar.

1. Anotar la hora exacta del día; programar una alarma de cinco minutos y detener el proceso inmediatamente después de que suene la alarma.

2. Ir a la carpeta ”Mis documentos” y configurarlo para que sea la carpeta de trabajo actual.

3. Examinar el contenido de la carpeta de trabajo actual, en la búsqueda de un archivo con algún nombre significativo que haga referencia a datos personales como cuentas bancarias. Si es posible se envía el archivo por correo electrónico o se toma una fotografía con un teléfono móvil. Repetir este paso con cada archivo de interés.

4. Aplicar este proceso de forma recursiva, profundizando en la estructura de árbol, es decir, indagar en cada carpeta que parezca atractiva.

5. Si es posible, repetir todo el procedimiento con el escritorio, o cualquier otra carpeta de interés.

Tabla 2.2: Perfil de los primeros 20 usuarios del conjunto de datos *WUIL*

Usuario	**Ocupación**	**Versión de MS Windows**	**No. de días registrados**
1	Gerente	Windows XP	49
2	Gerente	Windows XP	48
3	Área de Sistemas	Windows XP	35
4	Contador	Windows XP	29
5	Secretaria	Windows XP	54
6	Secretaria	Windows XP	55
7	Secretaria	Windows XP	56
8	Secretaria	Windows XP	31
9	Secretaria	Windows XP	34
10	Área de compras	Windows Vista	33
11	Secretaria	Windows XP	13
12	Programador	Windows 7	44
13	Estudiante	Windows Vista	40
14	Ventas	Windows 7	34
15	Estudiante doctorado	Windows Vista	54
16	Estudiante doctorado	Windows 7	37
17	Estudiante	Windows 7	35
18	Estudiante	Windows 7	30
19	Secretaria	Windows Vista	36
20	Estudiante doctorado	Windows XP	23

Ataque intermedio

En el ataque intermedio, se asume que el intruso planificó de forma premeditada su robo de información, por lo cual, lleva consigo una memoria USB. El proceso de intrusión igualmente se definió por los autores de [6] de la siguiente manera:

1. Anotar la hora exacta del día; programar una alarma de cinco minutos y detener el proceso inmediatamente después de que suene la alarma.
2. Se inserta la memoria USB.
3. Abrir la herramienta de búsqueda de Windows MS.
4. Buscar todos los archivos que coincidan con las siguientes cadenas: * bank *. *, * Password *. *, * Pass *. *, * Account *. *, * Number *. *, Y * pin * . *. Copiar los archivos en la memoria USB.

Ataque avanzado

El último caso modela un intruso con conocimientos avanzados para la extracción de archivos, es evidente que se trata de un ataque planeado, donde, el atacante lleva consigo una unidad flash USB y un archivo .bat que copiará automáticamente los archivos propuestos en el ataque intermedio. Los pasos para el proceso son los siguientes:

1. Anotar la hora exacta del día; programar una alarma de cinco minutos y detener el proceso inmediatamente después de que suene la alarma.
2. Se inserta la memoria USB.
3. Se hace doble clic en el archivo .bat
4. Retirar la unidad USB y eliminar todas las pistas.

Los tres ataques mencionados anteriormente se llevaron a cabo por una sola persona y en la máquina de cada usuario. Los archivos extraídos se destruyeron en presencia de cada usuario, además, las acciones se suspendieron inmediatamente al sonar la alarma programada.

Extracción de características

La herramienta de Windows *Audit* genera registros de información de cada usuario, por tanto, es necesario extraer las características más representativas que permitan distinguir entre un usuario y un intruso basado en la navegación de archivos. Para lograr interpretar los datos en el tiempo, se crean ventanas de 30 segundos para la extracción de características. A continuación en la tabla 2.3 se especifica cada una de ellas.

Finalmente, al realizar la extracción de características de cada usuario se hizo un conteo del número de instancias totales, normales y anómalas de cada uno de ellos, en la tabla 2.4 se muestra a detalle cada una de estas mediciones. En resumen, el conjunto de datos *WUIL* se constituye por 70 usuarios con un total de 413,794 instancias, de las cuales, 411,951 ($99.55\,\%$) corresponde al comportamiento genuino del usuario y 1843 ($0.45\,\%$) a los tres ataques mencionados anteriormente. Es evidente que, *WUIL* es un conjunto altamente des-balanceado y es un escenario perfecto para un algoritmo de detección de anomalías y la clasificación de una clase (*OCC*).

Tabla 2.3: Descripción de la extracción de características de la base de datos *WUIL*

Característica		**Descripción**	**Número**
Acceso	T	Objetos tocados en una ventana	1
	N	Nuevos accesos	2
Profundidad	Media	Estadísticas sobre la profundidad a la que cada objeto es tocado en una nueva ventana en la estructura del directorio	3
	Mediana		4
	Moda		5
	Varianza		6
	Desviación estándar		7
	Rango de muestra		8
Tiempo entre accesos	Media	El tiempo transcurrido entre dos accesos de objetos consecutivos	9
	Mediana		10
	Moda		11
	Varianza		12
	Desviación estándar		13
	Rango de muestra		14
Características gráficas	$\|V\|$	El número total de vértices	15
	$\|E_d\|$	Número total de aristas dirigidas	16
	$\|E_u\|$	Número total de aristas no dirigidas	17
	$\frac{\|V\|}{\|E_d\|}$	Vértices entre aristas dirigidas	18
	$\frac{\|V\|}{\|E_u\|}$	Vértices entre aristas no dirigidas	19
Distancia de trayecto entre accesos	Media	La distancia existente de la trayectoria entre dos objetos asociados con accesos consecutivos	20
	Mediana		21
	Moda		22
	Varianza		23
	Desviación estándar		24
	Rango de muestra		25

Tabla 2.4: Conteo de instancias por usuario del conjunto de datos $WUIL$

Usuario	Instancias totales	Instancias normales	Anomalías	Usuario	Instancias totales	Instancias normales	Anomalías
1	5,130	5,104	26	40	6,574	6,555	19
2	3,459	3,436	23	41	2,056	2,031	25
3	2,468	2,447	21	42	4,063	4,034	29
4	3,576	3,549	27	43	9,418	9,390	28
5	3,605	3,575	30	44	4,225	4,206	19
6	6,060	6,034	26	45	5,193	5,165	28
7	5,216	5,187	29	46	2,879	2,850	29
8	3,074	3,044	30	47	4,825	4,798	27
9	4,953	4,923	30	48	7,407	7,392	15
11	7,905	7,881	24	50	4,939	4,911	28
12	5,168	5,142	26	51	7,985	7,959	26
14	4,034	4,005	29	52	5,463	5,443	20
15	11,379	11,350	29	53	9,880	9,850	30
16	10,236	10,208	28	54	2,390	2,362	28
17	6,240	6,210	30	55	6,895	6,867	28
18	2,581	2,554	27	56	4,523	4,500	23
19	4,576	4,557	19	57	5,241	5,212	29
20	2,918	2,888	30	58	5,385	5,359	26
21	6,474	6,446	28	59	8,263	8,234	29
22	4,545	4,521	24	60	2,708	2,680	28
24	4,424	4,397	27	61	6,720	6,695	25
25	2,880	2,850	30	62	3,643	3,614	29
26	11,486	11,459	27	63	5,255	5,230	25
27	9,741	9,718	23	64	6,084	6,058	26
28	7,790	7,760	30	65	4,687	4,659	28
29	6,130	6,106	24	66	4,750	4,723	27
30	14,354	14,324	30	67	3,788	3,768	20
31	2,037	2,013	24	68	5,091	5,061	30
32	5,093	5,068	25	69	8,044	8,018	26
33	3,487	3,460	27	71	9,187	9,159	28
34	5,627	5,599	28	72	5,905	5,876	29
35	7,227	7,199	28	73	19,738	19,721	17
36	4,420	4,392	28	74	7,938	7,909	29
37	9,993	9,967	26	75	2,890	2,860	30
39	6,209	6,190	19	76	5,267	5,239	28

2.3.2. Detección de riesgos personales mediante el procesamiento de señales de bio-sensado

La base de datos *PRIDE* (por sus siglas en inglés, *Personal RIsk DEtection*) fue desarrollada por Barrera-Animas et al. [8] con la intención de identificar cuando una persona se encuentra inmersa en una situación de peligro, por ejemplo, un accidente automovilístico, una crisis de salud o cualquier evento que propicie un daño a la integridad física de una persona. Este conjunto de datos se creó bajo la hipótesis de que una persona al estar en una situación de peligro produce desviaciones repentinas y significativas en los patrones fisiológicos al comportamiento normal. Para hacer posible la construcción del conjunto de datos *PRIDE* fue necesario utilizar una herramienta que sea capaz de medir características fisiológicas del comportamiento normal de una persona, por tanto, se utilizó un brazalete de la compañía *Microsof* (*Microsoft Band*) ilustrado en la figura 2.6.

Figura 2.6: Microsoft Band

La *Microsoft Band* está compuesta de varios sensores que permiten obtener un estatus del estado fisiológico del usuario, en la tabla 2.5 se muestran los sensores con una breve descripción.

Tabla 2.5: Descripción de los sensores de la *Microsoft Band*

Sensor	**Descripción**	**Frecuencia**
Acelerómetro	Provee la aceleración en los ejes X, Y y Z en unidades g (gravitacionales). $1\ g = 9.81\frac{m}{s^2}$	8 Hz
Giroscopio	Provee la velocidad angular en los ejes X, Y y Z en unidades de grados por segundo ($\circ/seg$)	8 Hz
Distancia	Provee la distancia total en centímetros y velocidad actual en centímetros por segundo (cm/seg)	1 Hz
Ritmo cardiaco	Provee el número de latidos por minuto	1 Hz
Podómetro	El número total de pasos que el usuario ha dado	1 Hz
Temperatura de la piel	Temperatura actual de la piel en grados celsius	33 mHz
UV	Proporciona la intensidad de exposición a la radiación ultravioleta actual (ninguna, baja, media, alta, muy alta)	16 mHz
Calorías	Proporciona el número total de calorías que el usuario ha quemado	1 Hz

La base de datos se construyó con la ayuda de 23 sujetos de prueba, estos fueron sometidos a dos contextos de medición, el comportamiento normal y el comportamiento anómalo.

Comportamiento normal

En el comportamiento normal los usuarios fueron censados con la *Microsoft Band* durante una semana las 24 horas del día, perdiendo pequeños intervalos de tiempo para recargar el dispositivo o realizar actividades de aseo personal en las cuales no era posible recolectar información. El objetivo fue obtener una medición de las actividades diarias de cada usuario que representarán el comportamiento típico, a continuación, en la tabla 2.6 se muestran las posibles ocupaciones en el transcurso del día.

Tabla 2.6: Actividades diarias del comportamiento normal de la base de datos *PRIDE*

Número	**Actividad**	**Número**	**Actividad**
1	Desayunar	8	Dormir
2	Comer	9	Andar en bicicleta
3	Cenar	10	Trotar
4	Tomar café	11	Correr
5	Conducir un automóvil	12	Reunión social
6	Mirar la televisión	13	Usar el transporte público
7	Ir al cine	14	Ir al gimnasio

Comportamiento anormal

Para construir los datos de las condiciones anómalas, se diseñó cuidadosamente una serie de escenarios que simulaban situaciones de peligro o estrés para los sujetos. A continuación, se muestra en la tabla 2.7 cada una de las actividades propuestas, así como su analogía con la vida real.

Tabla 2.7: Descripción de los escenarios para construir el comportamiento anómalo de *PRIDE*

Escenario	**Analogía con situaciones reales**
Correr 100 metros	El correr 100 metros a toda velocidad se puede equiparar con escenarios donde el usuario este huyendo de una situación de peligro
Subir y bajar escaleras	En este escenario se pretende simular la evacuación en un edificio a causa de algún desastre
Sesión de boxeo	Se pretende identificar cuando el usuario tiene una pelea con algún atacante
Simular caídas	Se diseñó con la intención de identificar un desmayo
Sostener la respiración	Se puede identificar con alguna enfermedad respiratoria como la disnea

Para asegurarse que en el conjunto de datos *PRIDE* haya diversidad, los 23 sujetos seleccionados son de ambos géneros, tienen edades entre 21 y 52 años, alturas entre $1.56m$ a $1.86m$, el peso varía de 42 a 101 kg, realizan ejercicio en un rango de 0-10 horas por semana y dedican entre 20-84 horas por semana al ocio.

Extracción de características

Para realizar la extracción de características se plantearon una serie de pasos, que permitieron construir el conjunto de datos *PRIDE* en el contexto de riesgos personales. A continuación, se muestran los pasos propuestos:

1. Para cada sujeto de prueba en el conjunto de datos, la información capturada se ordenó por día.

2. Se eliminaron aquellas características que no eran de utilidad para el experimento, como la fecha y actividad.

3. El vector de características se obtuvo al crear ventanas de un segundo. El cual específicamente contiene:

    a) Media y desviación estándar para medidas de giroscopio y acelerómetro.

    b) Valores absolutos de frecuencia cardíaca, temperatura de la piel, número de pasos, velocidad y sensores UV.

    c) El cambio incremental para valores absolutos de pasos totales, distancia total y calorías quemadas.

Finalmente, se obtiene un vector de características de 26 dimensiones y cuyos valores son actualizados cada segundo, estas características se pueden observar en le tabla 2.8. En resumen, el conjunto de datos *PRIDE* se constituye por 23 sujetos, generando un total de 7,432,715 instancias, de las cuales, 7,416,564 ($99.78\,\%$) corresponden al comportamiento normal de los usuarios y 16,151 ($0.22\,\%$) representan las situaciones de peligro simuladas en los 5 escenarios anteriormente mencionados, esto se puede analizar en la tabla 2.9. Análogo a la base de datos *WUIL* se puede observar que *PRIDE* es un conjunto de datos altamente des-balanceado y por lo cual un candidato perfecto para la detección de anomalías mediante la clasificación de una clase.

Tabla 2.8: Características del conjunto de datos *PRIDE*

Características			**Número**
Acelerómetro Giroscópico	**Eje X**	$\bar{x}$	1
		s	2
	Eje Y	$\bar{x}$	3
		s	4
	Eje Z	$\bar{x}$	5
		s	6
Velocidad Angular	**Eje X**	$\bar{x}$	7
		s	8
	Eje Y	$\bar{x}$	9
		s	10
	Eje Z	$\bar{x}$	11
		s	12
Acelerómetro	**Eje X**	$\bar{x}$	13
		s	14
	Eje Y	$\bar{x}$	15
		s	16
	Eje Z	$\bar{x}$	17
		s	18
Ritmo Cardíaco			19
Temperatura de la piel			20
Pasos			21
Velocidad			22
UV			23
Δ Podómetro			24
Δ Distancia			25
Δ Calorías			26

Tabla 2.9: Conteo de instancias por usuario del conjunto de datos *PRIDE*

Usuario	**Instancias totales**	**Instancias normales**	**Anomalías**
1	466,932	466,162	770
2	315,569	314,961	608
3	342,126	341,632	494
4	373,596	373,007	589
5	297,048	296,256	792
6	329,636	328,962	674
7	289,143	288,373	770
8	398,030	397,306	724
9	337,027	336,461	566
10	252,137	251,370	767
11	442,938	442,291	647
12	244,367	243,688	679
13	431,968	431,483	485
14	162,028	160,962	1066
15	303,731	302,850	881
16	328,369	327,791	578
17	134,723	133,782	941
18	336,315	335,411	904
19	401,662	400,893	769
20	301,034	300,448	586
21	360,013	359,590	423
22	374,354	373,770	584
23	209,969	209,115	854

Capítulo 3

Escenario 1: Detección de Intrusos en sesiones de computadora

Resumen: En este capítulo se aborda el primer escenario para la detección de anomalías, haciendo un análisis de la literatura respecto a la detección de intrusos en sesiones de computadora. También, se presenta el algoritmo propuesto en la tesis de investigación denominado *Bagging-RandomMiner* a través de pseudocódigos y diagramas de flujo, así como una comparación de la regiones de decisión con el algoritmo más novedoso en este primer escenario. Por último, se evalúa su desempeño en el conjunto de datos *WUIL*.

3.1. Revisión de literatura

La detección de intrusos en una sesión de computadora es un contexto amplio y que ha sido analizado desde distintas perspectivas. Existen varias formas para perfilar el comportamiento de un usuario. A continuación se muestran algunos de los trabajos más representativos.

3.1.1. Uso de comandos UNIX

Perfilar el comportamiento de un usuario frente a una computadora es tan importante como el algoritmo a utilizar para detectar un ataque. Por esta razón se han desarrollado distintas estrategias que buscan la mejor representación de un usuario. En esta vertiente se considera que el historial de

comandos *UNIX* es la mejor opción para capturar el comportamiento genuino del usuario.
Uno de los trabajos más representativos fue propuesto por Schonlau et al. [19] en el 2001. Su aportación fue la construcción del conjunto de datos *SEA*, que se convirtió en el repositorio más concurrido al comparar algoritmos *OCC*.
El conjunto de datos *SEA* se constituye por 70 usuarios. Las sesiones se realizaron en *UNIX* y cada una consta de 15,000 comandos, las secuencias se dividen en bloques de 100 comandos que representan una sesión. Para contaminar las sesiones con secuencias ilegítimas, fueron seleccionados 50 usuarios aleatoriamente de los 70 iniciales, estos reemplazaron algunas secuencias para simular una intrusión. El problema propuesto por los autores de [19] consistía en crear un mecanismo capaz de identificar las secuencias de comandos legitimas de las ilegitimas. A consecuencia de esto, algunos investigadores han aportado distintos clasificadores para darle solución a este conjunto de datos, así como nuevas perspectivas para abordar esta problemática.

Una estrategia que demostró tener buenos resultados se basa en el modelo de *Naive Bayes* en su versión original para problemas multi-clase. En los trabajos [20, 21] se implementó dicho algoritmo para evaluar la probabilidad de que una secuencia legítima halla sido escrita por un usuario genuino, después utilizando la misma metodología se comparó con una secuencia escrita por un intruso para identificar ambas clases. Además, se implementó una estrategia donde se considera el historial de los comandos comunes de cada usuario como perfil de comportamiento. Existen otros trabajos como [22], que demostraron que la implementación de *Naive Bayes* en su versión de una clase mencionada en la sección 2.2.1, donde solo se analiza el historial de comandos de un usuario genuino era suficiente para identificar una intrusión.

Existen otros trabajos como [18, 23] que implementaron un *Support Vector Machine (SVM)* para la detección de anomalías del conjunto de datos *SEA*. *Kim* y *Cha* investigaron la efectividad de *SVM* para detectar intrusiones en el conjunto de datos *SEA* y la versión utilizada por Maxion en [23] mediante un estudio empírico. Los experimentos realizados tuvieron éxito al superar el estado del arte en un $5.1\,\%$, esto pudo repercutir en la estrategia de los autores al incluir los nombres y los argumentos en los comandos comunes.
Por otro lado, Min Yang et al. [18] argumentaban la relevancia de proponer un algoritmo para la

detección de intrusos, debido a que el *Access Control* y el *Firewall* no tienen la capacidad de lograrlo. Estos autores propusieron un nuevo algoritmo denominado *SK-OCSVM*, siendo una versión de *SVM* pero con la diferencia de que evalúa las cadenas introducidas por los usuarios, obteniendo un aprendizaje de la frecuencia de aparición y el orden en el que son escritas ciertas cadenas. Los resultados obtenidos se analizaron sobre la base de datos de *SEA* mostrando que *SK-OCSVM* tiene un rango de detección más alto que algunos algoritmos del estado del arte.

Los autores de [25] definieron que distintas versiones de *SVM* han sido utilizadas para la detección de intrusos a sesiones de computadora. Éstas han obtenido las puntuaciones más altas para dicha tarea, sin embargo, existía la incertidumbre respecto a cuál modelo de *SVM* (One-class o Multi-Class) es el más indicado para realizar la clasificación. Para las experimentaciones tomaron el conjunto de datos propuesto por la universidad de *Purde* análogo a SEA, el cual considera las secuencias de comando de *UNIX*. Al finalizar, concluyeron que la mejor opción para atacar este problema es aplicar *SVM* con la versión de una sola clase mostrada en la sección 2.2.2.

En la investigación propuesta en [24] se indagó en un tema muy importante, estos autores mostraron la relevancia de implementar un método que sea fácil de interpretar por los administradores de seguridad, debido a que *SVM* es una propuesta efectiva en la detección de intrusos, sin embargo, su interpretación no es trivial para usuarios poco relacionados con el área de aprendizaje máquina. En consecuencia, los autores diseñaron un modelo basado en un lenguaje similar al de los expertos, este modelo se construye con un enfoque de reglas utilizando n-gramas para representar las secuencias de comandos del conjunto *SEA*. Con este enfoque se implementó el método de *boosting decision stumps* mejorando significativamente la clasificación en un $8\,\%$ aproximadamente.

Existen trabajos de investigación con enfoques distintos a lo convencional, por ejemplo, en [57] los autores utilizaron algoritmos genéticos para identificar un intruso. Utilizando la alineación de secuencias para comparar la similitud entre dos cadenas como se realiza con secuencias de proteínas de *ADN* o *ARN*. Estos autores tomaron el concepto de **mutación**, que consiste en la sustitución, eliminación o inserción de comandos en una secuencia, para medir la similitud entre un comando de un usuario comúnmente denominado firma y la secuencia a evaluar. Es evidente que, al ocu-

rrir menos mutaciones en una cadena la medida de similitud es muy alta y es más probable que pertenezca al usuario, en caso contrario la persistente existencia de mutaciones indica una cadena maliciosa creada por un intruso.

En lo que respecta al uso de comandos *UNIX* el conjunto de datos *SEA* es el repositorio más popular en la comunidad científica, sin embargo, la contribución ha tenido más peso en la clasificación que en la detección de un intrusos. La razón principal es que *SEA* está muy lejos de simular un escenario real, el simple hecho de necesitar la escritura de comandos es poco práctico para detectar un intruso.

3.1.2. Uso del ratón

Dentro del ámbito científico se han desarrollado otras estrategias para perfilar un usuario en una sesión de computadora, ciertamente los dispositivos I/O son muy populares e indispensables al utilizar alguna computadora, esto implica que los investigadores utilicen estas herramientas para lograr un perfilado más preciso. En esta sección se comentarán algunos trabajos relevantes en los cuales el uso del ratón fue el centro de aprendizaje debido al control que proporciona en una interfaz gráfica de usuario (*GUI*).

Los autores de [26] basaron su investigación en la construcción de un conjunto de datos capaz de recabar la información de los movimientos del ratón en una pantalla 2-D. Estos se resumen en: las coordenadas finales del puntero, la distancia recorrida desde la última posición del puntero, el ángulo y el tiempo. Aunque el conjunto de datos fue pionero en este enfoque tiene una gran limitante, se restringe al uso del Internet para recabar los datos y esto lo aleja de un escenario de intrusión real. A cada uno de los 18 usuarios que lo conforman se les pidió navegar por Internet en un determinado tiempo para capturar la información necesaria. Los autores propusieron un árbol de decisión (*C5.0*) para clasificar el conjunto de datos y evidentemente no es un clasificador de una clase.

Existe otro trabajo muy similar donde igualmente se creó un conjunto de datos basado en la información que proporciona el ratón, pero particularmente esta investigación realizada por Garg et al. [27] incluye los clics ejecutados por los usuarios, lamentablemente solo consta de 3 usuarios sesgando demasiado las experimentaciones. Para la evaluación de los datos estos autores utilizaron

el clasificador *SVM* en su versión multi-clase, donde es necesario utilizar el comportamiento ordinario del usuario como el de la comunidad restante.

Por último, hablaremos del trabajo de Shen et al. [28]. Estos autores realizaron un estudio experimental muy vasto para equiparar algoritmos de detección de anomalías con el enfoque del uso del ratón. Evaluaron un total de 17 algoritmos sobre un conjunto de datos que ellos mismos diseñaron, este se compone de 17,400 muestras de un total de 58 usuarios. Las parámetros a evaluar consisten en, la precisión de detección, la complejidad computacional en entrenamiento y prueba, así como la escalabilidad del algoritmo en los 58 usuarios. Al finalizar las experimentaciones se concluyó que los 6 clasificadores con el mejor rendimiento obtuvieron un error que oscila entre el 8.81 y 11.63 % y con un tiempo de ejecución de 6.1 segundos en promedio.

Es importante mencionar que los escenarios de intrusión de usuarios analizados hasta el momento, están restringidos por una sola aplicación y además no siguen el enfoque de una sola clase, esto obliga a la comunidad a proponer nuevos escenarios que se asemejen a la vida real y puedan ser implementados en un futuro cercano.

3.1.3. Uso del teclado

Dentro de los dispositivos de *HCI* (por sus siglas en inglés, *Human Computer Interaction*) el uso del teclado sigue ocupando los primeros lugares en popularidad, por esta razón, algunos investigadores decidieron aprovechar este aspecto y construir escenarios para la detección de intrusos. El perfilado de usuarios en el contexto del teclado se puede definir en estático (por ejemplo, escribir una secuencia en repetidas ocasiones como una contraseña) o texto libre. En referencia al tecleo estático se puede mencionar el trabajo de Killourhy y Maxion [29] que evaluaron varios algoritmos del estado del arte basados en la clasificación de una clase y en su mayoría pertenecientes a métodos de reconstrucción 2.2.3. El conjunto de datos empleado se constituye de 51 usuarios y cada uno aportó 8 sesiones. En cada sesión, los usuarios escribieron una contraseña fija 50 veces y en base a esto se construyó un vector de características para ser evaluado. Entonces, cada clasificador evalúa la forma en que los usuarios escriben su contraseña. En el 2015 los autores de [31] utilizaron este mismo conjunto de datos con la peculiaridad de identificar los usuarios con alta probabilidad de ser

protegidos contra un ataque, esto debido a la pulsación de las teclas.
En el contexto del tecleo libre se puede mencionar el trabajo de Messerman et al. [30], la mayor aportación de estos autores es un conjunto de datos constituido por 55 usuarios que trabajan en una aplicación de correo web. La información recolectada considera las pulsaciones de las teclas y los lapsos de tiempo. En este trabajo el algoritmo implementado posee un enfoque multi-clase.

Diseñar un perfil basado en las pulsaciones de teclas puede ser poco realista para detectar un intruso, por ejemplo, en el contexto de tecleo estático a la hora de definir una contraseña puede convertirse en un problema si el sitio web o la aplicación de escritorio tienen políticas de seguridad y exigen un cambio de contraseña en determinado tiempo. Además, si un intruso intenta acceder a un computador tendría que interactuar específicamente con el programa que requiere de la contraseña para poder ser detectado, haciendo evidente que el uso del teclado no es una buena opción para diseñar un escenario real.

3.1.4. Sistema de navegación de archivos (*FSN*)

Los *FSN* (por sus siglas en inglés *File System Navigation*) surgieron a partir de las limitaciones que presentaban los enfoques de secuencias de comandos *UNIX* 3.1.1, el uso del ratón 3.1.2 y el uso del teclado 3.1.3. Las problemáticas más representativas y recurrentes se pueden resumir en dos principales. Primero, el problema de detección de intrusos no está estructurado bajo la naturaleza de una clase, es decir, la simulación de los ataques no son muy equiparables con un escenario real y por lo tanto, es necesario aplicar un enfoque *OVA* (por sus siglas en inglés, *One Versus All*) que consiste en construir un clasificador para cada clase tomando las demás como muestras negativas y evidentemente esto resulta poco realista en un escenario real.
Segundo, la construcción de los conjuntos de datos se limita exclusivamente a una sola aplicación dentro del sistema.
Después de analizar las ventajas y desventajas de los enfoques anteriores surgió el *FSN* que basicamente perfila un usuario en base a la navegación de su sistema de archivos.
Benito Camiña et al. [6] propusieron un nuevo conjunto de datos denominado *WUIL* (por sus siglas en ingles, Windows Users and Intruder simulations Logs) con una perspectiva diferente a las bases de datos propuestas por el estado del arte para la detección de intrusos. Ellos indicaron que detectar un intruso basados en la secuencia de comandos de *UNIX* podría no ser la mejor opción,

considerando que un intruso no necesariamente utilizaría una secuencia de comandos para robar información del usuario. Esto los llevó a proponer *WUIL*, una base de datos centrada en el acceso a los archivos de la computadora, recolectando la información de las rutas y los nodos tocados por cada usuario. Está constituido por 20 voluntarios que fueron monitoreados en alguna distribución de *MS Windows* en un periodo de 5 a 10 semanas como el comportamiento genuino y fueron creados 3 escenarios de intrusión, denominados básico, intermedio y avanzado. Extendiendo en segundo trabajo [33] a 70 usuarios totales. Para la evaluación de *WUIL* implementaron dos versiones de *SVM* y *KNN* de una sola clase, con la intención de mostrar el poder de discriminación de ésta base de datos. Los resultados obtenidos son muy prometedores, pero la aportación de estos autores fue fundamental para la construcción de modelos de clasificación de una clase en futuros trabajos.

Miguel Medina et al. [34] propusieron un novedoso algoritmo de detección de anomalías denominado *Bagging-TPMiner* que se caracteriza por identificar todas las regiones de interés del comportamiento normal. Éste algoritmo se basa en la técnica de ensamble [58], donde la idea es definir un número de clasificadores y muestrear aleatoriamente con reemplazo una cierta porción de la población para cada uno de ellos y a su vez construir un modelo. Éste algoritmo está definido en el área de *clustering* y determina objetos típicos que representen a la población. Se diseñó con la intención de no tener que definir un número de *clusters* que suele ser un problema en algoritmos no supervisados como lo es *k-Means* [59]. Los experimentos se realizaron en la base de datos *WUIL* y los resultados mostraron que *Bagging-TPMiner* es superior significativamente a los obtenidos por Benito Camiña et al. [6, 33] que implementaron *SVM* y otros algoritmos del estado del arte.

Resumiendo los antecedentes en la detección de intrusos en sesiones de computadora, podemos concluir que el conjunto de datos *WUIL* es el más apto para simular un escenario real, en el cual puede identificarse un intruso significativamente. Por este motivo, la base datos *WUIL* es uno de los conjuntos seleccionados para evaluar el algoritmo de clasificación de una clase propuesto en este trabajo de investigación.

3.2. *Bagging-TPMiner*, un algoritmo del estado del arte para detección de anomalías en la base de datos *WUIL*

Debido a que *Bagging-TPMiner (BTPM)* es la metodología líder al evaluar el conjunto de datos *WUIL*, es necesario analizar a detalle su funcionamiento y detectar las ventajas y desventajas de este algoritmo. A continuación se muestra minuciosamente el entrenamiento y la evaluación de esta metodología.

Bagging-TPMiner es un algoritmo de clasificación de una clase basado en ensamble, se diseñó para la tarea de detección de intrusos en sesiones de computadora. La justificación de su diseño indica, que un usuario a menudo sigue los mismos patrones de comportamiento o simplemente efectúa pequeñas variaciones de ellos al navegar por el sistema de archivos de su computadora. Por lo cual, para entrenar un modelo, el clasificador encuentra patrones de similitud utilizando el agrupamiento, cada grupo contiene objetos muy similares entre sí y en particular a los centroides de agrupación definidos como objetos típicos [34]. Para clasificar si un nuevo objeto es genuino o una intrusión, el clasificador lo compara con todos los objetos típicos encontrados en la fase de entrenamiento y selecciona el más cercano para calcular un valor de tipicalidad, este indicará la probabilidad de pertenecer al comportamiento genuino del usuario.

3.2.1. Fase de Entrenamiento

En la fase de entrenamiento se crea un ensamble de múltiples clasificadores, cada clasificador consta de una tripleta: objetos típicos, umbral de decisión y la matriz de covarianza. Para definir los objetos típicos de cada clasificador, la literatura sugiere utilizar *k-Means* [59] para calcularlos, sin embargo, el utilizar esta metodología tiene dos problemáticas cuestionables. Primero, determinar el mejor valor de k es un desafío; segundo, la aleatoriedad del algoritmo produce objetos típicos en regiones más densas, por tanto, ciertos objetos quedan demasiado alejados de las regiones menos concurridas. Es por esta razón, que los objetos típicos determinados por *Bagging-TPMiner* se calculan con el algoritmo 3.1, asegurando que cada objeto típico está lo suficientemente cerca (respecto a un umbral) de al menos un objeto del comportamiento genuino del usuario.

- **Algoritmo 3.1 TPMiner: minería de objetos típicos**

  El algoritmo 3.1 es el encargado de computar los objetos típicos del conjunto de entrenamiento. Primeramente, se determina para cada objeto todos aquellos miembros del conjunto de entrenamiento que se encuentren más cercanos que cierto umbral (δ), después se calcula el valor de tipicidad para cada uno de ellos en base a la distancia que los separa (paso 2 y 3). Iterativamente, el algoritmo selecciona el objeto con mayor tipicidad (paso 8) y lo añade al conjunto de objetos típicos R, los objetos que fueron soportados por él son añadidos al conjunto de objetos soportados S. Este proceso se realiza hasta que todo el conjunto de entrenamiento sea soportado por algún objeto típico. Finalmente, el algoritmo 3.1 retorna los objetos típicos encontrados.

Algoritmo 3.1 TPMiner: minería de objetos típicos

Entrada: T: conjunto de entrenamiento; M: matriz de covarianza; δ: umbral de distancia.

Salida: R: objetos típicos calculados.

1: **para** $o \in T$ **hacer**
2: $C_o \leftarrow$ objetos más cercanos a o respecto a δ; i.e., $o' \in T$: $d(o, o', M) \leq \delta$
3: $t_o \leftarrow$ tipicalidad del objeto calculado con la ecuación $\sum_{o' \in C_o} e^{-0.5(d(o,o',M)/\delta)^2}$
4: **fin para**
5: $R \leftarrow$ () Objetos típicos computados
6: $S \leftarrow$ () Conjunto de objetos soportados
7: **mientras** Existan objetos no soportados por los objetos típicos en R **hacer**
8: $ot \leftarrow$ objeto más típico en la colección de no soportados; i.e., $arg\ \underset{o\ \in\ T-S}{max}(t_o)$
9: Añade ot a los objetos típicos R
10: Añade a S los objetos soportados por ot
11: **fin mientras**
12: **devolver** R

- **Algoritmo 3.2: Construcción del ensamble de Bagging-TPMiner**

Es bien conocido que combinar clasificadores es una técnica muy efectiva para aumentar el

porcentaje de precisión de un clasificador simple [58], es por esto que *Bagging-TPMiner* en el algoritmo 3.2 construye un ensamble de múltiples clasificadores para tomar una decisión sobre la etiqueta de un nuevo objeto. La idea del ensamble es sencilla, se eligen objetos con reemplazo (*bootstrap*) del conjunto de entrenamiento un número τ de veces y se construye un clasificador con cada muestra (paso 2 y 3). Después de hacer el *bootstrap* se calcula la matriz de covarianza M del conjunto T' (paso 4). En la paso 7 se determina el umbral de decisión δ con la distancia de *Mahalanobis*. Al obtener estos 3 elementos se invoca la función *TPMiner* (paso 6) que corresponde al algoritmo 3.1. Por ultimo, se almacena una tripleta: los objetos típicos P, la matriz de covarianza M y el umbral de decisión δ para cada clasificador.

Algoritmo 3.2 Construcción del ensamble de Bagging-TPMiner

Entrada: T: conjunto de entrenamiento; τ: número de clasificadores en el ensamble; υ: fracción del conjunto de entrenamiento a muestrear.

Salida: $Params$: parámetros calculados para este clasificador.

1: $Params \leftarrow ()$
2: **para** $i = 1...\tau$ **hacer**
3: $T' \leftarrow$ Bootstrap(T, υ)
4: $M_i \leftarrow$ ComputarMatrizCovarianza(T')
5: $\delta_i \leftarrow$ AvgDistancias(T', M_i)
6: $P_i \leftarrow$ TPMiner(T', M_i, δ_i)
7: $Params \leftarrow Params \bigcup \{(P_i, M_i, \delta_i)\}$
8: **fin para**
9: **devolver** $Params$

3.2.2. Fase de Clasificación

Para clasificar un nuevo objeto, *Bagging-TPMiner* en el algoritmo 3.3 retorna un valor de similitud que indica la probabilidad de no ser un intruso. Inicialmente en el paso 2 se itera sobre cada tripleta obtenida en el algoritmo 3.2 y se determina el grupo al que pertenece calculando el objeto típico más cercano (d_{min}). La función de $e^{-0.5(d_{min}/\delta_i)^2}$ transforma la distancia en una medida de similitud que corresponde al voto del clasificador. En la figura 3.1 se muestra el comportamiento de la función de similitud, se propone como ejemplo didáctico un umbral $\delta = 1.5$ y se define que las

distancias mínimas d_{min} se establecen en un intervalo de [0-4] con un incremento de 0.5 unidades, esto permite demostrar que para aquellos valores de distancia ($d_{min} < \delta$) se produce un alto valor de similitud (> 0.6), por otro lado, el valor de similitud sería muy bajo indicando que el objeto se encuentra muy alejado de todos los objetos típicos.

Bagging-TPMiner promedia las similitudes del nuevo objeto con respecto a todas las tripletas de parámetros (paso 6). Además, promedia las similitudes calculadas de los objetos anteriores (paso 7) que se combinan con la similitud del objeto que se esta clasificando (paso 12). Finalmente, se obtiene el valor probabilístico de no ser etiquetado como una intrusión.

Algoritmo 3.3 Fase de clasificación de Bagging-TPMiner

Entrada: o: objeto a clasificar; $Classifiers$: miembros de ensambles retornados por el algoritmo 3.2; Q: cola con resultados de clasificaciones pasadas; ν número de objetos pasados a considerar en la clasificación actual.

Salida: probabilidad de no ser intruso

1: $s \leftarrow 0$ s almacenará los votos para los clasificadores existentes
2: **para** $C \in Classifiers$ **hacer**
3: $d_{min} \leftarrow$ Distancia entre o y el objeto más cercano en C
4: Añade a s el voto $e^{-0.5(d_{min}/\delta_i)^2}$
5: **fin para**
6: Promedia los votos en $s \leftarrow s/|Classifiers|$
7: $s' \leftarrow$ Promedio de previos votos almacenados en Q
8: **si** la cola Q esta llena ($|Q| = \nu$) **entonces**
9: Remover el primer elemento de Q, el cual es el valor más antiguo
10: **fin si**
11: Añade s a Q
12: **devolver** Promedio de dos votos s y s'; i.e., $(s' + s)/2$

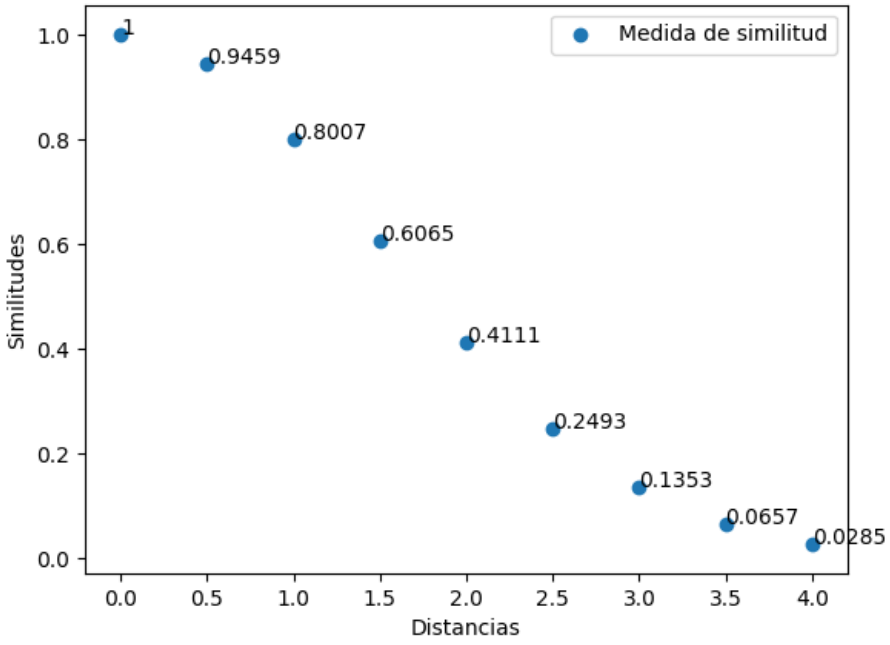


Figura 3.1: Análisis de la función de similitud

3.3. Un nuevo algoritmo basado en aprendizaje de ensamble para detección de anomalías - *Bagging-RandomMiner*

La aportación principal de este trabajo de investigación es proponer un algoritmo de clasificación de una clase que sea competitivo en los dos escenarios seleccionados: detección de intrusos en sesiones de computadora y detección de riesgos personales mediante el procesamiento de señales de bio-sensado. Después de un análisis de la literatura pertinente se diseñó el siguiente modelo, este demostró ser competitivo y superior en ambos casos. A continuación se detalla la construcción del método.

Bagging-RandomMiner (BRM) es una colección de clasificadores de una sola clase (ensamble) y basa su aprendizaje en la aleatoriedad. El diseño del algoritmo fue una inspiración basada en *Bagging-TPMiner* 3.2, debido a su alto desempeño en el contexto de detección de intrusos. Sin embargo, el tiempo de ejecución de *Bagging-TPMiner* se aproxima a $O(n^2)$, siendo n el número de objetos de entrenamiento, lo que implica un problema al clasificar conjuntos de datos media-

namente grandes. *Bagging-RandomMiner* se modeló con la intención de mitigar la complejidad computacional sin perder el desempeño al evaluar un nuevo objeto. Su funcionamiento se basa en dos fases, aprendizaje y clasificación. La fase de aprendizaje inicia construyendo τ clasificadores con un porcentaje *RSPercent*. Internamente cada modelo selecciona al azar un porcentaje de los objetos de entrenamiento llamados los *Objetos más Representativos (MRO)*. Cada miembro del conjunto de *MRO* es similar en naturaleza a un objeto típico determinado por *Bagging-TPMiner*, excepto que los objetos típicos se encuentran utilizando un búsqueda exhaustiva. El siguiente paso es calcular la matriz de covarianza sobre los objetos muestreados por *RSPercent* y se determina el umbral de decisión δ que indica la distancia promedio de los *MRO*. Al finalizar, cada clasificador consta de una tripleta: el conjunto de *MRO*, el umbral de decisión δ y la matriz de covarianza M. En la fase de clasificación, para determinar si un nuevo objeto es genuino, el clasificador lo compara con todos los *MRO* encontrados en la fase de entrenamiento y selecciona el más cercano para calcular un valor de tipicalidad, este indicará la probabilidad de no ser un intruso. Cabe mencionar que la clasificación de un nuevo objeto es análoga a la propuesta por *Bagging-TPMiner* en la sección 3.2.2. En las siguientes secciones se indagará a detalle el funcionamiento de *Bagging-RandomMiner*.

3.3.1. Diseño del algoritmo

En esta sección, se explica a detalle las fases de entrenamiento y clasificación de *Bagging-RandomMiner* haciendo uso de pseudocódigos para ejemplificar el funcionamiento.

Fase de entrenamiento

Como se mencionó anteriormente, *Bagging-RandomMiner* es un algoritmo basado en ensamble, lo que implica la construcción de N clasificadores de una sola clase para etiquetar un nuevo objeto. En el algoritmo 3.4 se lleva a cabo el aprendizaje, los parámetros de entrada son: T el conjunto de entrenamiento, N el número de clasificadores a considerar en el ensamble, *RSPercent* el porcentaje de objetos seleccionados aleatoriamente del conjunto T para construir los clasificadores, *MROPercent* la fracción de objetos aleatorios que corresponderán a los *MRO* del conjunto T'. Al inicio, en el paso 3 se realiza la selección aleatoria de objetos con reemplazo para el clasificador i, tomando como parámetros el conjunto T y la porción *RSPercent*. Después, se determina la matriz

de covarianza del conjunto T' (paso 4). El siguiente paso, realiza una selección aleatoria **sin reemplazo** del conjunto de datos T' respecto al porcentaje *MROPercent*, estos objetos serán los MRO_i del clasificador (paso 5). En consecuencia, con ayuda de la matriz de covarianza M_i y los MRO_i se calcula el umbral de decisión δ determinado por el promedio de la matriz de distancias de los *MRO* (paso 6). Finalmente, en el paso 7 se almacenan las tripletas de cada uno de los clasificadores: *MRO*, M y δ.

Algoritmo 3.4 Construcción del ensamble de *Bagging-RandomMiner*

Entrada: T: conjunto de entrenamiento; N: número de clasificadores en el ensamble; $RSPercent$: fracción del conjunto de entrenamiento a muestrear; $MROPercent$: la porción de objetos más representativos (MRO) a seleccionar.

Salida: $Params$: tripletas de los N clasificadores calculados.

1: $Params \leftarrow ()$
2: **para** $i = 1...N$ **hacer**
3: $T' \leftarrow$ SelecciónDeObjetosDelClasificador (T, $RSPercent$)
4: $M_i \leftarrow$ ComputarMatrizCovarianza (T')
5: $MRO_i \leftarrow$ SelecciónAleatoriaDeMROs (T', $MROPercent$)
6: $\delta_i \leftarrow$ AvgDistancias (T', M_i)
7: $Params \leftarrow Params \bigcup \{(MRO_i,\ M_i,\ \delta_i)\}$
8: **fin para**
9: **devolver** $Params$

Fase de Clasificación

La clasificación de un nuevo objeto se realiza en el algoritmo 3.5, este algoritmo es análogo al propuesto en la fase de clasificación de *Bagging-TPMiner* 3.3, con la pequeña diferencia que la comparación se realiza con los *MRO* de cada tripleta y no con los objetos típicos que propone *BTPM*. Los parámetros de entrada del algoritmo 3.5 son: o el objeto a clasificar, *Params* la tripleta de cada clasificador en el ensamble, Q la cola para el almacenamiento de resultados pasados y ν el número de objetos pasados a considerar en la clasificación actual. Este algoritmo retorna la probabilidad de no ser un intruso. Inicialmente, en el paso 2 se itera sobre cada tripleta obtenida por el algoritmo 3.4 con la finalidad de encontrar el *MRO* más cercano de cada clasificador

(d_{min}). La distancia calculada se transforma en una medida de similitud determinada por la función $e^{-0.5(d_{min}/\delta_i)^2}$, la cual corresponde al intervalo [0-1] e indica la probabilidad de pertenencia al comportamiento genuino. En el paso 6, se promedia cada una de las similitudes determinadas por los clasificadores y se almacenan en la cola Q, los pasos 8 y 9 revisan la longitud de la cola para decidir si es necesario eliminar el primer elemento (el más antiguo) e ingresar el valor de s, en caso contrario, de existir capacidad simplemente se añade el valor (paso 11). Finalmente, el paso 12 retorna el valor probabilístico de no ser etiquetado como intruso, el cual se compone de un promedio entre s (el promedio actual) y s' (el promedio actual de la cola).

Algoritmo 3.5 Fase de clasificación de Bagging-RandomMiner

Entrada: o: objeto a clasificar; $Params$: miembros de ensambles retornados por el algoritmo 3.4; Q: cola con resultados de clasificaciones pasadas; ν número de objetos pasados a considerar en la clasificación actual.

Salida: probabilidad de no ser intruso

1: $s \leftarrow 0$ s almacenará los votos para los clasificadores existentes
2: **para** $C \in Params$ **hacer**
3: $d_{min} \leftarrow$ Distancia entre o y el objeto más cercano en C
4: Añade a s el voto $e^{-0.5(d_{min}/\delta_i)^2}$
5: **fin para**
6: Promedia los votos en $s \leftarrow s/|Params|$
7: $s' \leftarrow$ Promedio de previos votos almacenados en Q
8: **si** la cola Q esta llena ($|Q| = \nu$) **entonces**
9: Remover el primer elemento de Q, el cual es el valor más antiguo
10: **fin si**
11: Añade s a Q
12: **devolver** Promedio de dos votos s y s'; i.e., $(s' + s)/2$

3.3.2. Diagrama de flujo de la fase de entrenamiento

En la figura 3.2 se muestra la fase de entrenamiento del algoritmo *Bagging-RandomMiner*.

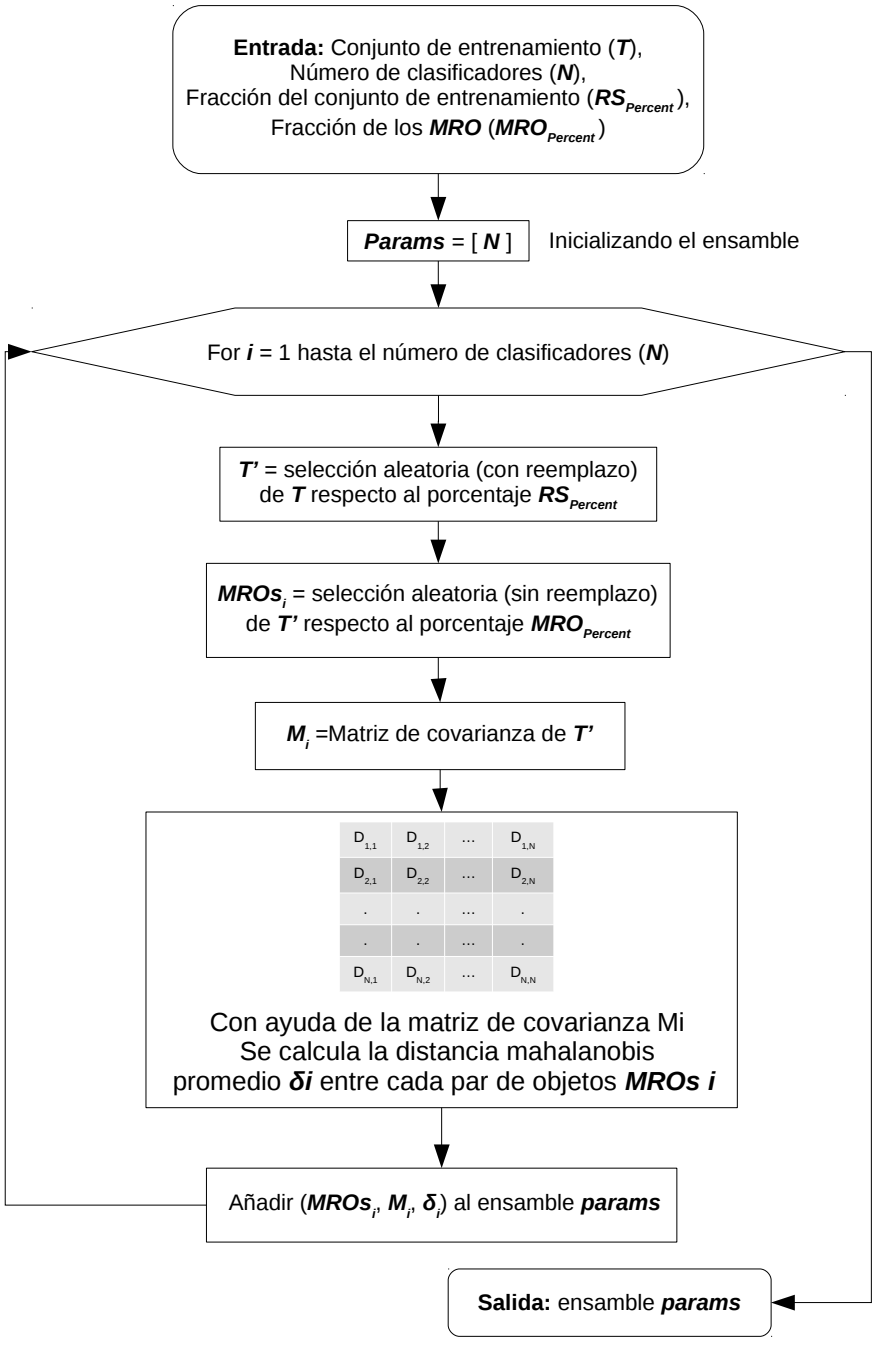


Figura 3.2: Diagrama de flujo de la fase de aprendizaje de *Bagging-RandomMiner*

3.3.3. Diagrama de flujo de la fase de clasificación

En la figura 3.3 se muestra la fase de clasificación del algoritmo *Bagging-RandomMiner*.

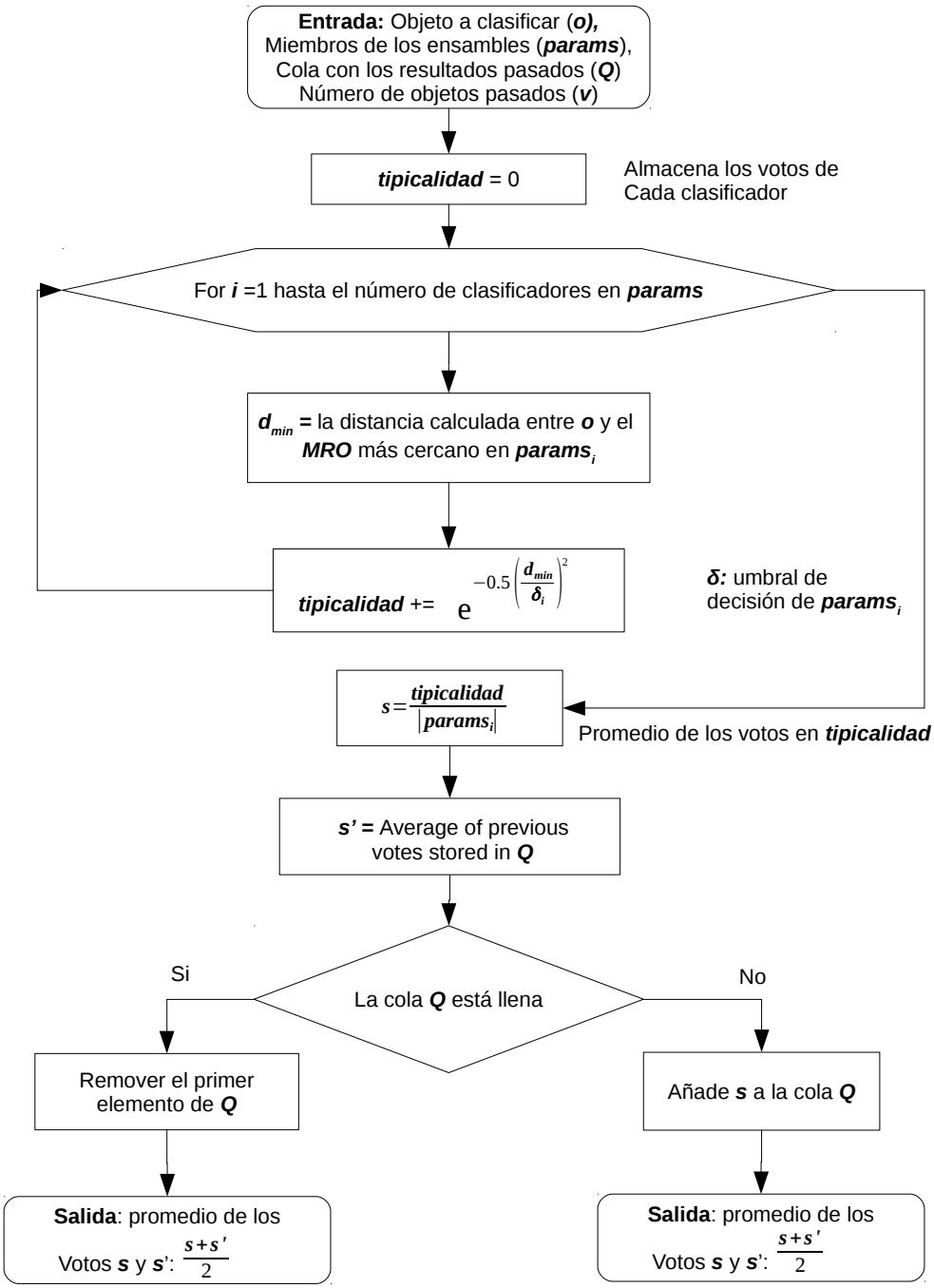


Figura 3.3: Diagrama de flujo de la fase de clasificación de *Bagging-RandomMiner*

3.4. Fronteras de decisión de *Bagging-TPMiner* vs *Bagging-RandomMiner*

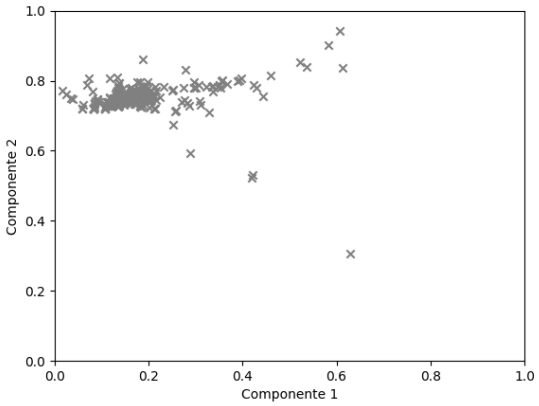
Debido a que *Bagging-RandomMiner* fue inspirado en *Bagging-TPMiner*, es interesante comparar las regiones de decisión de ambos algoritmos, esto permitirá comprender las ventajas y debilidades de ambos métodos. Lo referente a *Bagging-RadomMiner* se muestra en las figuras 3.4 (a y c) y los resultados de *Bagging-TPMiner* se observa en las figuras 3.4 (b y d). Si analizamos las regiones de decisión podemos concluir ciertos aspectos.

Los objetos típicos (P) calculados por *Bagging-TPMiner* ilustran una representación más completa del conjunto de datos, debido a que la naturaleza del método es llegar a las regiones más alejadas y poco visitadas por el usuario. Este enfoque ha permitido que *Bagging-TPMiner* sea el algoritmo más novedoso en el contexto de detección de intrusos, sin embargo, existen algunas limitaciones importantes por mencionar, la fase de aprendizaje del algoritmo es muy costosa, alcanzando una complejidad de *O(n^2)* donde n representa el número de instancias de entrenamiento. Claramente esto se vuelve un problema cuando el conjunto de datos es medianamente grande, por otro lado, la fase de clasificación tiene una complejidad de $O(N \cdot P \cdot m)$ donde N es el número de clasificadores, P el número de objetos típicos y m el número de instancias de prueba.

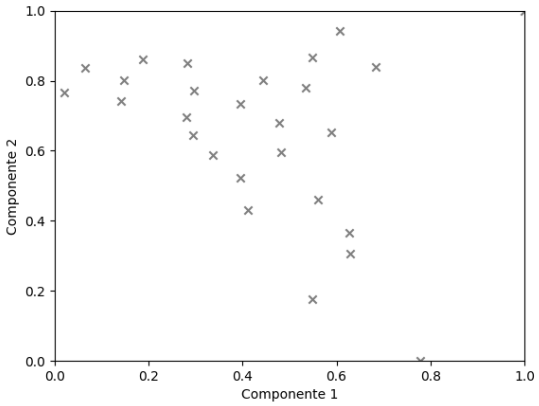
En el caso de *Bagging-RandomMiner* los *MRO* calculados permiten diseñar una frontera de decisión menos extensa y considerando exclusivamente las regiones más densas de la población, este cambio al parecer favorece la detección de intrusos, además, la fase de aprendizaje será de $O(MRO^2)$ que se reduce a una función semi-cuadrática debido a que no depende directamente del número de instancias de entrenamiento y esto permite trabajar en conjuntos de datos medianamente grandes. En cambio, en la fase de clasificación alcanza una complejidad de $O(N \cdot MRO \cdot m)$, siendo N el número de clasificadores y los *MRO* la única diferencia con *Bagging-TPMiner*. Cabe mencionar que *BTPM* es constante respecto al número de objetos típicos que encuentra y empíricamente se puede intuir que el número extrañamente rebasará las centenas, sin embargo, *Bagging-RandomMiner* dependerá directamente de los porcentajes de selección aleatoria y en conjuntos de datos muy grandes podría convertirse en una problemática.

En las figuras 3.5 (a y b) se muestra el comportamiento de las asíntotas superiores de la O grande, para ejemplificar se consideraron diferentes escenarios, donde el número de instancias totales fue

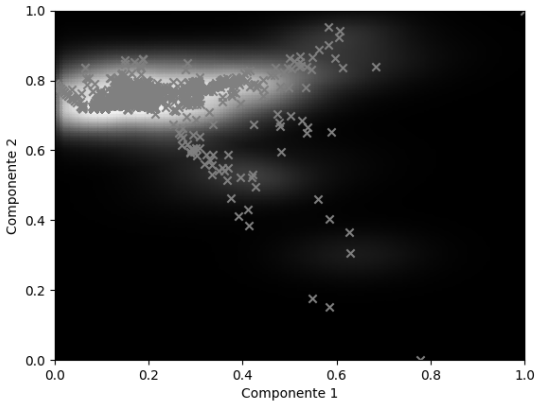
aumentando en 500 unidades hasta llegar a 500,000 para visualizar la tasa de crecimiento de los algoritmos. Se construyó un ensamble (N) de 100 clasificadores, en ambos algoritmos el porcentaje de selección se definió en 15 $\%$ y en el caso de *Bagging-RandomMiner* se definió un porcentaje de *MRO* (*MROPercent*) del 10 $\%$.



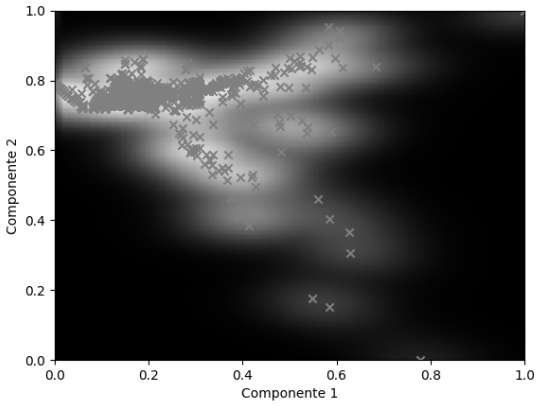
(a) MROs



(b) Objetos Típicos

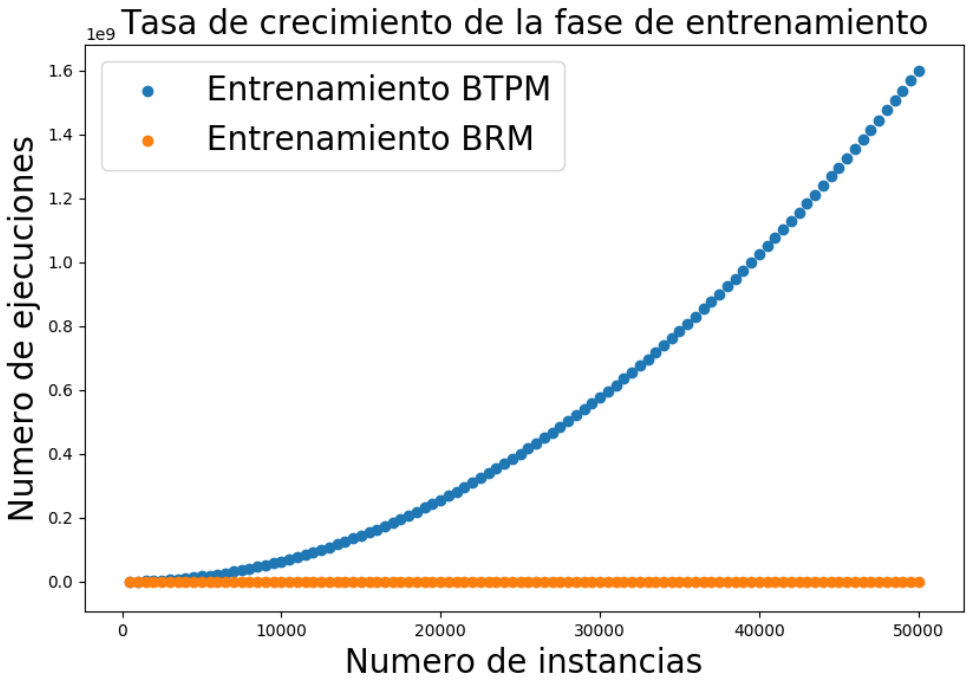


(c) Frontera de decisión *RandomMiner*

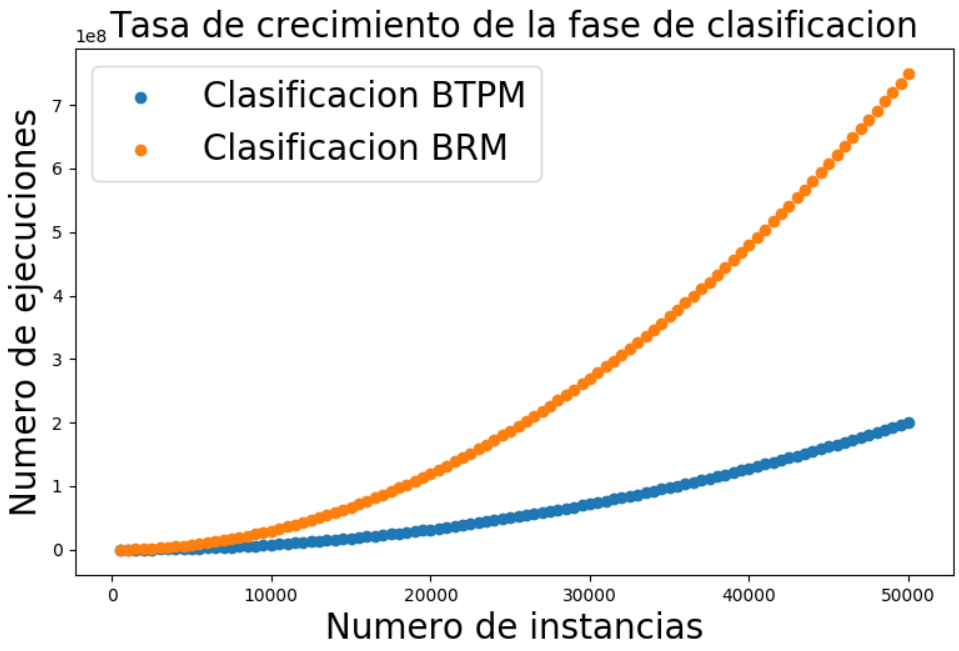


(d) Frontera de decisión *TPMiner*

Figura 3.4: La figura (a) muestra la distribución de los objetos más representativos (MROs) calculados por *RandomMiner*, la figura (b) indica los objetos típicos encontrados por *TPMiner*, en la figura (c) se observa la frontera de decisión obtenida por *RandomMiner* y la figura (d) representa la región de decisión del algoritmo *TPMiner*



(a) Entrenamiento



(b) Clasificación

Figura 3.5: La figura (a) muestra la tasa de crecimiento de la fase de aprendizaje de ambos algoritmos y la figura (b) muestra la tasa de crecimiento de la fase de clasificación de ambos algoritmos

3.5. Experimentos y resultados sobre *WUIL*

Para verificar el comportamiento de nuestro algoritmo *Bagging-RandomMiner*, se evaluaron los 70 usuarios propuestos en la base de datos *WUIL* y se comparó contra las mejores metodologías propuestas por la literatura.

3.5.1. Métricas de evaluación

Para medir la eficiencia del algoritmo se utilizaron tres métricas, *AUC* (por sus siglas en inglés, *Area Under Curve*), *ZFP* (por sus siglas en inglés, *Zero False Positive*) y la prueba estadística no-paramétrica de *Friedman*.

Area Under Curve (*AUC*)

El área bajo la curva (*AUC*) evalúa la tasa de detección positiva (comportamiento genuino) o *TP* (por sus siglas en inglés, *True Positive*) contra la tasa de detección de falsos positivos (comportamiento anormal) o *FP* (por sus siglas en inglés, *False Positive*) y se especifica con un valor numérico ($0 <= AUC <= 1$) que representa el área cubierta al construir la curva de aprendizaje. Este indicador de acuerdo con [2, 60] es una excelente herramienta para evaluar clasificadores de una clase, en la figura 3.6 se muestran sus componentes

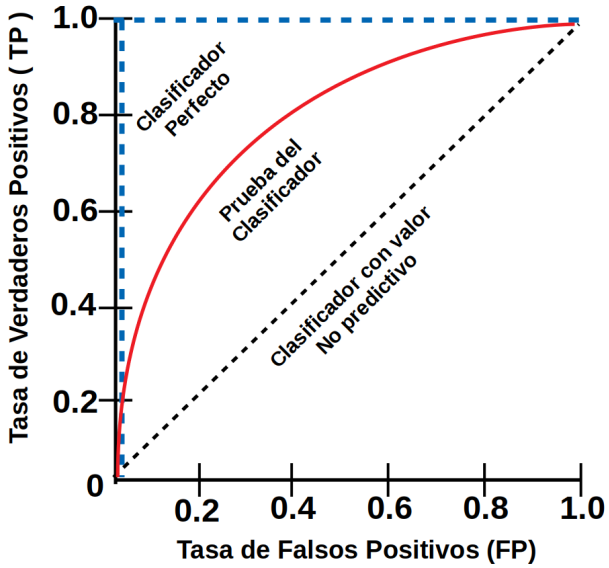


Figura 3.6: Interpretación del área bajo la curva (*AUC*), figura editada de [2].

***Zero False Positive* (*ZFP*)**

La métrica de *ZFP* mide el número de *TP* cuando no se han detectado ningún *FP*, es decir, el valor de *TP* cuando *FP* = 0. Cuanto más alto es el valor de *ZFP*, más confiable es el sistema de detección de anomalías. Esta medida de evaluación se ha implementado en trabajos como [34, 61] debido a que, detecta el número de alertas de falsos positivos en breves intervalos de tiempo, esto resulta útil ya que se vuelve molesto para los usuarios recibir alertas falsas continuamente.

Prueba no-paramétrica de *Friedman*

La prueba estadística de *Milton Friedman* es un equivalente no-paramétrico a la prueba de medidas repetidas *ANOVA* (por sus siglas en inglés, *ANalysis Of VAriance*). La prueba de *Friedman* clasifica los algoritmos para cada conjunto de datos por separado, el mejor algoritmo obtiene un rango de 1, el segundo un rango de 2 y así sucesivamente. En el caso de un empate se asigna un rango promedio [62].

Esta prueba compara los rangos promedios de los algoritmos, bajo la hipótesis nula de que todos los algoritmos son equivalentes y que sus rangos deben ser iguales. La prueba estadística de *Friedman* tiene una distribución de χ^2 (ecuación 3.1)

$$\chi^2 = \frac{12N}{k(k+1)} \left[\sum_j R_j^2 - \frac{k(k+1)^2}{4} \right] \tag{3.1}$$

con $k-1$ grados de libertad, siempre y cuando (k > 5 y N > 10), donde, k es el número de algoritmos, R el rango del algoritmo y N el número de conjuntos de datos. En el caso de que k y N sean más pequeños es recomendable utilizar la distribución *F* (ecuación 3.2)

$$F = \frac{(N-1)\chi^2}{N(k-1) - \chi^2} \tag{3.2}$$

con (k-1)(N-1) grados de libertad. El desempeño de dos clasificadores es significativamente diferente si el correspondiente rango promedio difiere con la diferencia crítica (ecuación 3.3)

$$CD = q_\alpha \sqrt{\frac{k(k+1)}{6N}} \tag{3.3}$$

donde los valores críticos q_α se basan en el estadístico de rango de *Studentized* dividido por $\sqrt{2}$

Diagrama de diferencia crítica (*CD*)

Cuando queremos comparar los resultados obtenidos por la prueba no-paramétrica de *Friedman* se puede representar en un diagrama de diferencia crítica (*CD*) [63], en la figura 3.7 se muestra un ejemplo, la línea superior en el diagrama es el eje en el que trazamos los rango promedio de los métodos. El eje define que los rangos más bajos (mejor algoritmo) estén a la derecha y se conectan con una línea horizontal aquellos algoritmos que no son significativamente diferentes entre sí.

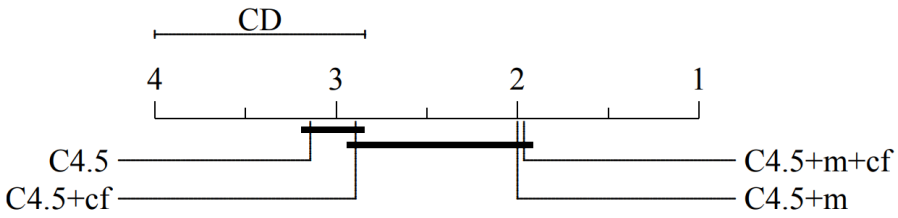


Figura 3.7: Ejemplo de diagrama de diferencia crítica (*CD*). El mejor algoritmo es el *C4.5+m+cf* aunque solo es significativamente diferente al *C4.5*

3.5.2. Resultados

Para mostrar el comportamiento de *Bagging-RandomMiner* respecto al estado del arte se capturaron los resultados más representativos sobre el conjunto de datos *WUIL*, estos fueron obtenidos por *Bagging-TPMiner* diseñado por Medina et al. [34], además, estos autores utilizaron un *Support Vector Machine* [64], *Parzen Window Classifier* [65] y dos versiones de *Parzen* que hacen uso de *k-Means*, uno basado en [59] denominado *k-Means1* y otro en [54] denominado *k-Means2*.
La configuración de *Bagging-RandomMiner* fue establecida en base a conocimiento empírico, su ejecución consta de:

- **N**: un ensamble constituido por 100 clasificadores.
- **RSpercent**: cada clasificador consta del 15 % del conjunto de entrenamiento
- **MROpercent**: particularmente el número de objetos más representativos es una selección aleatoria de 10 %
- **Tipo de distancia**: la distancia utilizada fue *Mahalanobis*

- ν: el número de objetos a considerar en la cola es de 3

A pesar de que *WUIL* se constituye por 70 usuarios, los autores de [34] solo reportan hasta el usuario 48, por esta razón se muestran los resultados de solo este número de participantes para poder realizar una comparación equitativa. En la tabla 3.1 se muestran las puntuaciones obtenidas para la métrica de *AUC*.

En la figura 3.8 se ilustra una representación gráfica (diagrama de radar) del comportamiento de los algoritmos respecto a la métrica *AUC*.

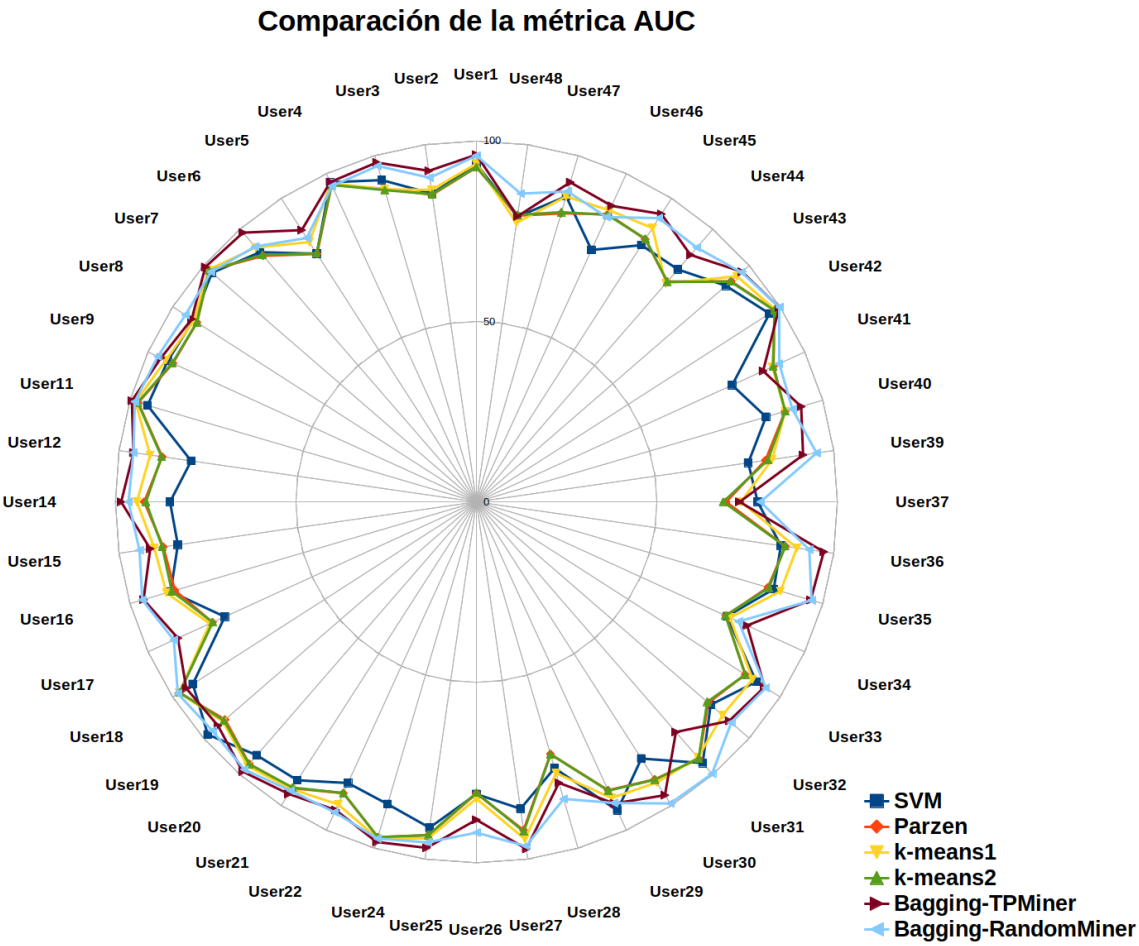


Figura 3.8: Gráfico de radar con el desempeño de los clasificadores respecto a la métrica de área bajo la curva (*AUC*) de los 48 usuarios seleccionados del conjunto de datos *WUIL*. Los puntos más alejados del centro tienen el mejor desempeño.

Tabla 3.1: Comparación de la métrica *AUC* en el conjunto de datos *WUIL*

Usuario	SVM	Parzen	k-Means1	k-Means2	Bagging-TPMiner	Bagging-RandomMiner
1	93.67	92.73	93.73	92.77	**96.23**	95.96
2	86.33	86.01	87.34	86.17	**92.68**	90.76
3	92.94	90.41	90.32	90.06	**98.03**	97.04
4	97.39	96.59	97.08	96.57	**97.44**	96.24
5	81.78	81.68	85.59	81.88	**89.52**	86.94
6	91.57	90.24	93.29	90.58	**98.74**	93.62
7	96.94	97.96	98.26	97.92	**99.38**	97.3
8	93.19	92.04	92.97	92.02	93.74	**95.82**
9	94.01	92.44	94.59	92.62	96.06	**97.05**
11	94.95	97.65	98.43	97.8	**99.48**	98.63
12	79.83	87.93	91.32	88.11	95.98	**95.99**
14	84.88	92.02	94.06	91.55	**98.42**	96.37
15	83.56	87.64	90.13	87.92	91.29	**94.29**
16	88.25	87.05	89.49	87.94	96.1	**96.44**
17	76.62	80.38	81.21	80.31	90.91	**92.16**
18	93.39	97.84	97.87	97.73	95.55	**98.31**
19	**98.46**	92.13	92.92	92.51	94.73	96.81
20	92.91	96.07	96.83	96.17	**98.88**	98.13
21	91.71	94.3	94.63	94.12	**96.21**	95.16
22	85.64	88.75	92.17	88.68	93.87	**94.54**
24	87.3	96.86	97.27	96.83	**98.22**	97.24
25	91.21	93.31	94.25	93.28	**96.86**	95.34
26	80.97	80.8	82.25	80.61	88.09	**91.66**
27	85.92	91.91	94.38	92.3	**97.14**	96.36
28	76.89	72.84	78.36	72.91	81.15	**85.79**
29	**94**	87.86	90.2	87.96	91.87	91.72
30	84.55	91.4	92.44	91.52	96.62	**99.34**
31	95.75	94.14	93.73	94.04	84.4	**99.7**
32	85.88	85	90.23	84.63	92.67	**93.42**
33	92.19	88.53	90.93	88.58	94.89	**95.31**
34	76.31	75.95	77.22	75.86	**82.47**	79.86
35	85.84	84.17	87.8	84.48	96.52	**96.79**
36	85.23	86.51	89.7	86.39	**97.22**	93.16
37	77.92	69.16	73.3	68.39	72.95	**78.7**
39	76.09	80.91	82.78	81.61	91.43	**95.22**
40	83.62	89.15	89.17	89.17	**93.78**	91.24
41	77.9	90.49	90.12	90.38	87.33	**92.16**
42	96.52	98.2	98.46	98.18	99.74	**99.82**
43	91.48	93.38	95.51	93.25	**97.37**	97.17
44	85.24	80.79	80.09	80.74	90.57	**93.24**
45	84.58	86.51	90.23	86.67	**94.83**	93.48
46	76.76	87.69	88.81	87.43	**90.2**	86.78
47	88.25	83.33	88.21	83.68	**92.32**	89.68
48	79.84	80.13	78.14	80.36	79.94	**86.3**

En la tabla 3.1 se muestra en negritas el mejor clasificador por usuario, en promedio *Bagging-RandomMiner* superó a *Bagging-TPMiner* pasando de 93.2 a 93.8 %. Para determinar si existe diferencia significativa entre los distintos algoritmos se aplicó la prueba de *Friedman* y se construyó un diagrama de diferencia crítica *CD* ilustrado en la figura 3.9

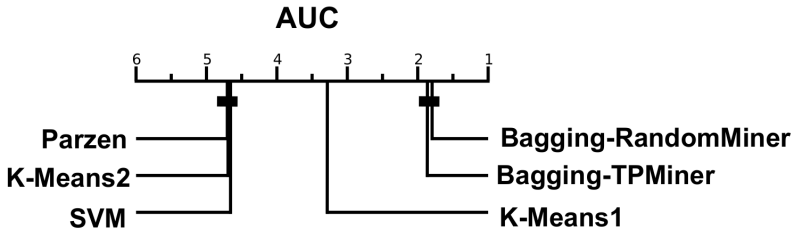


Figura 3.9: Diagrama *CD* con la comparación estadística de *SVM*, *Parzen*, *k-Means1*, *k-Mean2*, *Bagging-TPMiner* y *Bagging-RandomMiner* respecto a la métrica de *AUC*.

Después de observar los resultados, se puede concluir que el algoritmo *Bagging-RandomMiner* en la métrica de *AUC* según la prueba no-paramétrica de *Friedman* es superior significativamente a los algoritmos del estado del arte, excluyendo a *Bagging-TPMiner*, aunque el promedio general de *Bagging-RandomMiner* supera a *Bagging-TPMiner* no es significativo. Sin embargo, existe una gran diferencia en los tiempos de ejecución de ambos algoritmos (*Bagging-RandomMiner y Bagging-TPMiner)*, esto se puede visualizar en la tabla 3.2, en ella se promediaron los 48 usuarios de *WUIL*. Es evidente, que además de la superioridad de *Bagging-RandomMiner* en la métrica de *AUC* el desempeño en el tiempo de ejecución lo posiciona como el algoritmo más novedoso en la detección de intrusos.

Tabla 3.2: Comparación de tiempos en segundos de *Bagging-TPMiner vs Bagging-RandomMiner*, promediando los 48 usuarios.

Tiempo	Bagging-TPMiner	Bagging-RandomMiner
Aprendizaje	134.17 seg	**1.2 seg**
Clasificación	31.14 seg	**10.11 seg**
Total	165.32 seg	**11.31 seg**

La métrica de *Zero False Positive* se evaluó de manera análoga que el *AUC*. En la tabla 3.3 se muestran los resultados de los 48 usuarios de los algoritmos a evaluar respecto a la métrica *ZFP*.

Tabla 3.3: Comparación de la métrica *ZFP* en el conjunto de datos *WUIL*.

Usuario	SVM	Parzen	k-Means1	k-Means2	Bagging-TPMiner	Bagging-RandomMiner
1	0	15.39	21.54	17.69	43.85	**79.67**
2	3.48	3.48	0	3.48	8.7	**72.5**
3	46.67	32.38	37.14	32.38	57.14	**80.33**
4	44.44	41.48	42.22	41.48	51.85	**76**
5	2	9.33	9.33	8	26	**34.7**
6	0	10.77	8.46	10	30	**55.49**
7	9.66	20	20	20	35.86	**89.29**
8	42	30.67	36.67	33.33	52	**83.65**
9	12.67	8.67	14.67	9.33	34	**78.25**
11	9.17	13.33	12.5	10	37.5	**95.78**
12	13.85	25.39	26.92	23.85	34.62	**59.29**
14	33.1	24.14	24.14	23.45	44.83	**80.41**
15	12.41	11.03	22.76	11.03	30.35	**61.74**
16	21.43	1.43	4.29	1.43	35.71	**69.41**
17	0	5.33	1.33	5.33	10	**53.94**
18	39.26	46.67	42.96	46.67	65.19	**78.83**
19	6.67	2.11	4.21	2.11	37.9	**89.96**
20	48	34.67	38	34.67	63.33	**66.03**
21	22.86	22.86	23.57	20.71	36.43	**47.96**
22	10.83	3.33	4.17	3.33	27.5	**79.32**
24	40.74	54.82	54.07	54.82	56.3	**65.92**
25	28	28.67	28.67	28.67	54	**72.44**
26	6.67	5.93	4.44	4.44	9.63	**54.3**
27	6.09	24.35	22.61	23.48	35.65	**70.91**
28	0.67	0	0	0	0	**33.72**
29	**35.83**	29.17	31.67	26.67	35	34.72
30	22	14	14	14	33.33	**93.49**
31	**79.17**	72.5	70.83	71.67	25.83	58.43
32	24.8	27.2	23.2	24	33.6	**34.16**
33	30.37	24.44	27.41	25.93	**42.96**	39.63
34	8.57	0	6.43	0	**16.43**	15.64
35	0	0	0	0	17.14	**80.26**
36	7.14	2.86	4.29	2.86	45	**65.29**
37	0	7.69	8.46	9.23	28.46	**31.51**
39	16.84	11.58	8.42	11.58	24.21	**78.58**
40	26.32	30.53	30.53	30.53	27.37	**57.22**
41	12	44.8	43.2	44.8	19.2	**32.81**
42	55.17	48.28	46.9	47.59	76.55	**92.8**
43	18.57	19.29	20	20	25	**90.35**
44	7.37	17.14	16.43	15.71	24.29	**63.8**
45	13.57	27.14	27.86	27.14	38.57	**64.26**
46	**50.34**	37.24	37.24	37.24	45.52	44.34
47	28.15	19.26	23.7	22.22	**44.44**	40.97
48	0	5.33	5.33	5.33	0	**33.12**

Los mejores resultados se resaltan en negritas, para una mejor visualización se ilustra en la figura 3.10 un gráfico de radar del comportamiento de los algoritmos respecto a la métrica *ZFP*.

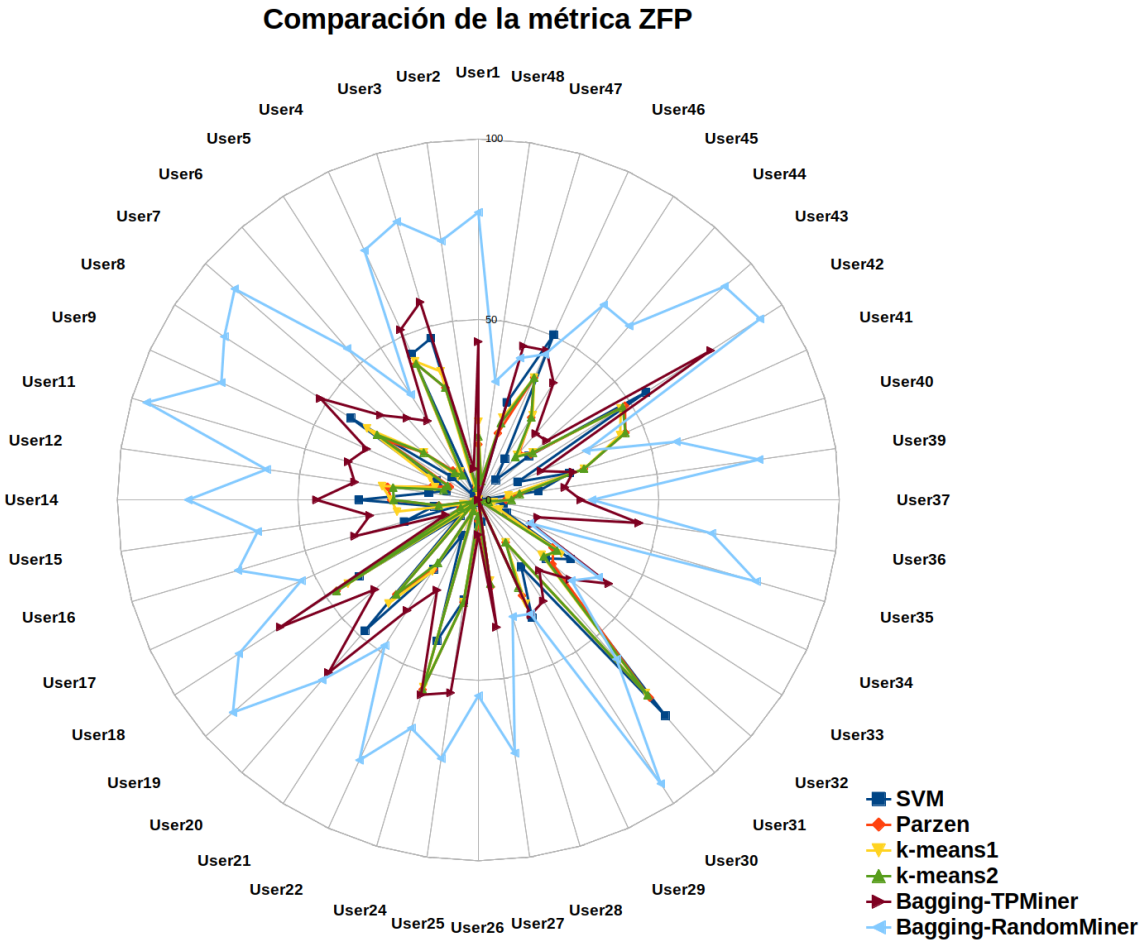


Figura 3.10: Gráfico de radar con el desempeño de los clasificadores respecto a la métrica de *Zero False Positive* (*ZFP*) de los 48 usuarios seleccionados del conjunto de datos *WUIL*. Los puntos más alejados del centro tienen el mejor desempeño.

Por último, se aplicó la prueba estadística no-paramétrica de *Friedman* para determinar si *Bagging-RandomMiner* supera significativamente a los demás algoritmos, en la figura 3.11 se muestra el diagrama de diferencia crítica *CD*.

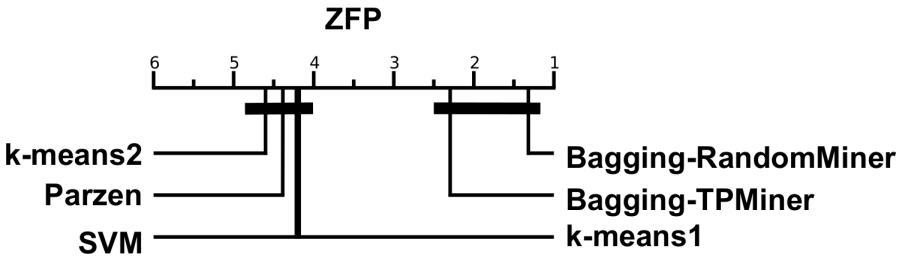


Figura 3.11: Diagrama *CD* con la comparación estadística de *SVM*, *Parzen*, *k-Means1*, *k-Mean2*, *Bagging-TPMiner* y *Bagging-RandomMiner* respecto a la métrica de *ZFP*.

En este caso, *Bagging-RandomMiner* obtuvo resultados muy similares que al evaluar la métrica de *AUC*, debido a que según la prueba no-paramétrica de *Friedman* el algoritmo supera significativamente a las metodologías del estado del arte, excluyendo a *Bagging-TPMiner*, aunque el promedio general en esta ocasión es superior en gran medida, no es suficiente para vencer a *Bagging-TPMiner* estadísticamente.

Se puede concluir que el algoritmo propuesto en este trabajo de investigación, denominado *Bagging-RandomMiner* es superior a los algoritmos más novedosos para la detección de intrusos en sesiones de computadora. Además, el tiempo de ejecución del algoritmo le da una gran ventaja para ser implementado en un escenario en tiempo real.

Capítulo 4

Escenario 2: Detección de riesgos personales mediante el procesamiento de señales de bio-sensado

Resumen: En este capitulo se evalúa el desempeño de *Bagging-RandomMiner* en el escenario de Detección de riesgos personales mediante el procesamiento de señales de bio-sensado. La base de datos seleccionada consta de aproximadamente 7.5 millones de instancias y aunque *Bagging-RandomMiner* se ejecuta en un tiempo semi-cuadrático la velocidad de respuesta no será inmediata, esto nos motivó a desarrollar una versión en la metodología de cómputo distribuido aprovechando las bondades de *frameworks* como *Apache Spark*.

4.1. Revisión de literatura

Durante los últimos años, la innovación tecnológica ha causado un crecimiento exponencial en distintas ramas de la ciencia, en su mayoría con la intención de mejorar la calidad de vida de los seres humanos. Entre los múltiples temas de investigación se destacan, el monitoreo de signos vitales *VSM* (por sus siglas en inglés, *Vital Signs Monitoring*) y el reconocimiento de actividad humana *HAR* (por sus siglas en inglés, *Human Activity Recognition*) debido a la alta diversidad de sensores portátiles existentes, ampliando la aplicabilidad de estos aparatos en escenarios reales. Algunos trabajos realizados por los investigadores se basan en la detección generalizada y ubicua, compu-

tación móvil y la vida ambiental asistida [66–69], con aplicaciones en campos como: seguridad, salud, medicina, entretenimiento y defensa militar [70–72].

Es interesante como los sensores portátiles nos permiten medir una cantidad inmensa de información de un usuario, sin embargo, en la mayoría de los casos *VSM* y *HAR* se enfrentan a problemáticas bastante complicadas de resolver. Esto causa que la construcción de conjuntos de datos confiables y públicos se vuelva todo un reto [66–72]. Existen complejidades como: el reclutamiento de voluntarios por preocupaciones de privacidad y anonimato, conflictos legales por la publicación de los datos y aspectos técnicos (número de sensores, la ubicación, el tiempo de medición, entre muchos más) que complican la construcción de los conjuntos de datos. A continuación se muestran algunos trabajos relacionados con la detección de riesgos personales mediante el procesamiento de bio-sensado.

La detección de riesgos personales [8] es conocida como la identificación oportuna de cuando una persona se encuentra en una situación peligrosa, como una crisis de salud, un accidente automovilístico y todos aquellos escenarios que pongan en peligro la integridad física de una persona.

Trejo et al. [36] en el 2010 desarrollaron un sistema denominado *ELISA* (por sus siglas en inglés, *Emergency, Positioning, and Immediate Assistance System*) con el objetivo de aumentar en gran medida la posibilidad de asistir a una persona, que sufra de alguna situación de emergencia. El número de usuarios oscila en los 750 y sigue en crecimiento. El sistema se desarrolló en la plataforma de *Blackberry* y su intención es expandirse a otras plataformas como *iPhone* y dispositivos de *Google*. *ELISA* fue diseñado para alertar 4 situaciones distintas, una crisis de salud (código I), un accidente automovilístico (código II), integridad física (código III) y algún derecho preferente por alguna discapacidad (código IV). Al activar manualmente una de las alarmas, se notifica directamente al departamento de seguridad para una asistencia inmediata. En el diseño original se tenía contemplado una detección automática en base al conjunto de sensores portátiles incluidos en un teléfono móvil, aunque esto sigue en desarrollo.

Los autores de [8] propusieron una base de datos que se basa en la detección de riesgos personales mediante el procesamiento de señales de bio-sensado, definida como *PRIDE* (por sus siglas en ingles, *Personal RIsk DEtection*). Está constituida por 23 usuarios y tiene la intención de simular

escenarios de riesgo que pueden presentarse en la vida cotidiana como, huir de una situación peligrosa, ser atacados por un agresor, estar inmersos en un sismo, presentar algún cuadro clínico que requiera asistencia inmediata y algunos otros escenarios de riesgo personal. La recolección de la información se obtuvo mediante la *Microsof band*, un dispositivo portátil con una serie de sensores que permiten conocer ciertos rasgos fisiológicos de los usuarios y algunos aspectos ambientales. Para evaluar la base de datos, los autores definieron algunas versiones de *k-Means*, *Parzen* y *ocSVM*, concluyendo que *ocSVM* es el mejor algoritmo para evaluar dicho conjunto de datos.

Por último, Jorge Rodríguez et al. [35] propusieron un nuevo algoritmo de detección de anomalías denominado *OCKRA* (por sus siglas en inglés, *One-Class K-means with Randomly-projected features Algorithm*), con el objetivo de clasificar el conjunto de datos *PRIDE*. *OCKRA* es un algoritmo basado en la técnica de ensamble y en la metodología de aprendizaje no supervisado *k-Means++* [52]. Aunque la idea del algoritmo difiere un poco a la metodología de ensamble, dado que *OCKRA* define un número de clasificadores pero variando las características del conjunto de entrenamiento y no el número de instancias. Los resultados concluyeron que *OCKRA* respecto a la métrica de *AUC* es superior a los demás algoritmos de *OCC* propuestos por la literatura, aunque esa diferencia según la prueba estadística no paramétrica de *Friedman* [63] no es significativa.

Es evidente que la detección de riesgos personales mediante el procesamiento de señales de bio-sensado es un área que tiene mucho potencial y que es muy poco explorada, sin embargo, podemos resumir algunos aspectos. El escenario seleccionado para medir el desempeño de *Bagging-RandomMiner* es *PRIDE*, debido a que la construcción del conjunto de datos asemeja un escenario real, además, el hecho de poseer más de 300,000 instancias por usuario justifica la implementación de *Bagging-RandomMiner* en un enfoque distribuido. Podemos observar que el algoritmo más novedoso y con el mejor desempeño en este conjunto de datos lo obtiene *OCKRA* [35], esto lo convierte en el método a superar en este segundo escenario de detección de anomalías.

4.1.1. *OCKRA*, un algoritmo del estado del arte para detección de anomalías en la base de datos *PRIDE*

One-Class K-means with Randomly-projected features Algorithm (*OCKRA*) es un algoritmo de ensamble propuesto para la detección de anomalías que basa su aprendizaje en *k-Means++* [35]. Cada clasificador es una proyección aleatoria de las características del conjunto de entrenamiento, esto garantiza la diversidad entre los clasificadores, de la misma manera que otros algoritmos lo hacen, por ejemplo, bosques aleatorios [73]. Primeramente, *OCKRA* aplica *k-Means++* [52] a cada subconjunto de características seleccionado y obtiene una colección de centroides por clasificador. Para clasificar un nuevo objeto, el algoritmo realiza el mismo procedimiento que *Bagging-TPMiner* 3.2 con una pequeña diferencia, al obtener el voto de similitud de cada clasificador simplemente se realiza un promedio y se elimina la cola y los promedios anteriores.

Fase de entrenamiento

El algoritmo 4.1 se encarga de la fase de aprendizaje de *OCKRA*, obteniendo como resultado un modelo entrenado constituido por un número finito de tripletas. El número de clasificadores (τ) se estableció en 100, según Breiman [73] empíricamente es un número que desempeña excelentes resultados, sin embargo, definir el número correcto seguirá siendo una pregunta abierta. Inicialmente, se tiene un conjunto de entrenamiento T de dimensiones $m \times n$, donde m es el número de instancias y n la cantidad de características. Para construir cada clasificador se hace una selección aleatoria con reemplazo de las características del conjunto de entrenamiento T (paso 3). En este paso, ν números aleatorios entre 1 y n se generan utilizando una distribución de probabilidad uniforme. Después de eliminar todos los números duplicados, los elementos restantes indican los índices de las características a seleccionar realizando una proyección (T') que tendrá un tamaño de $m \times n'$ (paso 4). En el paso 5, se calcula un umbral de decisión δ (correspondiente al promedio de la matriz de distancias de T'). Por ultimo, se aplica *k-Means++* para computar los k centroides ($k = 10$) que representan la distribución de cada clasificador. Al culminar la fase de entrenamiento, retorna un conjunto de parámetros de cada clasificador en particular, que consiste en una tripleta que contiene las características ($SelectedFeatures$) seleccionadas al azar, los centroides calculados de cada grupo ($Centres$) y el umbral de decisión (δ).

Algoritmo 4.1 Fase de aprendizaje *OCKRA*

Entrada: T: conjunto de entrenamiento; τ: número de clasificadores en el ensamble.

Salida: $OCKRAParameters$: parámetros calculados para este clasificador.

1: $OCKRAParameters \leftarrow \{\}$
2: **para** $i = 1...\tau$ **hacer**
3: $SelectedFeatures \leftarrow RandomFeatures(T)$
4: $T' \leftarrow$ Proyección(T, $SelectedFetures$)
5: $\delta_i \leftarrow$ AvgDistancias(T')
6: $Centres \leftarrow$ Calcula Centroides con K-Means(T')
7: $OCKRAParameters \leftarrow OCKRAParameters \bigcup \{(SelectedFeatures, Centres, \delta_i)\}$
8: **fin para**
9: **devolver** $OCKRAParameters$

Fase de clasificación

La clasificación de un nuevo objeto (Algoritmo 4.2) inicia con una instancia o que es un vector de dimensión $1 \times n$. Los siguientes pasos se aplican de manera iterativa sobre cada tripleta obtenida por el algoritmo 4.1. Inicialmente, se proyecta el objeto o en el espacio de características correspondiente (paso 5). Después, se calcula la distancia euclídea a cada centroide del clasificador y se almacena la más cercana a la cual se le aplica la medida de similitud para transformar la distancia en un valor definido en el intervalo de [0, 1] (paso 6 y 7). Al finalizar, cada valor de similitud obtenido por los clasificadores se promedia, este corresponde a la probabilidad de ser un comportamiento genuino (paso 10).

Algoritmo 4.2 Fase de clasificación *OCKRA*

Entrada: o: objeto a clasificar; $OCKRAParameters$: miembros del ensamble retornado por el algoritmo 4.1.

Salida: s: probabilidad de comportamiento normal

1: $s \leftarrow 0$ s almacenará los votos para los clasificadores existentes
2: **para** $(Features_i,\ Centres_i,\ \delta_i) \in OCKRAParameters$ **hacer**
3: $\quad o' \leftarrow Project(o,\ Features)$
4: $\quad d_{min} \leftarrow \min_{c_j \in Centres_i} EuclideanDistance(o',\ c_j)$
5: $\quad s \leftarrow s\ +\ e^{-0.5(d_{min}/\delta_i)^2}$
6: **fin para**
7: $s \leftarrow s/|Classifiers|$
8: **devolver** s

4.2. Introducción a un enfoque de cómputo distribuido para abordar problemas de detección de anomalías

La computación distribuida es un modelo diseñado para resolver problemas de cómputo masivo a través de un número finito de ordenadores organizados en *clusters* en una red de telecomunicaciones distribuida. Esto surgió debido a que el mundo se encuentra en una era tecnológica donde la generación de datos crece desmesuradamente, todos los días se generan quintillones de datos. La *International Data Corporation (IDC)* predijo en 2014 que para el 2020, el universo digital sería diez veces más grande que en el 2013[1]. Evidentemente esto resulta en una problemática significativa por resolver, con este en mente surgió el concepto de *Big Data* como un nuevo paradigma para permitir el almacenamiento y procesamiento de toda esta información.

Existen muchas definiciones de *Big data*, los autores de [3] lo definen como una colección de datos grande, complejos, que resultan muy difícil de procesar a través de herramientas de gestión y procesamiento de datos tradicionales. Muchos analistas usan el modelo de las 3V para definir *Big*

[1] IDC: The Digital Universe of Opportunities. 2018 [Online] Available: http://www.emc.com/infographics/digital-universe-2014.htm

data, estas representan, **volumen**, **velocidad** y **variedad** [74]

- **Volumen:** hace referencia al análisis de grandes cantidades de datos en distintos dominios de aplicación, por lo general a partir de decenas de *terabytes*.
- **Velocidad:** refleja la velocidad de generación de datos, así como su constante cambio. Por ejemplo, las publicaciones en *Twitter* como un *hashtag* tienen una alta velocidad, en algunos casos se mueven a una rapidez tan impresionante que la información que contienen no se puede almacenar fácilmente, pero aún necesita ser analizada.
- **Variedad:** se atribuye al hecho de que la inmensa cantidad de datos proviene de diferentes fuentes, en distintos formatos y estructuras. Por ejemplo, imágenes, vídeos, audio y texto de la redes sociales.

Big data es un paradigma que ha logrado dominar el procesamiento masivo de los datos en los últimos años. Aunque, desde el comienzo de la era de las computadoras, cada vez que surge un sistema informático más nuevo, más rápido y de mayor capacidad se presentan problemas que rebasan los límites de estas nuevas tecnologías. A pesar de esto, la industria ha logrado resolver ciertos percances combinando las capacidades de cómputo y almacenamiento de los sistemas en la red. Con este antecedente *Big data* ha tomado la iniciativa de resolver estos desafíos, por lo cual, surgió una nueva metodología con una distribución confiable y escalable denominada *MapReduce* [3].

4.2.1. Metodología MapReduce

La metodología *MapReduce* surgió a principios del siglo *XXI* por un grupo de investigadores de la reconocida compañía de *Google*, estas personas se dieron cuenta que la empresa tenia sistemas muy sofisticados para el rastreo web, la frecuencia de consultas etc., sin embargo, el aumento exponencial de usuarios se convertiría en un problema significativo. Estos investigadores dedujeron que la distribución de la carga de procesamiento podía dividirse en computadoras de bajo costo y luego conectarse a la red en forma de *cluster* [3].

La distribución por sí sola no es una respuesta suficiente. Esta distribución del trabajo debe realizarse en paralelo por los siguientes tres motivos [75]:

- El procesamiento debe poder expandirse y contraerse automáticamente.
- El procesamiento debe poder continuar independientemente de las fallas en la red o en los sistemas individuales.
- Los desarrolladores que aprovechan este enfoque deben poder crear servicios que otros desarrolladores puedan utilizar fácilmente.

Estos requisitos llevaron al diseño de *MapReduce* como un modelo de programación genérico que sea independiente de donde se han ejecutado los datos y los cálculos. Los ingenieros a cargo del proyecto lo llamaron *MapReduce* porque combina dos capacidades de los lenguajes funcionales existentes: mapear y reducir. A continuación, se muestran las características principales de los lenguajes funcionales y algunos de los motivos por los cuales predominan el cómputo distribuido.

- Los operadores en lenguajes funcionales no modifican la estructura de los datos.
- Los datos iniciales nunca son modificados.
- No es necesario gestionar explícitamente el flujo de los datos.
- No es necesario prescribir el orden de las operaciones, siendo esta una gran ventaja ya que en un *cluster* los cálculos se realizan en paralelo.

Función mapear (*map*): Aplica una función a cada elemento (definido como un par clave-valor) de una lista y produce una nueva lista [3].

Función reducir (*reduce*): La función *reduce* toma la salida de la función *map* y reduce la lista de la manera que desee el programador. En primera instancia, se crea un acumulador con un valor inicial, que después de procesar cada elemento de la lista en función de alguna operación solicitada, el acumulador devuelve los valores de la lista con dichas operaciones.

En la figura 4.1 se observa el proceso de la metodología *MapReduce*, se puede apreciar en la etapa de *map* que cada paso se realiza por separado, esto le permite ser tolerante a los fallos [3].

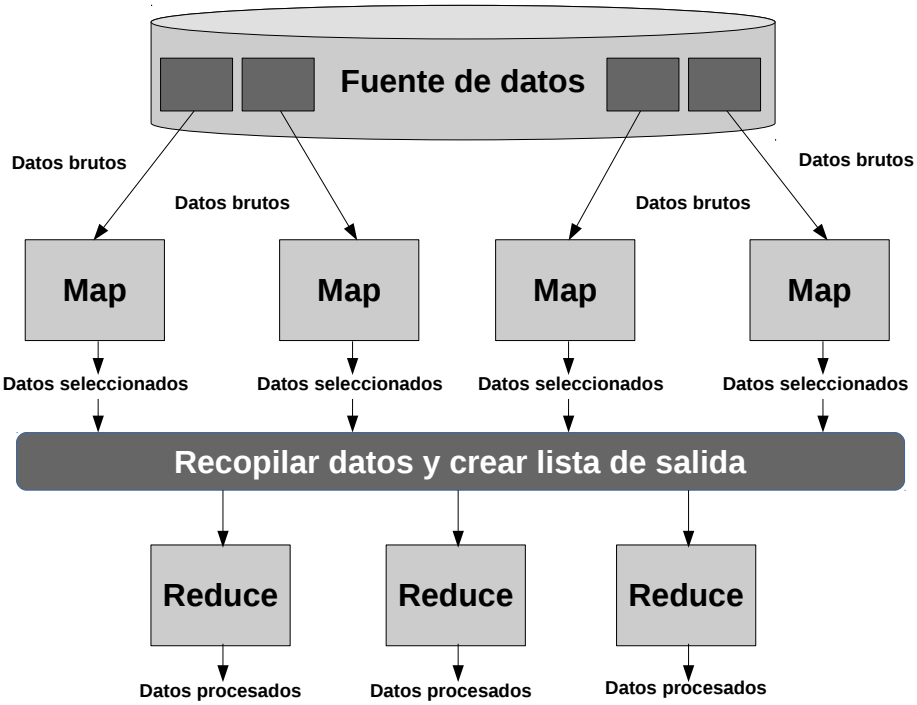


Figura 4.1: Flujo de datos del proceso de *MapReduce*, imagen editada de [3]

Por último, en la tabla 4.1 se muestran las principales características y limitaciones de *MapReduce*.

Tabla 4.1: Características y limitaciones de la metodología *MapReduce.*

Ventajas	**Limitaciones**
* Su arquitectura es escalable	* Aprendizaje automático:
* Planificación optimizada	Computación iterativa
* Elasticidad y disponibilidad	* Procesamiento de grafos
* Flexibilidad	* Procesamiento en tiempo
* Seguridad y Autenticación	real (*streams*)

4.2.2. *Apache Spark*

Apache Spark es una plataforma de computación en *cluster* diseñada para ser rápida y de propósito general [76]. Se basa en el modelo de *MapReduce* extendiendo la velocidad y los tipos de cálculo, incluyendo las herramientas de consultas interactivas. La característica más representativa y que sin duda alguna la distingue de la competencia es la capacidad de realizar los cálculos en memoria, permitiendo un aumento de velocidad de hasta 100 veces respecto a las demás plataformas como lo es *Apache Hadoop*. El diseño de *Apache Spark* cuenta con una alta diversidad para realizar distintas cargas de trabajo, por ejemplo, aplicaciones por lotes, algoritmos iterativos, consultas interactivas y transmisión por secuencia [76]. *Spark* es altamente accesible y consta de bibliotecas muy enriquecidas, ofreciendo un desarrollo en *APIs* como *Python*, *Java*, *Scala* y *R*. Además, es compatible con cualquier fuente de datos de *Hadoop* (*HDFS* por sus siglas en inglés, *Hadoop Distributed File System*). *Spark* se basa en una abstracción de datos denominada *Resilient Distributed Datasets (RDDs)*. Los *RDDs* son una representación de colecciones distribuidas y son pieza fundamental de esta plataforma [4].

Modelo en paralelo de *Apache Spark*

El proceso inicia con el *Spark Driver* (nodo maestro), este se encarga de orquestar el funcionamiento distribuido en el *cluster*. Cada uno de los procesos que el usuario quiera paralelizar necesitan estar representados por la estructura de datos *RDD* (colección de objetos distribuidos e inmutables) y de esta manera el *Spark Driver* los puede almacenar en los *Spark Executor* (nodos esclavos). Entonces, un *RDD* está constituido por un número finito de particiones distribuido por los nodos esclavo y pueden ser (pero no necesariamente) computados en diferentes nodos de un sistema distribuido [4] como se ilustra en la figura 4.2. Además los *RDD* pueden ser almacenados en la memoria *RAM* durante todo el ciclo de vida del programa para un acceso más rápido.

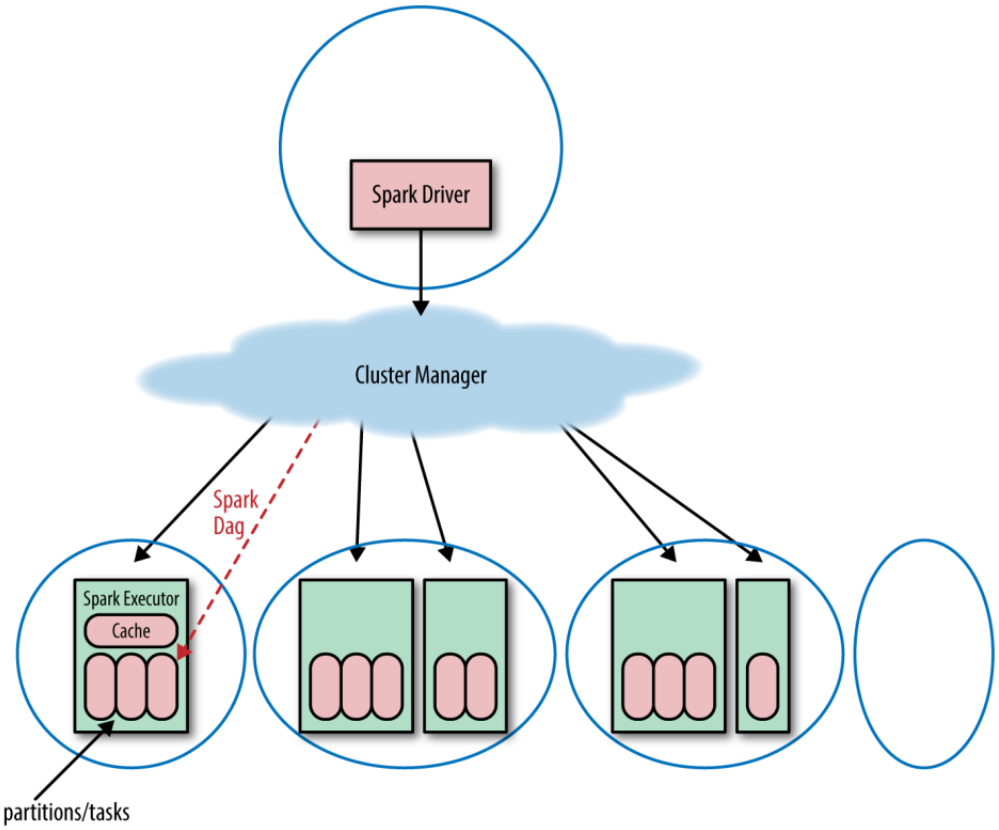


Figura 4.2: Inicio de una aplicación de *Spark* en un sistema distribuido, imagen tomada de [4]

Cuando un *RDD* es definido en una aplicación estos se vuelven inmutables, por lo cual la transformación de un *RDD* crea un nuevo *RDD*. Este paradigma de evaluación *lazy* y almacenamiento en memoria e inmutabilidad, permite que *Spark* sea fácil de usar, tolerante a fallos, escalable y eficiente [76].

Transformaciones *vs* Acciones

Apache Spark tiene dos tipos de funciones definidas en un *RDD*: acciones y transformaciones. Las **acciones** son funciones que retornan algo que no es un *RDD* y las **transformaciones** son funciones que devuelven otro *RDD*, a continuación en la tabla 4.2 se muestran algunas acciones y transformaciones de la documentación proporcionada por *Apache Spark*: [2]

[2] Documentación *Apache Spark* [Online] disponible: https://spark.apache.org/docs/latest/api/scala/

Tabla 4.2: Acciones y transformaciones en un *RDD*

Acciones	**Transformaciones**
collect: regresa todos los elementos de un conjunto de datos como un *array*	***join:*** se utiliza con dos *RDDs* de tipo (*K, V*) y (*K, W*), regresa un conjunto datos de (*K, (V, W)*) de todos los pares de elementos respecto a la llave
count: regresa el número de elementos en un conjunto de datos	***map:*** regresa un nuevo conjunto de datos distribuido pasando cada elemento a través de una función
first: regreso el primer elemento de un conjunto de datos	***filter:*** regresa un conjunto de datos formado por la selección de aquellos elementos que cumplen con una función
take: regresa un *array* con los primeros n elementos de un conjunto de datos	***sample:*** selecciona una fracción del conjunto de datos con o sin reemplazo, usando una semilla para generar números aleatorios
takeOrdered: regresa los primeros n elementos de un *RDD* usando un parámetro de ordenamiento	***union:*** regresa un nuevo conjunto de datos que contiene la unión de dos *RDD* respecto a una llave
countByKey: regresa un *Hasmap* de (*K, int*) de pares con el número de cada llave	***coalesce:*** decrece el número de particiones en un *RDD* para estructurarlo en un nuevo número de particiones
foreach: ejecuta una función en cada elemento del conjunto de datos	***distinct:*** regresa un conjunto de datos que contiene todos los elementos distintos

Apache Spark brinda una serie de paquetes de código abierto que permiten realizar distintas actividades, por ejemplo, *MLlib* es un conjunto de algoritmos del área de *Machine Learning* y estadística muy utilizados por la comunidad para la clasificación [4].

Por último, cuando *Apache Spark* fue creado su adversario a vencer era *Hadoop*, en la tabla 4.3 se ilustra una comparación de ambas metodologías, esto se puede observar en *databricks*[3]

[3] Databricks [Online] disponible: http://databricks.com/blog/2014/10/10/spark-petabyte-sort.html

Tabla 4.3: Comparación del desempeño de *Apache Hadoop vs Apache Spark*

Métrica	Record mundial Hadoop	Spark
Tamaño de conjunto de datos	102.5 TB	100 TB
Tiempo transcurrido	72 minutos	23 minutos
Número de Nodos	2100	206
Número de Cores	50400	6592
Ratio	1.42 TB/min	4.27 TB/min
Ratio/Nodo	0.67 GB/min	20.7 GB/min

Es evidente que *Apache Spark* es un *framework* muy poderoso para distintos dominios de aplicación, por lo tanto, la versión escalable del algoritmo *Bagging-RandomMiner* propuesto en esta investigación se desarrolló en esta plataforma.

En este capitulo, se pretende equiparar el algoritmo de *Bagging-RandomMiner* en el escenario de Detección de riesgos personales en dos enfoques, local (ejecución en un solo hilo) y distribuido (ejecución en múltiples hilos). Para realizar una comparación equitativa con el método más novedoso en este escenario (*OCKRA)*, fue necesario diseñar una versión distribuida de ambos algoritmos. A continuación se presentan las dos metodologías en el *framework Apache Spark* con el enfoque de *MapReduce*.

4.3. Implementación de *Bagging-RandomMiner* en *Apache Spark*

La implementación de *Bagging-RandomMiner* bajo la perspectiva de *MapReduce* propuesta en esta sección es una versión exacta del algoritmo diseñado en la sección 3.3.
Bagging-RandomMiner está compuesto por dos fases: aprendizaje y clasificación. En la fase de aprendizaje, se selecciona de manera aleatoria un porcentaje *RS* de los datos. A partir de esta muestra, se realiza otro sub-muestreo para seleccionar aquellos objetos que representarán la distribución completa de los objetos originales, los cuales son llamados Objetos Más Representativos de la población (*MRO*). A partir de aquí, se calcula un umbral de decisión (δ) promediando la matriz de

distancia de los *MRO* con la ayuda de la función *mapPartitions*. Esto se hace iterativamente para un número T de clasificadores. En la fase de clasificación, debido a que los *RDD* se distribuyen aleatoriamente de forma natural, la función *zipWithIndex* se aplica para agregar un índice a cada instancia. Luego, cada instancia de test se compara con los *MRO*, eligiendo solo el que tenga la distancia más cercana a la instancia de prueba. Finalmente, la similitud se calcula con la distancia más cercana y el umbral (δ), esto corresponde a la probabilidad del comportamiento genuino. Este proceso se repite hasta que se completen los T clasificadores y mediante la función *join*, se almacena el voto de cada clasificador, lo que nos permite promediar un resultado final. La figura 4.3 muestra un diagrama del funcionamiento de ambas fases.

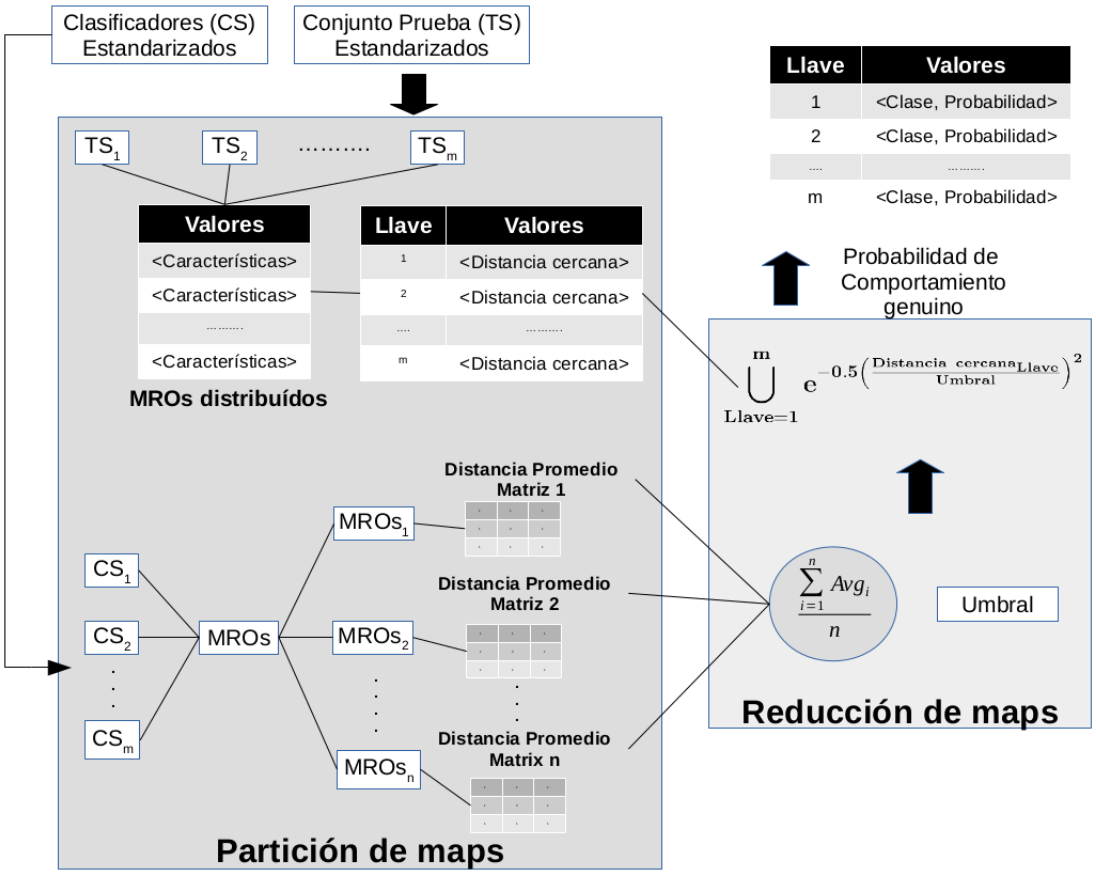


Figura 4.3: Diagrama de aprendizaje y predicción de *Bagging-RandomMiner* en la metodología *MapReduce*

4.3.1. Fase de aprendizaje

El algoritmo 4.3 es el encargado de administrar el funcionamiento de *Bagging-RandomMiner*, este algoritmo recibe como entrada los parámetros de *dataTrain* (conjunto de entrenamiento), *dataTest* (conjunto de prueba) ambos de naturaleza *RDD* y *LabeledPoint*, el número de clasificadores *T* en el ensamble, el porcentaje de objetos para cada clasificador (*RS*), el porcentaje de Objetos Más Representativos ($MRO_{percent}$) , el tipo de distancia a utilizar (*disType*) y el número de particiones (*maps*).

Se crea un ciclo para construir cada uno de los clasificadores (paso 1). En el paso 2, un clasificador se entrena aplicando el algoritmo 4.4, pasando los parámetros de *dataTrain*, *RS* y $MRO_{percent}$ como argumentos. En el paso 3, se obtiene una predicción del conjunto de prueba aplicando el algoritmo 4.5, pasando como parámetros el conjunto de prueba (*dataTest*) y el modelo entrenado (*model*). Luego, cada vez que se hace una predicción del conjunto de prueba, la función de *union* se usa para almacenar las predicciones de cada clasificador (paso 4). Finalmente, en el paso 6, se promedian los resultados del clasificador para obtener la probabilidad de pertenencia al comportamiento genuino de cada instancia de prueba, y se devuelve en una variable con estructura *RDD* (índice, probabilidad, clase real).

El algoritmo 4.4 es la parte ángular del aprendizaje, es aquí donde se obtienen los *MROs* y el umbral de decisión δ. Los parámetros de entrada son: *dataTrain*, *RS*, $MRO_{percent}$, *disType* y *maps*. Incialmente, se determinan los objetos del primer clasificador del conjunto de entrenamiento, esto se logra seleccionando aleatoriamente un porcentaje *RS* del *dataTrain* (paso 1). Después, el paso 2 selecciona los *MROs* del clasificador construido respecto al porcentaje $MRO_{percent}$. Ahora, para determinar el umbral de decisión (δ) es necesario calcular la matriz de distancias, con la ayuda de la función *mapPartitions*, la matriz de distancias se calcula para cada nodo, es decir, en paralelo cada nodo es responsable de calcular el promedio de la matriz de distancias con los *MRO* correspondientes. Al final, cada uno de los promedios se recopila y se calcula el promedio de estos resultados, que es el umbral de decisión δ (pasos 5-10). Finalmente, se crea un modelo y se devuelven los *MRO*, el umbral calculado δ y el tipo de distancia utilizada (paso 13).

Algoritmo 4.3 Clase principal *Bagging-RandomMiner* en *Apache Spark*

Entrada: *dataTrain*: *RDD* de tipo *LabeledPoint* (característica, clase); *prueba*: *RDD* de tipo *LabeledPoint* (característica, clase); *T:* número de clasificadores; *RS:* porcentaje de muestreo; $MRO_{percent}$: porcentaje de *MROs*; *disType:* tipo de distancia; *maps:* número de particiones.

Salida: probabilidad de pertenencia al comportamiento genuino.

1: **para** $i = 0$ `Hasta` T **hacer**
2: *model* ← Aprendizaje(*dataTrain*, *RS*, $MRO_{percent}$, *disType*)
3: *predicTemporal* ← Predicción(*dataTest*, *model*)
4: *predicciones.join*(*predicTemporal*)
5: **fin para**
6: **devolver** Promedio(*predicciones*)

Algoritmo 4.4 Aprendizaje *Bagging-RandomMiner* en *Apache Spark*

Entrada: *dataTrain*: *RDD* de tipo *LabeledPoint* (característica, clase); *dataTest*: *RDD* de tipo *LabeledPoint* (característica, clase); *T:* número de clasificadores; *RS:* porcentaje de muestreo; $MRO_{percent}$: porcentaje de *MROs*; *disType:* tipo de distancia; *maps:* número de particiones.

Salida: el modelo entrenado, un objeto de clase *RandomMiner*

1: $dataTrain'$ ← Seleccionar(*dataTrain*, *RS*)
2: *MROs* ← Seleccionar($dataTrain'$, MRO_{percen})
3: **mapPartition** $l \in MROs$ **hacer**
4: *distance* ← 0
5: **para** $i = 0$ `Hasta tamaño(l)` **hacer**
6: **para** $j = i + 1$ `Hasta tamaño(l)` **hacer**
7: *distance* ← *distance* + CalcularDistancia(*l(i), l(j), disType)*
8: **fin para**
9: **fin para**
10: *umbral*.añadir(promedio(*distance*)) {Promedio de cada partición}
11: **fin mapPartition**
12: δ ← Promedio(*umbral*) {Promedio global de las particiones}
13: **devolver** (*MROs*, δ, *disType*)

4.3.2. Fase de clasificación

Para clasificar un nuevo objeto se recurre al algoritmo 4.5, éste recibe como parámetros de entrada *dataTest* y el modelo entrenado por el algoritmo 4.4. Para poder identificar cada elemento en el conjunto de prueba se aplica la función *zipWithIndex* (paso 1) que asigna un índice a cada instancia para no perder el orden y poder aplicar la función de *union* en el algoritmo 4.3. Después, con la ayuda de la función *mapPartitions*, la distancia de cada objeto de prueba se calcula con todos los *MRO* distribuidos en cada nodo. Al final, se elige la distancia más cercana (*dmin*) al objeto de prueba (paso 7) y la medida de similitud se calcula con la *dmin* obtenida y el umbral de decisión δ (paso 8). Cuando todos los nodos terminan sus respectivas ejecuciones, el algoritmo 4.5 devuelve un *RDD* (índice, probabilidad) (paso 11).

Algoritmo 4.5 Predicción *Bagging-RandomMiner* en *Apache Spark*

Entrada: *dataTest*: *RDD* de tipo *LabeledPoint* (característica, clase); *model:* modelo entrenado (*MRO*, δ, *disType*; *maps:* número de particiones);

Salida: probabilidad de pertenencia al comportamiento genuino

1: *testIndex* ← zipWithIndex(*dataTest*)
2: **mapPartition** $l \in$ `testIndex` **hacer**
3: **para** $i = 0$ `Hasta tamaño(l)` **hacer**
4: **para** $j = i + 1$ `Hasta tamaño(MROs)` **hacer**
5: *distance*(j) ← CalcularDistancia(*l*(i), *MROs*(j), *disType*)
6: **fin para**
7: *dmin* ← min(*distance*) {se determina la distancia mínima a los *MROs*}
8: *similaridad* ← $e^{-0.5(dmin/\delta)^2}$
9: **fin para**
10: **fin mapPartition**
11: **devolver** *similaridad* de tipo *RDD*

4.4. Implementación de One-Class K-means with Randomly-projected features Algorithm (OCKRA) en *Apache Spark*

En la librería de *MLlib* se encuentra una versión aproximada de *k-Means++*, por lo cual, no fue necesario diseñar una versión de dicho algoritmo, esto implica que la versión distribuida de *OCKRA* es aproximada. Análogo a *Bagging-RandomMiner*, *OCKRA* consta de dos fases para su ejecución, aprendizaje y clasificación. A continuación se describen a detalle cada una de ellas.

4.4.1. Fase de aprendizaje

En esta fase el algoritmo 4.6 se encarga de orquestar el funcionamiento de *OCKRA*, los parámetros de entrada son: *dataTrain* (el conjunto de entrenamiento en formato *RDD* y *LabeledPoint*), *dataTest* (conjunto de prueba en formato *RDD* y *LabeledPoint*), T (número de clasificadores en el ensamble), K (número de centroides a considerar en cada clasificador) y *maps* (número de particiones para dividir). El proceso inicia en el paso 1 con una selección aleatoria de características del conjunto de entrenamiento, después, se realiza una proyección seleccionado explícitamente las características obtenidas (paso 3). En el paso 4 se invoca el algoritmo de *k-Means++* implementado en la librería de *MLlib* para encontrar los K centroides pertinentes, cabe mencionar que esta parte del algoritmo se realiza en paralelo. Al tener identificados los centroides del conjunto $dataTrain'$ se procede a calcular la distancia promedio entre cada par de objetos del conjunto $dataTrain'$ para definir δ y posteriormente definir el modelo como una tripleta (*centroides*, *características*, δ) (pasos 5 y 6).

Algoritmo 4.6 Fase de aprendizaje *OCKRA* en *Apache Spark*

Entrada: *dataTrain*: *RDD* de tipo *LabeledPoint* (característica, clase); *dataTest*: *RDD* de tipo *LabeledPoint* (característica, clase); *T:* número de clasificadores; *K:* número de centroides; *maps:* número de particiones.

Salida: probabilidad de pertenencia al comportamiento genuino.

1: **para** $i = 0$ `Hasta` T **hacer**
2: *caracteristicas* ← SeleccionAleatoriaDeCaracteristicas(*dataTrain*)
3: $dataTrain'$ ← Proyección(*dataTrain*, *caracteristicas*)
4: *centroides* ← *k-Means++*($dataTrain'$, *K*)
5: δ ← DistanciaPromedio($dataTrain'$)
6: *model(i)* ← (*centroides*, *características*, δ)
7: **fin para**
8: **devolver** *model*

4.4.2. Fase de clasificación

En el algoritmo 4.7 se lleva a cabo la clasificación del conjunto de prueba (*dataTest*). Los parámetros requeridos para su funcionamiento son: *dataTest* (conjunto de prueba en estructura *RDD* y *LabeledPoint*), *T* (el número de clasificadores en el ensamble) y *model* (el modelo entrenado por el algoritmo 4.6). El primer paso es aplicar la transformación *zipWithIndex* para identificar cada elemento del conjunto de prueba mediante una llave. En el paso 3 se aplica la función *mapPartition*, esto permitirá trabajar la siguiente parte del algoritmo en cada bloque (nodo) de forma paralela. Cada nodo en particular debe computar la distancia de cada objeto de prueba a cada centroide almacenado en la variable *model* (pasos 4-6). Después, se determina la distancia mínima de este conjunto y se calcula la similaridad (paso 9 y 10) que será almacenada en la variable *predicciones* con ayuda de la función *join*. Cuando cada partición haya terminado su parte correspondiente, los resultados de cada clasificador se estarán almacenando en la variable *probabilidad* mediante la función *union* con su respectiva llave (paso 13). Finalmente, cuando se ejecutan los *T* clasificadores indicados, se calcula el promedio de todos los votos almacenados en *probabilidad* para retornar la posibilidad de pertenencia al comportamiento genuino mediante un *RDD* de tipo (*llave*, *probabilidad*).

Algoritmo 4.7 Fase de clasificación *OCKRA* en *Apache Spark*

Entrada: *dataTest*: *RDD* de tipo *LabeledPoint* (característica, clase); *T:* número de clasificadores; *model:* modelo entrenado; *maps:* número de particiones.

Salida: probabilidad de pertenencia al comportamiento genuino

1: *testIndex* ← zipWithIndex(*dataTest*)
2: **para** $i = 0$ `Hasta` T **hacer**
3: **mapPartition** $l \in$ *testIndex* **hacer**
4: **para** $j = 0$ `Hasta tamaño(l)` **hacer**
5: **para** $k = j + 1$ `Hasta tamaño(`*model(j).centroides*`)` **hacer**
6: *distance*(k) ← CalcularDistancia(*l*(j), *centroides*(j, k))
7: **fin para**
8: **fin para**
9: *dmin* ← min(*distance*) {se determina la distancia mínima a los *centroides*}
10: *similaridad* ← $e^{-0.5(dmin/model(i).\delta)^2}$
11: *predicciones.join*(*similaridad*) {Almacena la similaridad de cada objeto de prueba}
12: **fin mapPartition**
13: *probabilidad*.union(*predicciones*) {Almacena los votos de cada clasificador según su llave}
14: **fin para**
15: **devolver** Promedio(*probabilidad*) {*RDD* de tipo (*llave*, *probabilidad*)}

4.5. Experimentos y resultados sobre *PRIDE*

Anteriormente, se mencionó que el conjunto de datos seleccionado para medir la detección de riesgos personales es *PRIDE* (descrito en la sección 2.3.2). Además, el algoritmo más novedoso para esta tarea fue diseñado por Rodriguez et al. [35] denominado *OCKRA*. En esta sección se realiza una comparativa entre *Bagging-RandomMiner* el algoritmo diseñado en esta investigación y *OCKRA* desde dos perspectivas, local y distribuido.

4.5.1. Métricas de evaluación

Análogo al escenario 1 (sección 3.5.1) se utilizó la métrica de área bajo la curva (*AUC*) para medir la eficiencia de los algoritmos, en este caso, se omitió la medición de *ZFP* (*Zero False Positive*) debido a que los autores de *OCKRA* no reportan esta medición.

Prueba no-paramétrica *Bayesian Signed Rank*

Para el análisis estadístico se optó por implementar la prueba no-paramétrica *Bayesian Signed Rank* a un nivel de confianza del 95 %. Esta prueba a diferencia de los análisis frecuentistas (*t-test*) nos brinda información a detalle de las diferencias entre las dos metodologías [77]. Se basa en el Proceso de *Dirichlet* (*DP*) [78], visto como una distribución de probabilidad que se realiza en dos pasos [79]:

1. Obtenemos la función de densidad de probabilidad posterior, esta se determina mediante una combinación lineal de los deltas de *Dirac* centrados en las observaciones, cuyos pesos provienen de la distribución de *Dirichlet*.

2. Se calcula la probabilidad de pertenencia del parámetro a cada región de interés, estas se definen como:

   - θ_L: región de puntos donde el algoritmo 1 es superior al 2.
   - θ_R: región de puntos donde el algoritmo 2 es superior al 1.
   - θ_r: región de puntos donde ambos algoritmos son iguales.

Una forma de visualizar las probabilidades posteriores es calcular los pesos de *Monte Carlo* que son aproximadamente igual al Proceso de *Dirichlet* y representarlos en coordenadas baricéntricas [77]. La representación gráfica se muestra en la figura 4.4, se puede observar que los puntos que se encuentren en la parte superior indicarán que los dos algoritmos a evaluar son iguales estadísticamente. Aquellas observaciones posicionadas en el lado izquierdo asemejan la superioridad del algoritmo 1 (A1) sobre el algoritmo 2 (A2) y por último, la región del lado derecho indicará que el A1 es inferior al A2. Evidentemente, la región con más puntos definirá la resolución de la prueba estadística.

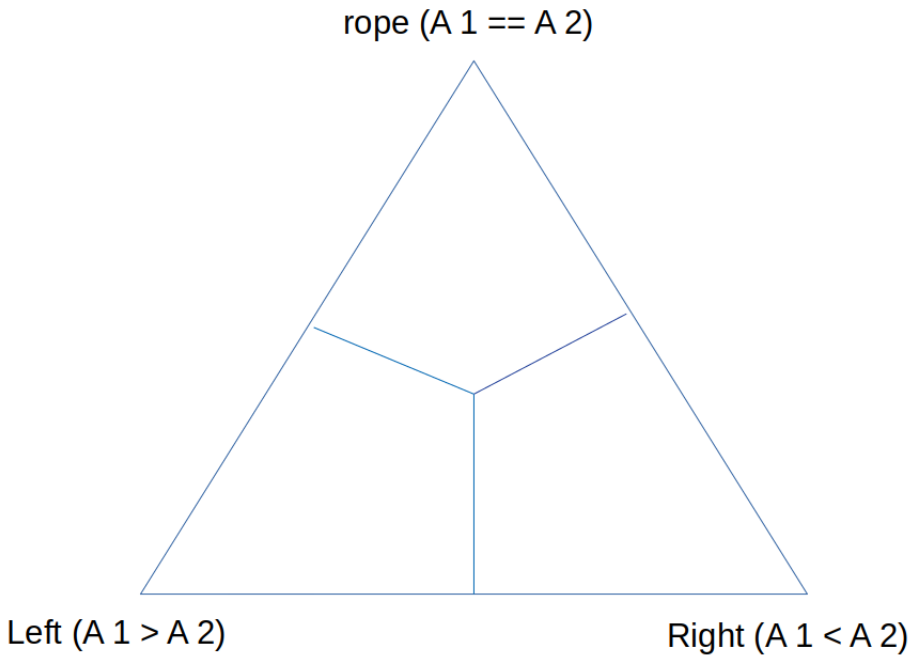


Figura 4.4: Muestras de *Monte Carlo* en coordenadas baricéntricas, para representar la prueba estadística no-paramétrica *Bayesian Signed Rank*.

4.5.2. Resultados

Analizar los resultados obtenidos por ambos algoritmos en el segundo escenario es de vital importancia, esto permitirá definir las ventajas y desventajas de cada uno de los métodos. Para las ejecuciones en local, se utilizó un equipo de cómputo que consiste de 16Gb de RAM, un procesador *Intel Core i7* de sexta generación y un sistema operativo de 64 bits, en el aspecto distribuido se ejecutaron en un *cluster* de 14 nodos administrados por un nodo maestro, todos los nodos tienen el mismo *hardware* y *software*. Respecto al *hardware*, cada nodo tiene dos procesadores *Intel Xeon E5-2620*, 6 núcleos (12 hilos) por procesador, 2Ghz y 64Gb de RAM. El sistema operativo es *Cent OS 6.5*, con *Apache Spark* y *MLlib 2.2.0*, 336 núcleos (24 núcleos/nodo), 728 GB de RAM (52 GB/nodo). A continuación en la tabla 4.4 se especifica la configuración de ambos algoritmos. Los parámetros establecidos para *Bagging-RandomMiner* se obtuvieron de manera empírica, probando distintos porcentajes para la construcción del clasificador y la selección de *MROs*, concluyendo que el 1 % en ambos casos era la ruta más eficiente. Para identificar el número de *maps* más adecuado al igual que el tipo de distancia, se aplicó un *grid search*.

En el caso de *OCKRA*, los parámetros utilizados fueron los propuestos por su versión original en [35], ya que los autores mencionan haber realizado un análisis empírico para encontrar dichos valores, sin embargo, en la versión distribuida fue necesario determinar el número correcto de *maps*.

Tabla 4.4: Configuración de los algoritmos *Bagging-RandomMiner* y *OCKRA* en ambos enfoques, local y distribuido.

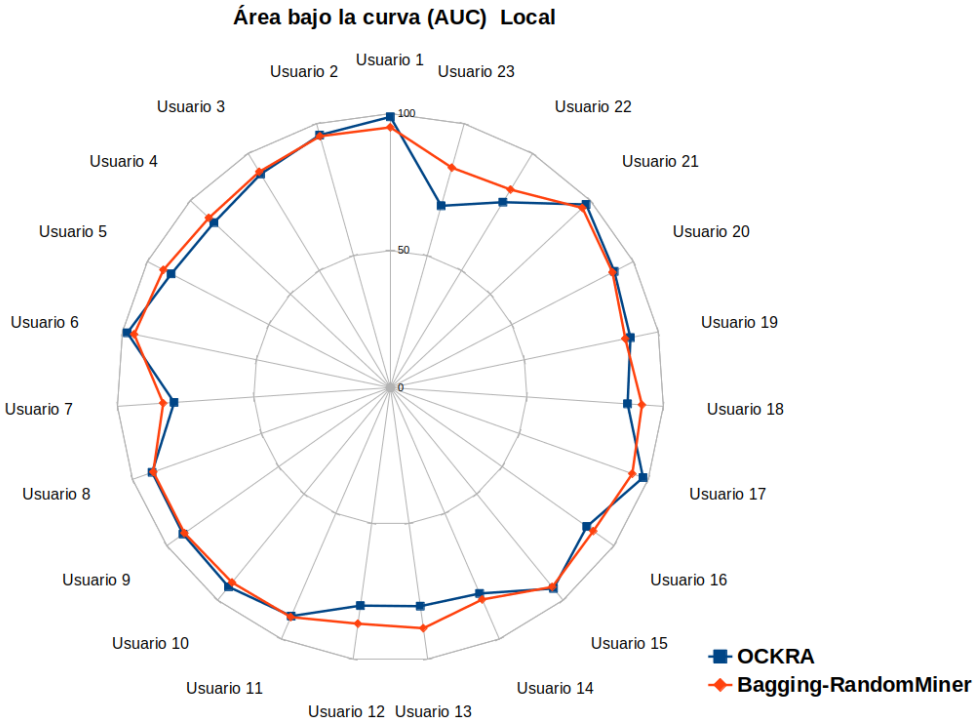
Bagging-RandomMiner	**OCKRA**
Distribuido	
T = 50 (número de clasificadores)	***T*** = 100 (número de clasificadores)
RS = 1 % (porcentaje de muestreo)	K = 10 (número de centroides por clasificador)
$\mathrm{MRO}_{\mathbf{perecnt}}$ = 1 % (porcentaje de MROs)	***disType*** = Euclídea (tipo de distancia)
disType = Chebyshev (tipo de distancia)	***maps*** = 64 (número de particiones)
maps = 256 (número de particiones)	
Local	
T = 50 (número de clasificadores)	***T*** = 100 (número de clasificadores)
RS = 1 % (porcentaje de muestreo)	K = 10 (número de centroides por clasificador)
$\mathrm{MRO}_{\mathbf{perecnt}}$ = 1 % (porcentaje de MROs)	***disType*** = Euclídea (tipo de distancia)
disType = Chebyshev (tipo de distancia)	

En la tabla 4.5 se muestran los valores obtenidos para ambos clasificadores en su versión local y distribuida respecto a la métrica de *AUC*, se resalta en negritas aquellos puntajes donde un método es superior a otro. En la figura 4.5 se ilustra el comportamiento de los algoritmos en una gráfica de radar, esto facilitará determinar en cuales usuarios es superior cada método.

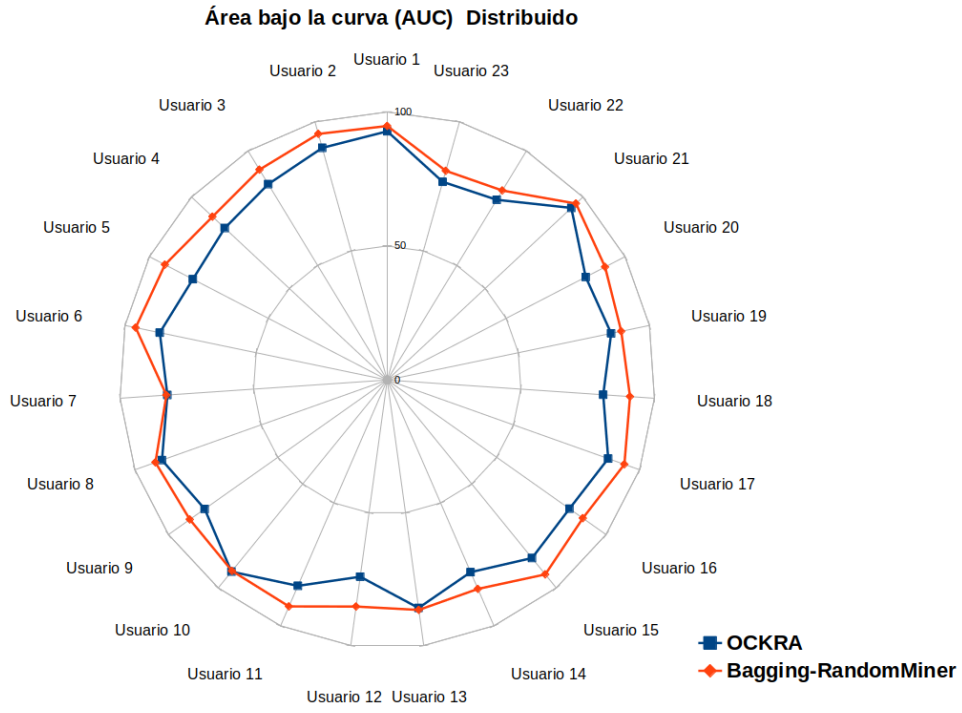
A simple vista se puede observar que *Bagging-RandomMiner* es superior en un promedio general, donde la versión local de *OCKRA* alcanza un 89.1 % y *Bagging-RandomMiner* un 90.6 %. En el enfoque distribuido *Bagging-RandomMiner* obtiene un 90.2 % y *OCKRA* un 84.5 %, al parecer la versión distribuida de *Bagging-RandomMiner* no se benefició de la metodología *MapReduce* más allá de la escalabilidad del método, por otro lado, una posible hipótesis por la cual *OCKRA* bajó su rendimiento, es que la versión distribuida utiliza una aproximación del algoritmo *k-Means++*.

Tabla 4.5: Comparación de los algoritmos *Bagging-RandomMiner* y *OCKRA* en la métrica de *AUC* en el conjunto de datos *PRIDE* en ambos enfoques, local y distribuido.

Usuario	**Distribuida**		**Local**	
	OCKRA	***Bagging-RandomMiner***	***OCKRA***	***Bagging-RandomMiner***
1	92.8	**94.8**	**98.8**	94.98
2	89.9	**95.3**	**95.7**	95.27
3	85.5	**91.8**	91.2	**92.19**
4	83.0	**89.3**	88.2	**90.73**
5	81.8	**93.5**	90.2	**93.41**
6	86.8	**96.0**	**98.2**	95.62
7	82.3	**82.7**	79.2	**83.21**
8	89.3	**91.9**	**92.4**	91.95
9	83.5	**90.2**	**92.7**	92.09
10	**92.2**	91.8	**93.7**	91.67
11	83.7	**92.1**	90.9	**91.29**
12	74.1	**85.3**	80.3	**86.96**
13	85.9	**86.6**	80.5	**88.62**
14	78.2	**85.0**	81.9	**84.3**
15	85.6	**93.5**	**94.5**	93.77
16	83.3	**89.4**	87.9	**90.79**
17	87.6	**94.0**	**98**	93.84
18	80.8	**90.9**	86.9	**92.18**
19	85.4	**89.3**	**89.6**	87.7
20	83.5	**91.6**	**92.2**	91.45
21	94.1	**96.4**	**97.9**	96.05
22	78.7	**82.7**	79.2	**84.58**
23	76.8	**81.1**	68.9	**83.35**
Average	84.5	**90.2**	89.1	**90.6**



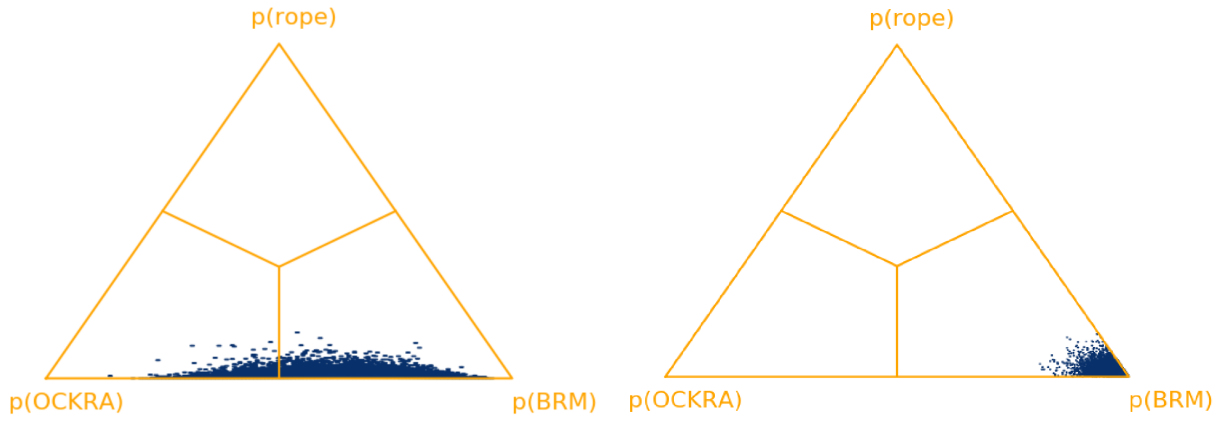
(a) Local



(b) Distribuido

Figura 4.5: La figura (a) muestra el desempeño del área bajo la curva en el enfoque local y la figura (b) representa el desempeño del *AUC* en el enfoque distribuido

Anteriormente, se había mencionado que la prueba estadística para comparar los algoritmos sería la prueba no-paramétrica *Bayesian Signed Rank*. En la figura 4.6 se muestran los diagramas de las coordenadas baricéntricas de ambas comparaciones, local y distribuido.



(a) Local (b) Distribuido

Figura 4.6: Se ilustran los resultados de las pruebas no-paramétricas *Bayesian Signed Rank* de *OCKRA* vs *Bagging-RandomMiner*. La figura (a) muestra la comparación del enfoque local y la figura (b) ilustra la comparación del enfoque distribuido

En el caso del enfoque local, se obtuvieron las siguientes probabilidades:

- P(*OCKRA* > *Bagging-RandomMiner*) = 33.3 %
- P(*OCKRA* < *Bagging-RandomMiner*) = 62.3 %
- P(*OCKRA* = *Bagging-RandomMiner*) = 4.6 %

Por otro lado, en el enfoque distribuido se determinó lo siguiente:

- P(*OCKRA* > *Bagging-RandomMiner*) = 2.1 %
- P(*OCKRA* < *Bagging-RandomMiner*) = 94.7 %
- P(*OCKRA* = *Bagging-RandomMiner*) = 3.2 %

Después de analizar los resultados podemos concluir algunas cuestiones. En el enfoque local se demostró que *Bagging-RandomMiner* es superior a *OCKRA* en la métrica de área bajo la curva. Otro punto a favor es la complejidad computacional, *Bagging-RandomMiner* demostró ser más rápido, alcanzando un tiempo promedio de ejecución de 739.3 segundos por usuario (por cada *folder* del *5-fold cross validation*) mientras *OCKRA* obtuvo un promedio de 5486.9 segundos.

En el enfoque distribuido, *Bagging-RandomMiner* volvió a superar a *OCKRA* muy significativamente en la métrica de *AUC*, debido a que la versión de *OCKRA* diseñada en el *framework Apache Spark* utilizó un método aproximado de *k-Means++*. En el aspecto de la complejidad computacional se puede observar la escalabilidad de los métodos, donde *OCKRA* obtiene 575.6 segundos y *Bagging-RandomMiner* 147 segundos en promedio.

Al resumir los resultados podemos concluir que en la detección de riesgos personales mediante el procesamiento de señales de bio-sensado, *Bagging-RandomMiner* el algoritmo para la clasificación de una clase propuesto en este trabajo de investigación es el método más eficiente para dicha tarea.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones

Este trabajo de investigación cumple con los objetivos propuestos. En ellos se pretendía diseñar un algoritmo competitivo para la detección de anomalías respecto a los escenarios de Detección de intrusos en sesiones de computadora 2.3.1 y la Detección de riesgos personales mediante el procesamiento de señales de bio-sensado 2.3.2. Al analizar los resultados obtenidos en los capítulos 3 y 4 podemos afirmar que los objetivos fueron alcanzados satisfactoriamente. A continuación, se observan las ventajas y debilidades del algoritmo:

Ventajas

- Posee un alto desempeño de predicción en los dominios de Detección de intrusos en sesiones de computadora y Detección de riesgos personales mediante el procesamiento de señales de bio-sensado.
- Debido a su complejidad computacional semi-cuadrática, es una opción altamente recomendaba para abordar escenarios medianamente grandes.
- La versión diseñada en la metodología *MapReduce* es escalable y permite abordar escenarios de Detección de anomalías con alta dimensionalidad.
- Obtiene excelentes resultados al construir ensambles con un número pequeño de clasificado-

res (10) y esto mejora sustancialmente el tiempo de ejecución.

Debilidades

- El desempeño del algoritmo se relaciona directamente con los porcentajes de selección ($RS_{Percent}$ y $MRO_{percent}$), estos dependerán de las características del problema a clasificar.
- Al visualizar los resultados del apéndice A, podemos concluir que *Bagging-RandomMiner* es un algoritmo que se comporta muy estable en escenarios que fueron construidos con la naturaleza de Detección de anomalías, ya que los conjuntos de datos propuestos en este apéndice fueron modificados para considerarse des-balanceados.

5.2. Trabajo futuro

Después de analizar los resultados obtenidos en esta tesis de investigación surgieron algunas preguntas que serían interesantes de indagar. *Bagging-RandomMiner* mostró ser el algoritmo con el mejor desempeño en los escenarios propuestos, sin embargo, al compararlo en otros dominios de aplicación de naturaleza des-balanceada, se posicionó en los primeros lugares (tercer lugar) sin presentar alguna diferencia significativa con los algoritmos del estado del arte, pero sería interesante hacer una análisis a detalle y encontrar las incógnitas de estos resultados para mejorar el algoritmo, es decir, que características tienen estas bases de datos que resultan un problema para *Bagging-RandomMiner*. También, sería posible definir todos aquellos dominios de aplicación en los cuales *Bagging-RandomMiner* es altamente recomendable, en ambos enfoques local y distribuido.

Capítulo 6

Publicaciones ligadas a la tesis

6.1. Revista

- **Title:** *Bagging-RandomMiner: a one-class classifier for file access-based masquerade detection*
- **Authors:** José Benito Camiña, Miguel Angel Medina Pérez, Raúl Monroy, Octavio Loyola González, Luis Ángel Pereyra Villanueva y Luis Carlo González Gurrola.
- **Year:** 2018
- **Month:** Jul
- **Journal:** Machine Vision and Applications
- **issn:** 1432-1769
- **doi:** 10.1007/s00138-018-0957-4
- **url:** https://doi.org/10.1007/s00138-018-0957-4

Abstract: Dependence on personal computers has required the development of security mechanisms to protect the information stored in these devices. There have been different approaches to profile user behavior to protect information from a masquerade attack; one such recent approach is based on user file-access patterns. In this paper, we propose a novel classification ensemble for file

access-based masquerade detection. We have successfully validated the hypothesis that a one-class classification approach to file access-based masquerade detection outperforms a multi-class one. In particular, our proposed one-class classifier significantly outperforms several state-of-the-art multi-class classifiers. Our results indicate that one-class classification attains better classification results, even when unknown attacks arise. Additionally, we introduce three new repositories of datasets for the identification of the three main types of attacks reported in the literature, where each training dataset contains no object belonging to the type of attack to be identified. These repositories can be used for testing future classifiers, simulating attacks carried out in a real scenario.

6.2. Congreso

- **Title:** *Bagging-RandomMiner - Un Algoritmo en MapReduce para Deteccion de Anomalías en Big Data*
- **Authors:** Luis Pereyra, Diego García Gil, Francisco Herrera, Luis C. González Gurrola, Jacinto Carrasco, Miguel Angel Medina Perez y Raúl Monroy
- **Year:** 2018
- **Month:** Oct
- **Proceedings:** 18th Conference of the Spanish Association for Artificial Intelligence
- **url:** https://sci2s.ugr.es/caepia18/proceedings/docs/CAEPIA2018_paper_190.pdf

Abstract: La detección de anomalías es una tarea de interés en muchas aplicaciones del mundo real cuyo objetivo es identificar el comportamiento irregular de un proceso basado en el aprendizaje de como este se comporta normalmente. En este trabajo, presentamos una versión *MapReduce* de una estrategia que ha demostrado ser exitosa para encontrar valores atípicos (anomalías) en conjuntos de datos de tamaño mediano. Esta estrategia, conocida como *Bagging-RandomMiner*, ahora ha sido extendida para aprovechar las bondades que *Apache Spark* ofrece y poder demostrar su utilidad en contextos de *Big Data*. Para evaluar su rendimiento, se ha utilizado el conjunto de

datos *PRIDE* como *Benchmark* para identificar situaciones de estrés (anomalías) en un grupo de 23 sujetos. En este estudio comparamos Bagging-RandomMiner contra otra versión *MapReduce* del algoritmo que mejores resultados ha reportado sobre *PRIDE*, llamado *OCKRA*. Los experimentos indican que *Bagging-RandomMiner* supera claramente a *OCKRA* en al menos 6 % en el *score AUC*, mas aún su tiempo de ejecución es al menos un orden de magnitud menor. Estos resultados soportan la idea de *Bagging-RandomMiner* como un algoritmo contendiente en tareas de *Big Data*. En este trabajo hacemos disponible el código de esta versión.

Referencias

[1] B. Lantz, *Machine Learning With R*. Packt Publishing Ltd, 2013.

[2] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861 – 874, 2006. ROC Analysis in Pattern Recognition.

[3] J. S. Hurwitz, A. Nugent, F. Halper, and M. Kaufman, *Big Data For Dummies*. For Dummies, 2013.

[4] H. Karau and R. Warren, *High Performance Spark: Best Practices for Scaling and Optimizing Apache Spark*. O'Reilly Media, 2017.

[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, pp. 15:1–15:58, July 2009.

[6] J. B. Camiña, C. Hernández-Gracidas, R. Monroy, and L. Trejo, "The windows-users and -intruder simulations logs dataset (wuil): An experimental framework for masquerade detection mechanisms," *Expert Systems with Applications*, vol. 41, no. 3, pp. 919 – 930, 2014. Methods and Applications of Artificial and Computational Intelligence.

[7] K. Sk and R. Vadlamani, "Credit card fraud detection using big data analytics: Use of psoaann based one-class classification," pp. 1–8, 08 2016.

[8] A. Y. Barrera-Animas, L. A. Trejo, M. A. Medina-Pérez, R. Monroy, J. B. Camiña, and F. Godínez, "Online personal risk detection based on behavioural and physiological patterns," *Information Sciences*, vol. 384, pp. 281 – 297, 2017.

[9] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123–140, Aug 1996.

[10] L. Rokach, "Ensemble-based classifiers," *Artificial Intelligence Review*, vol. 33, pp. 1–39, Feb 2010.

[11] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, pp. 181–207, May 2003.

[12] G. Valentini and T. G. Dietterich, "Bias-variance analysis of support vector machines for the development of svm-based ensemble methods," *J. Mach. Learn. Res.*, vol. 5, pp. 725–775, Dec. 2004.

[13] S. S. Khan and M. G. Madden, "A survey of recent trends in one class classification," in *Artificial Intelligence and Cognitive Science* (L. Coyle and J. Freyne, eds.), (Berlin, Heidelberg), pp. 188–197, Springer Berlin Heidelberg, 2010.

[14] R. Fujimaki, T. Yairi, and K. Machida, "An approach to spacecraft anomaly detection problem using kernel feature space," in *Proceedings of the Eleventh ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, KDD '05, (New York, NY, USA), pp. 401–410, ACM, 2005.

[15] L. Nanni, E. Maiorana, A. Lumini, and P. Campisi, *On-line signature verification: Comparison and fusion of feature based and function based classifiers*, pp. 91–108. 01 2010.

[16] B. Andras, R. Busa-Fekete, and B. Kégl, "A one-class classification approach for protein sequences and structures," 04 2009.

[17] R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, E. V. Koonin, D. M. Krylov, R. Mazumder, S. L. Mekhedov, A. N. Nikolskaya, B. S. Rao, S. Smirnov, A. V. Sverdlov, S. Vasudevan, Y. I. Wolf, J. J. Yin, and D. A. Natale, "The cog database: an updated version includes eukaryotes," *BMC Bioinformatics*, vol. 4, p. 41, Sep 2003.

[18] "Insider threat detection and its future directions," *Int. J. Secur. Netw.*, vol. 12, pp. 168–187, Jan. 2017.

[19] M. Schonlau, W. Dumouchel, W. hua Ju, A. F. Karr, M. Theus, and Y. Vardi, "Computer intrusion: Detecting masquerades," 2001.

[20] R. A. Maxion, "Masquerade detection using enriched command lines," in *2003 International Conference on Dependable Systems and Networks, 2003. Proceedings.*, pp. 5–14, June 2003.

[21] R. A. Maxion and T. N. Townsend, "Masquerade detection using truncated command lines," in *Proceedings of the 2002 International Conference on Dependable Systems and Networks*, DSN '02, (Washington, DC, USA), pp. 219–228, IEEE Computer Society, 2002.

[22] K. Wang and S. J. Stolfo, "One-class training for masquerade detection," 2003.

[23] H.-S. Kim and S.-D. Cha, "Empirical evaluation of svm-based masquerade detection using unix commands," *Computers Security*, vol. 24, no. 2, pp. 160 – 168, 2005.

[24] Z. Jian, H. Shirai, I. Takahashi, J. Kuroiwa, T. Odaka, and H. Ogura, "Masquerade detection by boosting decision stumps using unix commands," *Computers Security*, vol. 26, no. 4, pp. 311 – 318, 2007.

[25] Y. Ding, P. Sun, X. Chen, and C. Liu, "Masquerade detection based on one class svm," 12 2008.

[26] M. Pusara and C. E. Brodley, "User re-authentication via mouse movements," in *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, VizSEC/DMSEC '04, (New York, NY, USA), pp. 1–8, ACM, 2004.

[27] Garg, Rahalkar, Upadhyaya, and Kwiat, "Profiling users in gui based systems for masquerade detection," in *2006 IEEE Information Assurance Workshop*, pp. 48–54, June 2006.

[28] C. Shen, Z. Cai, X. Guan, and R. Maxion, "Performance evaluation of anomaly-detection algorithms for mouse dynamics," *Computers Security*, vol. 45, pp. 156 – 171, 2014.

[29] K. Killourhy and R. Maxion, "Why did my detector do that?!," in *Recent Advances in Intrusion Detection* (S. Jha, R. Sommer, and C. Kreibich, eds.), (Berlin, Heidelberg), pp. 256–276, Springer Berlin Heidelberg, 2010.

[30] A. Messerman, T. Mustafić, S. A. Camtepe, and S. Albayrak, "Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics," in *2011 International Joint Conference on Biometrics (IJCB)*, pp. 1–8, Oct 2011.

[31] A. Morales, J. Fierrez, and J. Ortega-Garcia, "Towards predicting good users for biometric recognition based on keystroke dynamics," in *Computer Vision - ECCV 2014 Workshops* (L. Agapito, M. M. Bronstein, and C. Rother, eds.), (Cham), pp. 711–724, Springer International Publishing, 2015.

[32] B. Camiña, R. Monroy, L. A. Trejo, and E. Sánchez, "Towards building a masquerade detection method based on user file system navigation," in *Advances in Artificial Intelligence* (I. Batyrshin and G. Sidorov, eds.), (Berlin, Heidelberg), pp. 174–186, Springer Berlin Heidelberg, 2011.

[33] J. B. Camiña, J. Rodríguez, and R. Monroy, "Towards a masquerade detection system based on user's tasks," in *Research in Attacks, Intrusions and Defenses* (A. Stavrou, H. Bos, and G. Portokalidis, eds.), (Cham), pp. 447–465, Springer International Publishing, 2014.

[34] M. A. Medina-Pérez, R. Monroy, J. B. Camiña, and M. García-Borroto, "Bagging-tpminer: a classifier ensemble for masquerader detection based on typical objects," *Soft Computing*, vol. 21, pp. 557–569, Feb 2017.

[35] J. Rodríguez, A. Y. Barrera-Animas, L. A. Trejo, M. A. Medina-Pérez, and R. Monroy, "Ensemble of one-class classifiers for personal risk detection based on wearable sensor data," *Sensors*, vol. 16, no. 10, 2016.

[36] L. A. Trejo, E. Sánchez, R. Alonso, J. Vázquez, and J. Cardona, "Elisa: emergency, positioning, and inmediate assistance system," *Health and Environmental World Congress Converging Towards Sustainability*, p. 26, 2010.

[37] C. Jing, C. Wang, and C. Yan, "Replay attack: A prevalent pattern of fraudulent online transactions," in *2018 5th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/2018 4th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom)*, pp. 75–82, June 2018.

[38] F. Y. EDGEWORTH *Philosoph. Mag*, p. 364–375., 1887.

[39] T. C. Minter, "Single-class classification," *Symposium on Machine Processing of Remotely Sensed Data, IEEE*, pp. 12–15, 1975.

[40] D. Tax, "Tax. one-class classification; concept-learning in the absence of counter-examples," *ASCI Dissertation Series*, 01 2001.

[41] M. Bishop, "Neural networks for pattern recognition," *Oxford University Press, Walton Street, Oxford OX2 6DP*, 01 1995.

[42] N. Ullman, "Elementary statistics, an applied approach," Wiley and Sons, 1978.

[43] E. Parzen, "On estimation of a probability density function and mode," *Ann. Math. Statist.*, vol. 33, pp. 1065–1076, 09 1962.

[44] R. A. Maxion and T. N. Townsend, "Masquerade detection using truncated command lines," in *Proceedings International Conference on Dependable Systems and Networks*, pp. 219–228, June 2002.

[45] E. Alpaydin, *Introduction to Machine Learning*. The MIT Press, 2014.

[46] J. Platt, B. Schölkopf, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," p. 30, November 1999.

[47] D. Tax, *One-class classification*. PhD thesis, Def University, Applied Sciences, 06 2001.

[48] V. Di Gesù, G. Lo Bosco, and L. Pinello, "A one class knn for signal identification: a biological case study," *IJKESDP*, vol. 1, pp. 376–389, 01 2009.

[49] S. Agarwal and A. Sureka, "Using knn and svm based one-class classifier for detecting online radicalization on twitter," in *Distributed Computing and Internet Technology* (R. Natarajan, G. Barua, and M. R. Patra, eds.), (Cham), pp. 431–442, Springer International Publishing, 2015.

[50] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLOS ONE*, vol. 11, pp. 1–31, 04 2016.

[51] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[52] D. Arthur and S. Vassilvitskii, "K-means++: The advantages of careful seeding," in *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '07, (Philadelphia, PA, USA), pp. 1027–1035, Society for Industrial and Applied Mathematics, 2007.

[53] N. Görnitz, L. A. Lima, K. Müller, M. Kloft, and S. Nakajima, "Support vector data descriptions and k-means clustering: One class?," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, pp. 3994–4006, Sep. 2018.

[54] G. Giacinto, R. Perdisci, M. D. Rio, and F. Roli, "Intrusion detection in computer networks by a modular ensemble of one-class classifiers," *Information Fusion*, vol. 9, no. 1, pp. 69 – 82, 2008. Special Issue on Applications of Ensemble Methods.

[55] A. Ypma and R. P. W. Duin, "Support objects for domain approximation," in *ICANN 98* (L. Niklasson, M. Bodén, and T. Ziemke, eds.), (London), pp. 719–724, Springer London, 1998.

[56] A. N. Kolmogorov, "ϵ-entropy and ϵ-capacity of sets in function spaces," *American Mathematical Society Translations*, vol. 17, no. 2, pp. 277–364, 1961.

[57] J. M. Vidal, A. L. S. Orozco, and L. J. G. Villalba, "Online masquerade detection resistant to mimicry," *Expert Systems with Applications*, vol. 61, pp. 162 – 180, 2016.

[58] L. Kuncheva, *Combining pattern classifiers: methods and algorithms*. Wiley, Hoboken, 2014.

[59] D. M. J. Tax and R. P. W. Duin, "Combining one-class classifiers," in *Multiple Classifier Systems* (J. Kittler and F. Roli, eds.), (Berlin, Heidelberg), pp. 299–308, Springer Berlin Heidelberg, 2001.

[60] J. Huang and C. X. Ling, "Using auc and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, pp. 299–310, March 2005.

[61] O. Loyola-González, M. A. Medina-Pérez, D. Hernández-Tamayo, R. Monroy, J. A. Carrasco-Ochoa, and M. García-Borroto, "A pattern-based approach for detecting pneumatic failures on temporary immersion bioreactors," *Sensors*, vol. 19, no. 2, 2019.

[62] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.

[63] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the American Statistical Association*, vol. 32, no. 200, pp. 675–701, 1937.

[64] V. N. Vapnik, *Statistical Learning Theory*. Wiley-Interscience, 1998.

[65] R. Duda, P. Hart, and D. Stork, *Pattern classification*. Wiley Interscience, Hoboken, 2001.

[66] A. Avci, S. Bosch, M. Marin-Perianu, R. Marin-Perianu, and P. Havinga, "Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey," in *23th International Conference on Architecture of Computing Systems 2010*, pp. 1–10, Feb 2010.

[67] L. Chen, J. Hoey, C. D. Nugent, D. J. Cook, and Z. Yu, "Sensor-based activity recognition," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, pp. 790–808, Nov 2012.

[68] M. A. Labrador and O. D. L. Yejas, *Human Activity Recognition: Using Wearable Sensors and Smartphones*. Chapman & Hall/CRC, 2013.

[69] G. Lamprinakos, S. Asanin, T. Broden, A. Prestileo, J. Fursse, K. Papadopoulos, D. Kaklamani, and I. Venieris, "An integrated remote monitoring platform towards telehealth and telecare services interoperability," *Information Sciences*, vol. 308, pp. 23 – 37, 2015.

[70] J. Parkka, M. Ermes, P. Korpipaa, J. Mantyjarvi, J. Peltola, and I. Korhonen, "Activity classification using realistic data from wearable sensors," *IEEE Transactions on Information Technology in Biomedicine*, vol. 10, pp. 119–128, Jan 2006.

[71] S. Patel, H. Park, P. Bonato, L. Chan, and M. Rodgers, "A review of wearable sensors and systems with application in rehabilitation," *Journal of NeuroEngineering and Rehabilitation*, vol. 9, p. 21, Apr 2012.

[72] M. Stikic, D. Larlus, S. Ebert, and B. Schiele, "Weakly supervised recognition of daily life activities with wearable sensors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, pp. 2521–2537, Dec 2011.

[73] L. Breiman, "Random forests," *Machine Learning*, vol. 45, pp. 5–32, Oct 2001.

[74] I. Mitchell, M. Locke, M. Wilson, and A. Fuller, *The White Book of... Big Data: The Definitive Guide to Big Data*. Fujitsu Services Ltd, 2013.

[75] R. Buyya, R. N. Calheiros, and A. V. Dastjerdi, *Big Data: Principles and Paradigms*. Morgan Kaufmann, 2016.

[76] H. Karau, A. Konwinski, P. Wendell, and M. Zaharia, *Learning Spark: Lightning-Fast Big Data Analysis*. O'Reilly Media, 2015.

[77] A. Benavoli, G. Corani, J. Demsar, and M. Zaffalon, "Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis," *Journal of Machine Learning Research*, vol. 18, pp. 1–36, 2017.

[78] A. Benavoli, F. Mangili, G. Corani, M. Zaffalon, and F. Ruggeri, "A bayesian wilcoxon signed-rank test based on the dirichlet process," *31st International Conference on Machine Learning, ICML 2014*, vol. 3, pp. 2703–2713, 01 2014.

[79] J. Carrasco, S. García, M. del Mar Rueda, and F. Herrera, "rnpbst: An r package covering non-parametric and bayesian statistical tests," in *Hybrid Artificial Intelligent Systems* (F. J. Martínez de Pisón, R. Urraca, H. Quintián, and E. Corchado, eds.), (Cham), pp. 281–292, Springer International Publishing, 2017.

[80] D. H. Wolpert, "The lack of a priori distinctions between learning algorithms," *Neural Computation*, vol. 8, no. 7, pp. 1341–1390, 1996.

[81] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms: Experiments and analyses," *Pattern Recognition*, vol. 74, pp. 406 – 421, 2018.

Apéndice A

Evaluación de *Bagging-RandomMiner* en otros dominios de aplicación

Resumen: En este apéndice se decidió evaluar a *Bagging-RandomMiner* en otros contextos de aplicación, sin perder el enfoque de Detección de anomalías. Esto permitió establecer una perspectiva más amplia de las ventajas y debilidades de nuestro algoritmo y dejar en claro en cuales dominios de aplicación resulta más eficiente.

En este apéndice, se comparará a *Bagging-RandomMiner* en distintos dominios de aplicación, aunque creemos fielmente en el teorema del *no free lunch* propuesto por el investigador *Wolpert* [80], se pretende evaluar a *Bagging-RandomMiner* en escenarios similares y determinar su efectividad. La selección de los conjuntos de datos al igual que los algoritmos para la comparación se recolectaron del trabajo de investigación de *Domingues et al.* [81]. A continuación se explican los detalles de las bases de datos empleadas:

- **Abalone:** Es un conjunto de datos del repositorio *UCI* que originalmente pretendía predecir la edad de los abulones (*abalone* en inglés), esta se determina cortando la cáscara a través del cono, teñiéndola y contando el número de anillos a través de un microscopio, una tarea aburrida y que consume mucho tiempo. Esta versión del conjunto de datos se modificó para ser abordado con un enfoque des-balanceado (detección de anomalías), se constituye por

1,545 instancias, 8 características (propiedades fisiológicas de los abulones) y se consideró la clase 18 como la minoritaria y el resto como la clase predominante.

- **ANN-Thyroid:** Es una base de datos del repositorio *UCI* que pretende identificar las personas con hipertiroidismo por medio de aspectos fisiológicos (21 características) y se conforma de 3,772 instancias. Este conjunto de datos consta de 3 clases (negativo, discordante, positivo) siendo la clase positiva la minoritaria.

- **Car:** Es un conjunto de datos del repositorio *UCI* que busca clasificar 4 clases (*unacc, acc, good* y *v-goog*) refiriéndose al aspecto general de un automóvil, este contiene 6 atributos que representan el estado actual del automóvil como: valor de compra (*v-high, high, med, low*), costo de mantenimiento (*v-high, high, med, low*), número de puertas (2, 3, 4 ó más), número de personas (2, 4 ó más), tamaño del porta equipajes (*small, med, big*) y la seguridad (*low, med, high*). Se compone por 1,728 instancias y la clase *good* es la minoritaria.

- **Mushroom-Sub:** Este conjunto de datos del repositorio *UCI* incluye descripciones de muestras hipotéticas correspondientes a 23 especies de hongos de la familia *Agaricus* y *Lepiota*. Cada especie puede identificarse como definitivamente comestible (clase *e*) y definitivamente venenosa (clase *p*), las 26 características se basan en los aspectos fisiológicos y la posición geográfica de los hongos. El número de instancias que comprende la base de datos es de 5,644 y la clase *p* es la minoritaria.

- **Wine-Quality:** Es una base de datos que representa una cata de vinos, constituida por dos variantes (rojo y blanco) del portugués *Vinho Verde*. Una serie de expertos los clasificaron del 0 al 10 según su calidad, dejando 10 clases presentes. Posee un total de 11 características, 4,898 instancias y el valor de calidad pésimo (clase 3) fue seleccionado como la clase minoritaria.

- **Yeast:** Esta base de datos seleccionada del repositorio *UCI* busca predecir los sitios de localización celular de proteínas (10 posibles soluciones), consta de 8 atributos referentes a la estructura de la proteína. El número de mediciones es de 1,484 y la clase *POX* (*peroximal*) se identifica como la minoritaria.

En la tabla A.1 se resumen las características de los conjuntos de datos seleccionados.

Tabla A.1: Definición de la estructura de los conjuntos de datos seleccionados para la detección de anomalías.

Nombre	Número de instancias	Número de características	Número de clases	Anomalías (clase minoritaria)
Abalone	1,545	8	2	Clase 18: 20 (2.58 %)
ANN-Thyroid	3,772	21	3	Clase 1: 93 (2.46 %)
Car	1,728	6	4	Clase good: 65 (3.76 %)
Mushroom-Sub	5,644	22	6	Clase p: 568 (10 %)
Wine-Quality	4,898	11	10	Clase 3: 20 (0.4 %)
Yeast	1,484	8	10	Clase POX: 20 (1.34 %)

En la tabla A.2 se muestran los algoritmos seleccionados respecto al trabajo de los autores en [81], indicando los parámetros de ejecución.

A.1. Resultados

Para medir la eficiencia de los algoritmos se utilizó la métrica de área bajo la curva (*AUC*), así como los tiempos de ejecución (en segundos) para poder comparar la eficiencia con la rapidez de los métodos. El equipo de cómputo para realizar las ejecuciones consiste de 16Gb de RAM, un procesador *Intel Core i7* de sexta generación y un sistema operativo de 64 bits.

El método estadístico más pertinente para valorar los algoritmos es la prueba no-paramétrica de *Friedman* ilustrada mediante un diagrama de diferencia crítica (CD). En la tabla A.3 se muestran los resultados de todos los algoritmos en cada uno de los conjuntos de datos seleccionados evaluando la métrica de *AUC*. Al evaluar los algoritmos podemos observar que un promedio general *Bagging-TPMiner* obtiene la mejor puntuación, seguido de *OCKRA* y *Bagging-RandomMiner*.

En la tabla A.4 se muestran los tiempos de ejecución (en segundos) de cada uno de los algoritmos seleccionados. En general el algoritmo más rápido es *LOF*, sin embargo obtiene el penúltimo lugar en la métrica de *AUC*. Si analizamos los tres algoritmos con el promedio más alto en *AUC* observamos que *Bagging-TPMiner* es el método más costoso, seguido de *OCKRA* y después *Bagging-*

RandomMiner. Esto indica que, depende del escenario de aplicación el investigador puede decidir en sacrificar un poco de precisión, pero, ganar rápidez.

Tabla A.2: Algoritmos de Detección de anomalías con sus respectivos parámetros de ejecución.

Algoritmo	**Lenguaje**	**Parámetros**
One-Class Support Vector Machine (OCSVM)	Python	nu = 0.1 kernel = rbf gamma = 0.1
Isolation Forest (IF)	Python	n_estimators = 100 max_samples = auto contamination = legacy
Kernel Density Estimation (KDE)	Python	bandwidth = 1.0 kernel = gaussian metric = euclidean
Local Outlier Factor (LOF)	Python	n_neighbors = 20 leaf_size = 30, metric = minkowski
Linear Discriminant Analysis (LDA)	Python	solver = svd tol = 0.0001
Bagging-RandomMiner (BRM)	Python	T = 100 (número de clasificadores) RS = 15 % (porcentaje de clasificadores) $\mathbf{MRO_{percent}}$ = 10 % (porcentaje MROs) disType = Mahalanobis
One-Class K-means with Randomly-projected features Algorithm (OCKRA)	Python	T = 100 (número de clasificadores) K = 10 (número de centroides) disType = Euclídea
Bagging-TPMiner (BTPM)	Python	T = 100 (número de clasificadores) RS = 15 % (porcentaje de clasificadores) disType = Mahalanobis

Tabla A.3: Comparativa de la métrica de *AUC* de los algoritmos de Detección de anomalías en conjuntos de datos *bench-mark*

Conjunto de datos	OCSVM	BTPM	BRM	OCKRA	IF	KDE	LDA	LOF
Abalone	71.62	**92.83**	88.85	87.46	57.98	78.55	76.71	77.62
NN-thyroid	69.60	**97.00**	91.98	93.36	64.36	89.22	80.79	71.84
Car	46.74	62.19	62.44	66.81	59.31	**68.84**	64.72	49.20
Mushroom	52.20	59.53	48.72	**71.47**	50.91	67.24	56.69	47.22
Wine Quality	75.17	86.04	**87.81**	74.38	67.94	74.18	75.07	77.57
Yeast	70.56	70.63	72.83	73.24	47.28	71.46	**74.25**	70.25
Promedio	64.31	**78.04**	75.44	77.79	57.96	74.91	71.37	65.62

Tabla A.4: Comparativa de la métrica tiempo (segundos) de los algoritmos de Detección de anomalías en conjuntos de datos *bench-mark*

Conjunto de datos	OCSVM	BTPM	BRM	OCKRA	IF	KDE	LDA	LOF
Abalone	0.02	34.09	7.09	19.68	0.19	0.05	1.16	**0.00**
NN-thyroid	0.13	129.05	27.73	40.14	0.23	0.23	1.31	**0.02**
Car	0.03	25.12	6.35	11.63	0.19	0.04	0.98	**0.00**
Mushroom	0.37	188.30	4.60	39.63	0.29	0.46	1.24	**0.04**
Wine Quality	0.19	192.85	42.74	66.91	0.26	0.32	1.20	**0.02**
Yeast	0.01	18.49	4.43	14.81	0.18	0.03	1.00	**0.00**
Promedio	0.13	97.98	15.49	32.14	0.22	0.19	1.15	**0.02**

Por último, se aplicó la prueba no-paraetrica de *Friedman* para corroborar si existe diferencia significativa entre los distintos algoritmos en la métrica de *AUC*, en la figura A.1 se muestra el diagrama de diferencia crítica (*CD*) de la comparación.

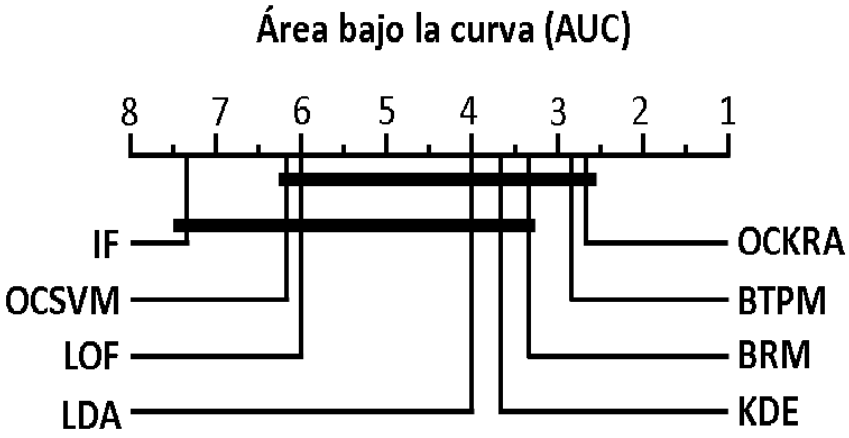


Figura A.1: Diagrama de diferencia crítica de la comparación de los algoritmos de Detección de anomalías en la métrica de *AUC*

Al realizar el análisis estadístico, se determinó que *OCKRA* tiene la puntuación más alta respecto a *AUC* en distintos escenarios de aplicación, sin embargo, *Bagging-RandomMiner* no presenta diferencia significativa con dicha metodología y su complejidad computacional es menor. Esto indica que, el algoritmo más novedoso en estos dominios de aplicación queda a criterio del investigador.