UNIVERSIDAD TÉCNICA DEL NORTE FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y REDES DE COMUNICACIÓN

TEMA:

DESARROLLO DE ALGORITMOS DE CLASIFICACIÓN SUPERVISADA DE BAJO COSTE COMPUTACIONAL PARA SISTEMAS EMBEBIDOS ORIENTADO A LA OPTIMIZACIÓN DE RECURSOS LÓGICOS

AUTOR:

Israel Alexander Gudiño Pabón.

DIRECTOR:

MsC. Luis Suárez

Ibarra-Ecuador

2021

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1004014369		
APELLIDOS Y NOMBRES:	Gudiño Pabón Israel Alexander		
DIRECCIÓN:	El Olivo		
EMAIL:	iagudiniop@utn.edu.ec		
TELÉFONO FIJO:	2-580387	**TELÉFONO MÓVIL:**	0958905611

DATOS DE LA OBRA	
TÍTULO:	DESARROLLO DE ALGORITMOS DE CLASIFICACIÓN SUPERVISADA DE BAJO COSTE COMPUTACIONAL PARA SISTEMAS EMBEBIDOS

	ORIENTADO A LA OPTIMIZACIÓN DE RECURSOS LÓGICOS
AUTOR (ES):	Gudiño Pabón Israel Alexander
FECHA: DD/MM/AAAA	26 de mayo del 2021
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** ☐ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Electrónica y Redes de Comunicación
ASESOR /DIRECTOR:	MsC. Luis Suárez

UNIVERSIDAD TÉCNICA DEL NORTE

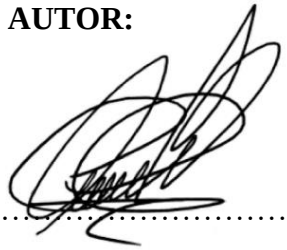
FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

2. CONSTANCIAS

El autor manifiesta que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es el titular de los derechos patrimoniales, por lo que asume la responsabilidad sobre el contenido de la misma y saldrá en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 26 días del mes de mayo del 2021

EL AUTOR:



……

Nombre: Israel Alexander Gudiño Pabón

CI:1004014369

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CERTIFICACIÓN

MSC. LUIS SUÁREZ, DIRECTOR DEL PRESENTE TRABAJO DE TITULACIÓN

CERTIFICA:

Que, el presente trabajo de Titulación "DESARROLLO DE ALGORITMOS DE CLASIFICACIÓN SUPERVISADA DE BAJO COSTE COMPUTACIONAL PARA SISTEMAS EMBEBIDOS ORIENTADO A LA OPTIMIZACIÓN DE RECURSOS LÓGICOS", ha sido desarrollado por el señor Gudiño Pabón Israel Alexander bajo mi supervisión

Es todo en cuanto puedo certificar en honor de la verdad.

LUIS EDILBERTO SUAREZ ZAMBRANO

Firmado digitalmente por LUIS EDILBERTO SUAREZ ZAMBRANO
Fecha: 2021.05.26 15:36:47 -05'00'

.....

MsC. Luis Suárez

1002304291

DIRECTOR

UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

DEDICATORIA

El presente trabajo de titulación lo dedico principalmente a Dios, por ser el inspirador y permitirme el haber llegado hasta este momento tan importante de mi formación profesional.

A mis padres, por ser el pilar más importante y por demostrarme siempre su cariño y apoyo incondicional. Gracias por cada consejo, y ejemplo de trabajo duro que me brindan.

AGRADECIMIENTOS

Quiero agradecer a mis padres Hugo y Shaned por su amor, cariño y dedicación en cada uno de mis pasos y metas, ustedes han sido un gran ejemplo a seguir y han sido quienes me hay ayudado a llegar tan lejos. A mi hermana Nicole quien ha sido un gran apoyo y parte importante de este logro. A Joselyn, por ser la mejor amiga que alguien puede tener, por creer en mí y estar siempre en cada momento de mi vida.

A mis compañeros y amigos quienes estuvieron presentes y formaron parte de la duración de mi carrera, gracias por su compañía y apoyo brindado. De manera especial agradezco a mi director de tesis el MsC. Luis Suárez por cada enseñanza, consejo y tiempo dedicado para la realización de este proyecto.

Israel Alexander Gudiño Pabón

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

En el presente trabajo de titulación se realiza una implementación de algoritmos de clasificación supervisada para sistemas embebidos de bajos recursos computacionales, ya que estos sistemas se ven limitados en el procesamiento de datos cuando se usan algoritmos de aprendizaje de máquina robustos, ocasionando en el sistema un alto tiempo de respuesta, debido a que, en el procesamiento es en donde se consume la mayor cantidad de recursos.

Para su desarrollo, se aplicó técnicas de optimización de código para simplificar los procesos y el uso de recursos lógicos. Además, los mencionados algoritmos fueron evaluados bajo criterios de tiempo de procesamiento, tasa de error, tamaño y rendimiento. En la parte final como resultados relevantes, se observan las pruebas de los algoritmos de clasificación con cada base de datos utilizada, como también la implementación de curvas ROC y AUC. Con ello, se realiza una comparación de los resultados para determinar la eficiencia lograda por cada algoritmo optimizado.

ABSTRACT

In this degree work, implementation of supervised classification algorithms is carried out for embedded systems with low computational resources, since these systems are limited in data processing when robust machine learning algorithms are used, causing the system to high response time because the processing is where the greatest amount of resources is consumed.

For its development, code optimization techniques were applied to simplify processes and the use of logical resources. In addition, the aforementioned algorithms were evaluated under the criteria of processing time, error rate, size, and performance. In the final part, as relevant results, the tests of the classification algorithms are observed with each database used and the implementation of ROC and AUC curves. With this, a comparison of the results is made to determine the efficiency achieved by each optimized algorithm.

CAPÍTULO I. ANTECEDENTES

Introducción

En este capítulo, se presenta la argumentación necesaria para el desarrollo del trabajo de titulación. Se inicia con la descripción del problema que esta investigación debe solucionar. A continuación, se define objetivos generales y específicos. De la misma forma, se presenta el alcance y la justificación donde se expone la importancia de la realización de este trabajo.

1.1. Problema

El diseño de sistemas embebidos y los problemas de optimización juegan un papel muy importante, y con frecuencia impactan significativamente en su rendimiento, ya que han sido ampliamente utilizados en numerosos campos, tales como en sistemas de control en la industria, recolección de información, electrodomésticos, equipos de comunicación, instrumentos médicos e instrumentos de inteligencia (Hongxing & Tianmiao, 2006). Por esta razón, los diseñadores de sistemas embebidos deben prestar especial atención en la creación de software óptimo para estos dispositivos (Park, Lee, & Lee, 2013).

Los sistemas embebidos se ven limitados en el procesamiento de datos cuando se usan algoritmos de aprendizaje de máquina robustos. Este es el caso de los algoritmos de clasificación supervisada, ocasionando en el sistema alto tiempo de respuesta, debido a que, en el procesamiento es en donde se consume la mayor cantidad de recursos (Park, Lee, & Lee, 2013).

Los algoritmos de clasificación supervisada operan usualmente sobre la información suministrada por un conjunto de muestras, patrones y conjuntos de datos. Logrando así, un alto rendimiento de clasificación al reducir una función a partir de datos de entrenamiento (Feng, 2012). Estos algoritmos que se están usando en los sistemas embebidos son muy complejos y

tienen baja velocidad de operación debido a que en su código fuente se identifican bucles críticos y son demasiado extensos, limitando su procesamiento (Park, Lee, & Lee, 2013). Esto ocasiona una carga computacional pesada, convirtiendo a las funciones de los sistemas embebidos en soluciones no viables, porque la característica principal de estos sistemas es que deben ser flexibles y adaptables (Mondkjar, 2002).

Por lo tanto, el presente proyecto pretende implementar algoritmos de aprendizaje automático en sistemas embebidos de bajos recursos computacionales con criterios de optimización de código para simplificar los procesos y el uso de recursos lógicos. Como resultado, el sistema embebido se convertirá en un dispositivo flexible, capaz de tomar sus propias decisiones. Además, los mencionados algoritmos evaluarán su desempeño bajo criterios de: tiempo de procesamiento, tiempo de respuesta, requisitos de memoria y utilización de espacio en disco.

En la actualidad los algoritmos que se están usando en los sistemas embebidos son muy complejos y tienen baja velocidad de operación, ocasionando alto tiempo de respuesta ya que en el procesamiento es en donde se consume la mayor cantidad de recursos, tanto de tiempo como de potencia (Park, Lee, & Lee, 2013). Esta es la razón por la cual se pretende implementar algoritmos de aprendizaje automático en sistemas embebidos de bajos recursos computacionales con criterio de optimización de código para simplificar los procesos y recursos del sistema.

1.2. Objetivos

1.2.1. Objetivo General.

Desarrollar algoritmos de clasificación supervisada de bajo coste computacional para Sistemas Embebidos orientado a la optimización de recursos lógicos.

1.2.2. Objetivos Específicos.

- Elaborar una fundamentación teórica acerca de los algoritmos de clasificación supervisada usados en sistemas embebidos.
- Realizar una comparación de algoritmos de clasificación usados en sistemas embebidos para determinar los idóneos a ser desarrollados.
- Desarrollar los algoritmos de clasificación con criterios de optimización de recursos lógicos para ser almacenados en los sistemas embebidos.
- Realizar las pruebas de rendimiento de los algoritmos de clasificación supervisada desarrollados para evaluar su desempeño en los sistemas embebidos.

1.3. Alcance

El presente proyecto tiene como finalidad el desarrollo de algoritmos de clasificación supervisada de bajo coste computacional con criterios de distancias, super planos y probabilidad para sistemas embebidos orientados a la optimización de recursos lógicos.

El proyecto iniciará con la fundamentación teórica de todo lo que se refiere a algoritmos de clasificación supervisada para sistemas embebidos, en el cual se detallará algoritmos de clasificación con criterios a base de distancias, super planos y probabilidad. Describiendo su funcionamiento y los tipos de datos que utiliza cada uno.

Para la comparación de los algoritmos de clasificación supervisada para sistemas embebidos, se usará la matriz de confusión y grandes conjuntos de datos obtenidos de los repositorios *IEEE DataPort* y *UCI Machine Learning*, con lo cual se evaluará el desempeño de los algoritmos con respecto a su rendimiento de clasificación. Esto se llevará a cabo a partir de un conteo de aciertos y errores, donde se obtendrá métricas de sensibilidad, especificidad, precisión, valor de predicción negativa y error de clasificación. Esto se hará con el fin de determinar los adecuados, que puedan servir de base para el diseño y desarrollo de los algoritmos a ser aplicados en la optimización de recursos para sistemas embebidos.

Para el desarrollo de los algoritmos de clasificación supervisada para sistemas embebidos, la implementación se la va a realizar en base a los algoritmos que se seleccionó y mediante técnicas de optimización de código como optimizaciones que no modifican la estructura, simplificaciones algebraicas, optimizaciones en bucles. Todo esto se lo realizará utilizando criterios de optimización de recursos lógicos en tiempo de procesamiento, tiempo de respuesta, requisitos de memoria y utilización de espacio en disco. La implementación se la va a realizar en lenguaje de programación *C++*. Posteriormente se creará una librería con los algoritmos de clasificación supervisada desarrollados para sistemas embebidos.

Finalmente, una vez terminado con el desarrollo de los algoritmos de optimización para sistemas embebidos se realizará las pruebas de rendimiento usando la notación *Big O*, la cual bajo métricas cuantitativas de equilibrio entre tiempo de procesamiento, memoria utilizada y rendimiento del sistema determinará el porcentaje de optimización.

1.4. Justificación

En la actualidad con el desarrollo de la tecnología y evolución de los sistemas embebidos se han generado gran cantidad de datos listos para ser recolectados, procesados y almacenados con diversos fines tecnológicos (Becker, Mcmullen, & King, 2015).

El Plan Nacional de Desarrollo 2017-2021, manifiesta en uno de sus objetivos, que el país debe gestionar sus recursos estratégicos en el marco de una inserción internacional, permitiendo la innovación y desarrollo tecnológico con el fin de mejorar la calidad de servicio, además, que contribuya también al incremento generalizado del bienestar de sus habitantes (Plan Nacional de Desarrollo, 2017). El presente proyecto, pretende implementar algoritmos de aprendizaje automático en sistemas embebidos de bajos recursos computacionales con criterio de optimización de código para simplificar los procesos y recursos del sistema. Como resultado, el sistema se convertirá en un dispositivo flexible, capaz de tomar sus propias decisiones ya que un algoritmo de clasificación supervisada va mejorando a medida que clasifica varios tipos de datos. Con esto, se mejorará el desarrollo de aplicaciones en Sistemas Embebidos, ya que en la actualidad no existen algoritmos de aprendizaje automático enfocados a redes de sensores (Fouad, Hafez, & Farouk, 2016).

Con lo expuesto anteriormente, se ve la necesidad del desarrollo de algoritmos de clasificación supervisada con criterios de optimización de recursos enfocados a sistemas embebidos, ya que se vuelve una necesidad latente. Ya que estos deben adaptarse a los sistemas inteligentes actuales y poder superar los desafíos de análisis y clasificación de datos.

Además, aporta al cumplimiento de la misión universitaria, la cual busca una generación que fomente y ejecute procesos de investigación, de transferencia de saberes, de conocimientos científicos, de tecnológicos y de innovación (UTN, 2008).

CAPÍTULO II. REVISIÓN BIBLIOGRÁFICA

En este apartado se establecerán conceptos relacionados al aprendizaje automático, desde conceptos generales como sistemas embebidos, fundamentos de programación, algoritmos de clasificación y métodos de validación.

2.1. Sistemas embebidos

Los sistemas embebidos (SE), son circuitos electrónicos capaces de realizar operaciones de computación en tiempo real, cuyo objetivo es cumplir ciertas tareas específicas, como medir la temperatura y procesamiento de información (Salas Arriarán, 2015). Al contrario de lo que ocurre con los ordenadores de propósito general que están diseñados para cubrir un amplio rango de necesidades, los sistemas embebidos fueron diseñados para cubrir necesidades específicas (Lifelong Learning, 2011).

Lifelong (2011), indica que "las principales características de un sistema embebido son el bajo costo y bajo consumo de potencia. Además, estos sistemas emplean procesadores muy básicos, relativamente lentos y memorias pequeñas que minimizan los costos". En la Figura 1, se puede observar los elementos básicos de un sistema embebido (SE), el cual posee una arquitectura semejante a la de un PC; es decir, está compuesto por una unidad de procesamiento CPU, memorias ROM, RAM, sistemas de control e interfaces de entrada/salida.

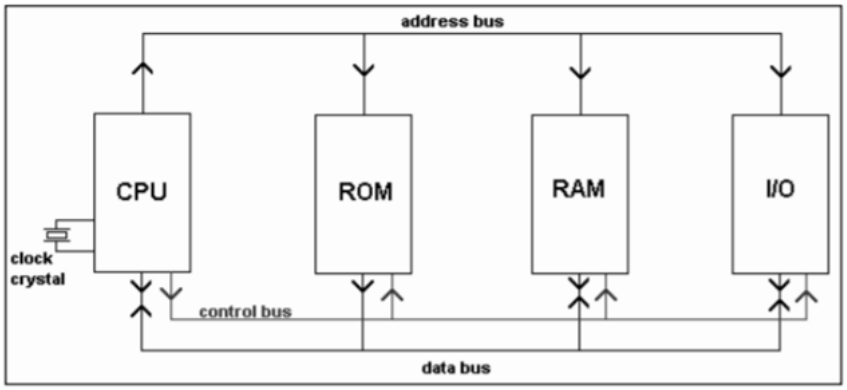


Figura 1 Arquitectura de un sistema embebido de configuración mínima

Fuente: (Lifelong Learning, 2011)

Estos sistemas, se encuentran garantizados por la ciencia del *software*, ya que su capacidad se basa en la complejidad del programa al ejecutarse y el tiempo de procesamiento del mismo (Yamane, 2017). Por tanto, su característica principal es que debe tener un acceso de lectura y escritura lo más rápido posible; para que el microprocesador no pierda tiempo en tareas que no son específicamente de cálculo; ya que en la memoria es en donde se almacena el código de los programas que el sistema puede ejecutar, dependiendo de la aplicación a la que este orientado. (Lifelong Learning, 2011).

2.1.1. Aplicaciones.

Los SE en la última década, han formado parte de casi todos los sectores en rápido desarrollo (A. Sharma, 2013), por ende, Sandoval (2019) afirma que "estos sistemas se diseñan para trabajar en ambientes hostiles como la intemperie, procesos con presencia de agentes corrosivos, presiones variables o temperaturas extremas" (p. 1). En la Tabla 1, se lista las aplicaciones más importantes que tienen los sistemas embebidos, desde el punto de vista de varios autores.

Tabla 1. Aplicaciones de los SE

Aplicaciones de los sistemas embebidos	**Autor**
Los sectores de aplicación de los SE más importantes son: automovilístico, aeronáutico, transporte y comunicaciones móviles.	(A. Sharma, 2013)
Los teléfonos inteligentes están equipados con muchos sensores y controladores, los cuales pueden conectarse a los sistemas del vehículo. Esto permite crear varias	(Alsibai, 2015)

aplicaciones de monitoreo integral para el correcto desempeño de todos los dispositivos.	
Estos sistemas, tienen numerosas aplicaciones en ciencias básicas, clínicas, visuales y biomédicas.	(Yan & Nirenberg, 2017)
Los SE encuentran un amplio uso en aplicaciones de sistemas de control de motores de automóviles, dispositivos biomédicos y electrodomésticos.	(Jayanth, 2018)
Estos sistemas inteligentes, tienen numerosas aplicaciones en el sistema de atención médica, las cuales proporcionan soluciones para el paciente y los profesionales de la salud.	(Thakkar, 2018)

Fuente: Autoría

Una vez expuesto el punto de vista de varios autores acerca de las aplicaciones de SE y en base a las más nombradas y relevantes de acuerdo con su impacto, se considera que los sectores de estudio son: medicina, sistemas móviles, sector automotriz y transporte.

Una de las aplicaciones de los SE que tiene más impacto está basada en la adquisición de datos orientados a la medicina; donde estos sistemas proporcionan numerosas soluciones para el paciente y los profesionales de la salud (Thakkar, 2018). Ya que los usuarios, pueden seleccionar diferentes funciones y aplicaciones según sus propias necesidades; como el monitoreo de la frecuencia cardiaca, sensores de presión de la sangre y glucómetros (Ya-lin Miao et al., 2006). Permitiendo así, que el equipo médico sea más eficiente al monitorear en tiempo real la salud del paciente (Thakkar, 2018).

En cuanto a la industria de las telecomunicaciones, esta utiliza numerosos SE como conmutadores telefónicos para la red de teléfonos móviles, ayudando a garantizar redes de alta velocidad al incorporar rápidamente conexiones *Ethernet* en aplicaciones embebidas avanzadas (A. Sharma, 2013). Además, estos sistemas son necesarios para el desarrollo de aplicaciones para diferentes tecnologías móviles como *3G, 4G / LTE, ZigBee y Z-wave* (Nuratch, 2018), siendo todo esto posible gracias a la nueva generación de sistemas embebidos de bajo costo computacional y con un alto nivel de capacidad de cálculo (Selmani, 2018).

En la Figura 2, se presenta un ejemplo de arquitectura de un sistema *Zigbee*, el cual consta de tres nodos que utilizan un SE cada uno para la comunicación; también consta de una estación de control y un dispositivo móvil para la recepción de notificaciones sobre el estado y monitoreo de la red.

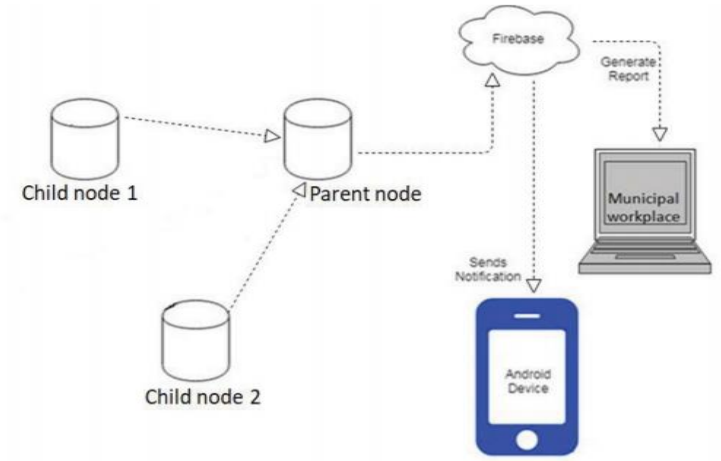


Figura 2. Arquitectura de un sistema *Zigbee* que utiliza un SE en cada nodo para la comunicación.

Fuente: (Lutful et al., 2019)

Por otra parte, en relación con el sector automotriz y de transporte, el mundo se mueve hacia nuevos enfoques innovadores para hacer la vida humana mucho más simple y fácil, esto se logra al automatizar cada tarea para proporcionar un mejor servicio vehicular a los consumidores (Lutful, 2019), sumado a esto, en la última década el monitoreo de vehículos ha ayudado con el análisis para determinar qué tipo de vehículos son los principales

contribuyentes de la contaminación y, por lo tanto, para así elaborar estrategias adecuadas para abordar tales problemas (Gupta & Rakesh, 2018).

La característica principal de este tipo de aplicaciones es su bajo consumo de energía y la capacidad de integrar a muchos sensores en cualquier momento; ya que los sistemas embebidos empleados como el *Raspberry Pi* ejecuta los algoritmos de aprendizaje automático para luego enviar los resultados y la predicción en vivo de la condición del vehículo al conductor mediante aplicaciones móviles (Srinivasan, 2018).

2.1.2. Clasificación de sistemas embebidos.

Agarwal (2015), plantea que "los sistemas embebidos se clasifican en independientes, sistemas que operan en tiempo real, sistemas de red y móviles; los cuales tienen un microprocesador a pequeña escala, mediana escala y sofisticado". Dividiendo en 2 categorías a los SE, basados en su rendimiento y en su microprocesador, las cuales se observa en la Figura 3.

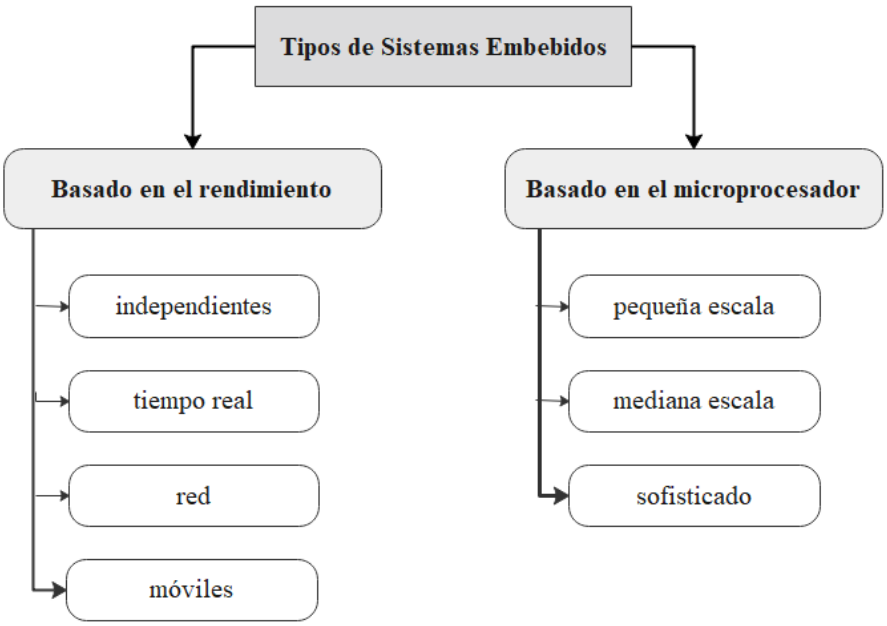


Figura 3. Tipos de sistemas embebidos

Fuente: (Agarwal, 2015)

2.1.3. Sistemas embebidos basados en el rendimiento.

De acuerdo con Baek & Lee (2007), “el rendimiento de los sistemas embebidos se basa en sus límites físicos, ya que de estos depende su potencia al momento de procesar información y esta varía de acuerdo con sus compontes o técnicas de procesamiento”. Por ejemplo, los sistemas embebidos independientes no requieren un sistema *host* como una computadora, ya que funcionan por sí mismos. (Agarwal, 2015). Donde, su función principal es lograr un tiempo de operación sin errores, sin la necesidad de un nodo trabajando en segundo plano (Kohnh & Katzenbeisser, 2019).

Este tipo de SE normalmente se usan en sistemas operativos robóticos, los cuales se basan en la transferencia de datos (Alberri, 2018). En la Figura 4, se muestra el proceso de transferencia de datos en un sistema operativo robótico el cual está formado por una red de sistemas embebidos a los cuales están conectados un sensor ultrasónico, infrarrojo, un motor DC y un codificador; los cuales proporcionan información sobre el entorno y la envían a un servidor mediante una conexión inalámbrica.

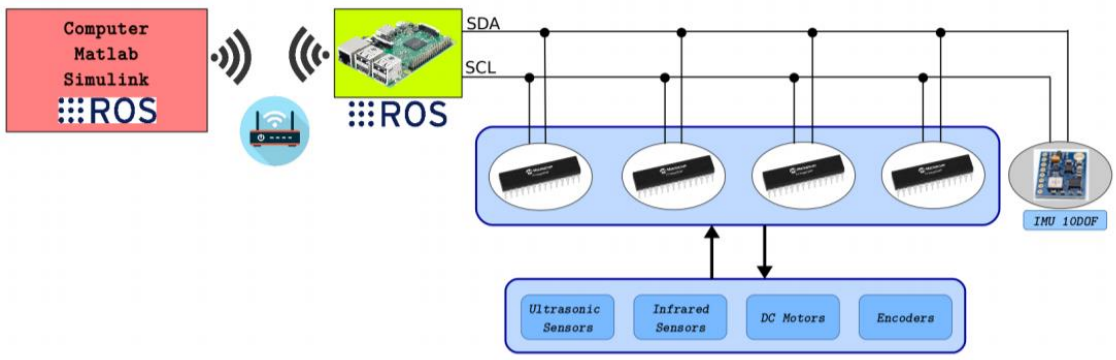


Figura 4. Diagrama de bloques que describe a los SE independientes utilizados en una comunicación de un sistema operativo robótico.

Fuente: (Alberri, 2018)

Por otra parte, con la aparición de nuevos dispositivos médicos portátiles y no intrusivos, se generó un desafío importante es el análisis de datos en tiempo real, por lo que

estos sistemas juegan un papel muy importante al momento de salvar una vida (Surrel, 2015). Un sistema embebido en tiempo real se define como un sistema que siguen los plazos de tiempo para completar una tarea. La característica principal de estos SE, es la de maximizar el número de tareas en un tiempo reducido (Lin, 2016). En la Figura 5, se presenta el esquema de un sistema médico de monitoreo y adquisición de datos; el cual consta de sensores y actuadores conectados a un sistema embebido de tiempo real, donde toda la información recolectada es subida a un servidor y a una página en la red; la cual maneja protocolos de Transferencia de Hipertexto (HTTP), protocolos de Transporte de Mensajes Cliente/Servidor (MQTT) y protocolos de seguridad en Servicios Web (WS).

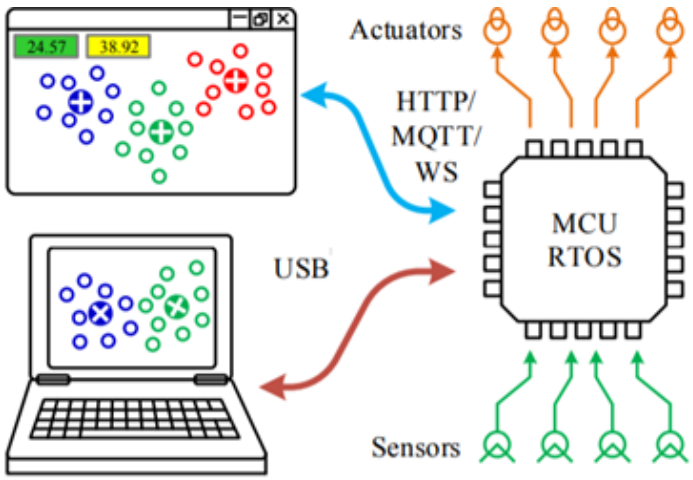


Figura 5. Esquema de un SE que opera datos en tiempo real para la adquisición de datos

Fuente: (Nuratch, 2018)

Por otra parte, los SE en red se relacionan con las redes *LAN*, *WAN* o redes de acceso, y la conexión puede ser cableada o inalámbrica. Ya que el diseño de este tipo de tecnologías de comunicación depende de la capacidad y rendimiento de estos sistemas (Sakurai, 2015). Por esto, la flexibilidad y el bajo precio, les ha permitido convertirse en la mejor solución a problemas de conectividad en el ámbito de las Telecomunicaciones (Saguil & Azim, 2020).

De modo que, de acuerdo con Pašalić (2017), “la utilización de teléfonos móviles con SE integrados con plataformas de servidores virtualizados son la solución al intercambio de datos a gran escala en el sector de la telefonía móvil”. Estos son sistemas inteligentes que se utilizan como terminal de monitoreo, cuya principal función es recibir y mostrar datos a través del análisis preliminar y la verificación de información (Zai-ying & Liu, 2015).

2.1.4. Sistemas embebidos basados en el microprocesador.

El microprocesador de un SE es de vital importancia, ya que de esto depende el procesamiento y tiempo de ejecución de los procesos designados que debe realizar (Luyan, 2013), ofreciendo un gran nivel de flexibilidad y escalabilidad del diseño (Patel & Rajawat, 2014). Agarwal (2015), plantea que “los sistemas embebidos basados en el rendimiento se clasifican en sistemas a pequeña escala, mediana escala y sofisticados”.

Los sistemas embebidos a pequeña escala están diseñados con un solo microcontrolador; que incluso puede ser activado por una batería. Estos también son usados en aplicaciones de reconocimiento facial capaz de realizar clasificaciones de edad y género en imágenes faciales (Wang, 2020). Mientras, que los SE de mediana escala se construyen principalmente con el objetivo de priorizar la seguridad en sistemas de comunicación, usando solo un microcontrolador de 16 o 32 bytes (Profentzas, 2019). Además, como lo hace notar Upadhyay & Dhapola (2015), estos “tipos de sistemas integrados tienen complejidades de *hardware* y *software”*.

En cuanto a los sistemas embebidos sofisticados Agarwal (2015), menciona que “estos tienen enormes complejidades de implementación en el *hardware* y *software*, ya que pueden necesitar procesadores escalables o configurables”. Además, una de sus aplicaciones más importantes son las plantas fotovoltaicas, ya que estos sistemas generan grandes beneficios en la monitorización y gestión de la capacidad térmica y de voltaje de las plantas (Tabari, 2015).

2.2. Arduino y Raspberry

Arduino es un sistema integrado que está diseñado para controlar una variedad de funciones ya sea de monitoreo o control. Este es un SE de *hardware* abierto en el cual el programa se implementa mediante un entorno propio de programación que utiliza el lenguaje de programación *C++*, donde su ejecución se divide en dos partes: configuración y compilación (Voudoukis, 2019). Los sistemas embebidos arduino están diseñados para adaptarse a tecnologías de ingeniería e informática por su fácil manejo (Fiore, 2020). En la Figura 6, se observa el aspecto de una placa arduino, la cual consta de pines de entrada, salida, comunicación, puertos *USB* y un puerto serial.

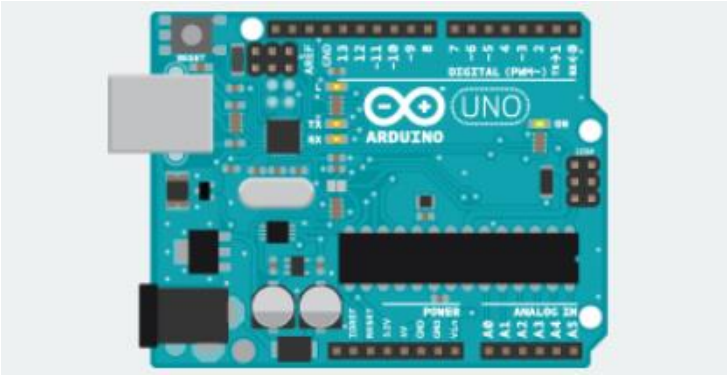


Figura 6. Sistema embebido Arduino

Fuente: (Arduino, 2020)

Estos SE tienen un hardware limitado y en algunos casos su memoria no soporta el tamaño de los datos o del programa a compilar, la cual varía dependiendo del tipo de memoria que se utilice, ya que las placas arduino tienen diferentes tipos de memoria, acorde a su versión (García, 2014). En la Tabla 2, se presenta los tipos de memoria *flash*, *RAM*, *EEPROM* que tiene un arduino y los rangos de memoria disponibles en el mercado.

Tabla 2. Tipos de memoria de un arduino

Tipo de memoria	Descripción	Rangos Disponibles			
		Pro mini	**Mini**	**Arduino UNO**	**Mega 2506**
Flash	Esta memoria almacena el código del programa.	16KB	32KB	32 KB	256 KB
RAM	Esta memoria es volátil y almacena los datos del programa	1KB	2KB	2KB	8KB
EEPROM	Esta memoria es usada para guardar configuraciones o datos importantes.	512 B	1KB	1KB	4KB

Fuente: (García, 2014), (Arduino, 2020)

En cuanto al *Raspberry*, este es un sistema embebido que tiene una gran potencia de cálculo, tomando en cuenta su pequeño tamaño y fácil configuración (Halfacree, 2018). Se la considera como una computadora portátil, de bajo poder, potente y muy barata de adquirir. Al igual que, el arduino también es un SE de código abierto, proporcionando un mejor desarrollo de código, menos líneas y provee de una función de gestión de memoria automática (Nayyar & Puri, 2015). En la Figura 7, se muestra el aspecto de una placa *Raspberry*, la cual consta de puertos *USB, ethernet* y un microprocesador de 1,4 *GHz*; siendo esta, más potente que las placas arduino.

Figura 7. Sistema embebido *Raspberry*

Fuente: (Halfacree, 2018)

2.3. Fundamentos de Programación

Joyanes (2013) plantea, que "la programación es un proceso de resolución de problemas y para que un procesador realice una tarea se le debe suministrar el algoritmo adecuado" (p. 25). Por lo que el proceso de programación de un sistema embebido requiere inicialmente de un computador, en el cual se ejecuta una herramienta de *software* denominada IDE (*Entorno de Desarrollo Integrado*). Este es un conjunto de instrucciones de programa diseñadas para controlar y coordinar los componentes de *hardware* de un computador y controlar las operaciones de un sistema informático, donde la eficiencia de este sistema depende del uso del lenguaje de programación correcto (Joyanes, 2013).

Por tanto, la evolución de los lenguajes de programación ha ido paralela a la idea de modelos de programación (Mcguire, 2009). En realidad, un modelo de programación se escribe en un formato específico, denominado lenguajes de programación (Joyanes, 2013). En la Tabla 3, se presenta una descripción de los tipos de lenguajes de programación.

Tabla 3. Tipos de lenguaje de programación

Lenguaje de programación	Descripción	Autor
Lenguaje imperativo	El lenguaje imperativo representa el método tradicional de programación. Este es un conjunto de instrucciones que se ejecutan una por una, de principio a fin, de modo secuencial.	(Joyanes, 2013)
Lenguaje declarativo	Los lenguajes declarativos describen el problema en lugar de encontrar una solución algorítmica; es decir, un lenguaje declarativo utiliza el principio del razonamiento lógico para responder a las preguntas o cuestiones consultadas.	(Joyanes, 2013)
Lenguaje orientado a objetos	El lenguaje de programación orientado a objetos se basa en el diseño y construcción de objetos que se componen a su vez de datos y operaciones. Este lenguaje de programación en primer lugar identifica los objetos del problema y a continuación los datos y operaciones que actuarán sobre esos datos.	(Mcguire, 2009), (Joyanes, 2013)

Fuente: Autoría

En este sentido, una vez presentado el resumen de los tipos de lenguaje de programación, se considera que el "lenguaje orientado a objetos" es el indicado a analizar con mejor detalle en el presente trabajo ya que, como lo hace notar Joyanes (2013), es "el lenguaje

de programación dominante y ha sustituido a los lenguajes de programación estructurada" (p. 31). Los lenguajes de programación orientado a objetos que más se usan en sistemas embebidos son *C++*, *Python* y *R* (Okoi, 2019). En la Tabla 4, se lista las características de los lenguajes de programación mencionados anteriormente.

Tabla 4. Lenguajes de programación orientado a objetos

Lenguajes de programación	**Características**	**Autores**
C++	- *C++* es un lenguaje de programación híbrido que contiene la funcionalidad del lenguaje de programación *C*. Su programación es modular. - Su lenguaje de programación es flexible y manejable. - Tiene un amplio catálogo de librerías - Su programación es usada en SE como arduino.	(Peter Kinz, 2017)
Python	- La sintaxis de *Python* es sencilla, similar a la de un pseudocódigo. - Tiene un amplio catálogo de librerías estándar, cuenta con decenas de módulos básicos de programador. - Posee un alto rendimiento al momento de compilar programas.	(Challenger-Pérez, Ivet. Díaz-Ricardo, Yanet. Becerra-García, 2014), (Bahit, 2018)

	- Su extensibilidad, es una de las características más importantes, esto permite la reutilización de código escrito en los lenguajes *C* y *C++*.	
R	- La capacidad gráfica de *R* es muy sofisticada. - Es un *software* libre, no necesita de licencias para su uso. - La librería de *R*, consta de una gran cantidad de paquetes de programación básicos e interactivos. - Debido a su estructura, consume muchos recursos de memoria al momento de procesar.	(Santana & Farfán, 2014), (Wilfrido, 2017)

Fuente: Autoría

Las características presentadas en la Tabla 4, indican que *Python* es un lenguaje de programación eficaz que cuenta con estructuras de datos eficientes de alto nivel enfocadas a la programación orientada a objetos, disponiendo de un extenso repositorio de librerías libres de terceros, programas, herramientas y material extra (Bahit, 2018). Por lo tanto, Python es el lenguaje de programación más eficiente para la programación de Algoritmos de Clasificación Supervisada.

2.3.1. Variables y datos.

Un programa típico utiliza varios valores que cambian durante su ejecución. Esto quiere decir que cuando se crea un programa y un usuario introduce valores en la entrada, estos pueden cambiar a lo largo de la ejecución del programa. A estos valores que cambian se los denomina variables (Nakov, 2013). En cambio, los tipos de datos son conjuntos de valores que tienen

características similares y su valor en bytes va desde 0 a 255; estos pueden ser datos números o tipo cadena (Peter Kinz, 2017).

Una cadena (*string*) de caracteres es una sucesión de caracteres que se encuentran delimitados por una comilla o dobles comillas, según el tipo de lenguaje de programación pueden estar formados de una o varias palabras (Krishnamurthi, 2017). En cuanto a los datos numéricos, estos pueden representarse como tipo numérico entero o tipo numérico real (Peter Kinz, 2017). En la Tabla 5, se presenta los tipos de datos numéricos más usados.

Tabla 5. Tipos de datos numéricos

Tipo numérico	**Tamaño**
Entero	El tamaño de los números enteros se puede representar en 8, 16, 32 bits y 64 bits.
Real (Float)	Los tipos de datos reales se representan en coma o punto flotante y suelen requerir 4, 8, 10 y 12 bytes,

Fuente: (Peter Kinz, 2017)

2.3.2. Vectores y matrices.

De acuerdo con la definición de Joyanes (2013), los "vectores se denominan *arrays* unidimensionales y en ellos cada elemento se define por un índice o subíndice. Estos vectores son elementos de datos escritos de forma secuencial" (p. 262). El almacenamiento de datos en un vector se lo realiza en celdas o posiciones secuenciales. En la Figura 8, se muestra el ejemplo de un vector de una fila con 3 datos almacenados uno en cada posición.

Figura 8. Vector de dimensiones 1x3 (1 fila y 3 columnas)

Fuente:(Joyanes, 2013)

Por el contrario, según (Joyanes, 2013), "una matriz se la puede considerar como un vector de vectores, es por consiguiente, un conjunto de elementos, todos del mismo tipo, en el cual el orden de los componentes es significativo" (p. 271). Ya que debido a que la memoria de un computador es lineal, una matriz debe estar linealizada para su disposición en el almacenamiento (Peter Kinz, 2017). El almacenamiento de las matrices en memoria se da de dos formas: orden de fila mayor y orden de columna mayor (Joyanes, 2013). En la Figura 9 y Figura 10, se presentan las dos formas de almacenamiento en una matriz; por filas y columnas.

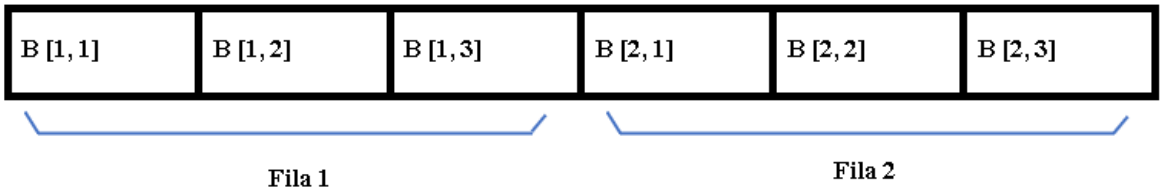


Figura 9. Orden de fila mayor

Fuente: (Joyanes, 2013)

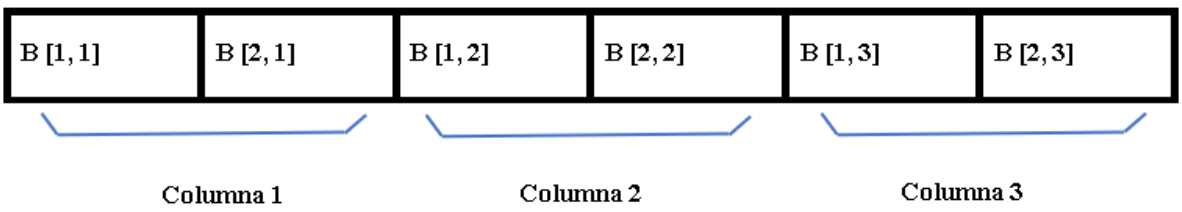


Figura 10. Orden de columna mayor

Fuente: (Joyanes, 2013)

2.4. Aprendizaje automático

El aprendizaje automático, se ha convertido en un componente clave de las soluciones de digitalización de la información, por su alta eficiencia y capacidad para la recolección de datos (Ray, 2019). Aprendizaje automático también es la capacidad de las máquinas para aprender; en donde se construye un sistema utilizando ciertos algoritmos, mediante los cuales el sistema puede tomar sus propias decisiones y proporcionar resultados inmediatos (Somvanshi, 2017).

Ahora bien; el aprendizaje automático visto desde el punto de vista de la clasificación, ayuda a la interconexión de diferentes dispositivos a través de internet, asegurando así el desarrollo generalizado de la inteligencia de los dispositivos y aplicaciones (K. Sharma & Nandal, 2019). Por otro lado, el aprendizaje supervisado es una técnica de construcción de modelos predictivos mediante la deducción de funciones a partir de datos de entrenamiento (Zhou, 2017). En este tipo de aprendizaje los algoritmos reciben los datos y estos son estudiados y analizados por el sistema; donde la categorización se realiza para que el algoritmo se pueda diferenciar correctamente entre los grandes conjuntos de datos (Somvanshi, 2017).

Por tanto, los algoritmos de clasificación supervisada operan usualmente sobre la información suministrada por un conjunto de datos de entrenamiento, donde estos datos se etiquetan con la clase o relación que representan basados en sus atributos (Fuentes Gómez, 2013). Para después ser analizados y estudiados logrando así un alto rendimiento de clasificación al producir una función que pueda ser utilizada para mapear nuevos datos (Somvanshi, 2017),

Los algoritmos se construyen mediante funciones; en donde las funciones son utilizadas para descomponer grandes problemas en tareas simples y para implementar operaciones que son comúnmente utilizadas durante un programa, para de esta manera reducir la cantidad de

código implementado (Velásquez, 2010). Los métodos más eficaces para el diseño de algoritmos se basan en el análisis del proceso de programación y en la etapa de diseño (Joyanes, 2013). El proceso de creación de algoritmos se muestra en la Figura 11, donde el proceso inicia en la programación del módulo, siguiendo con el entrenamiento del módulo para después comprobar sus funciones y depurar los errores encontrados.

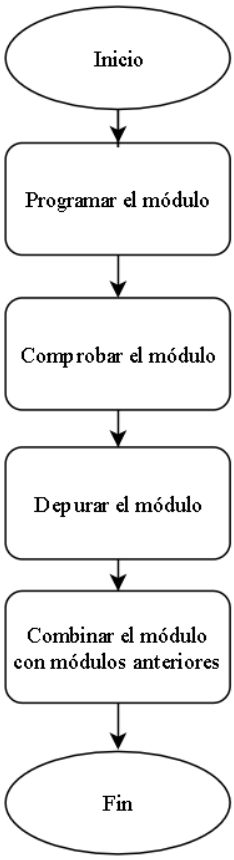


Figura 11. Diagrama de flujo de proceso de creación de un algoritmo.

Fuente: (Joyanes, 2013)

2.4.1. Algoritmos de clasificación a base de distancias.

Los métodos de clasificación a base de distancias utilizan una métrica que ayuda a la comparación de distancias entre los distintos objetos. Estas distancias se calculan en función de las diferentes clases; mientras estas sean más cercanas, las muestras tomadas pertenecerán a la misma clase (Araujo, 2006). El algoritmo k vecinos más cercano o "*k-Nearest Neighbours*" (k-NN), es uno de los métodos no paramétricos más críticos utilizados en la recuperación de

datos y tareas de similitud. (Lukač & Žalik, 2015). Este algoritmo requiere el cálculo de la distancia más corta entre los datos de entrenamiento y el dato a clasificar, para así identificar a que clase pertenece (Yigit, 2013). En la Figura 12, se presenta un ejemplo de clasificación del algoritmo k-NN, donde se observa las clases más cercanas al dato "Candidato", y así determinar a qué clase pertenece, dando como resultado que este pertenece a la clase de las estrellas.

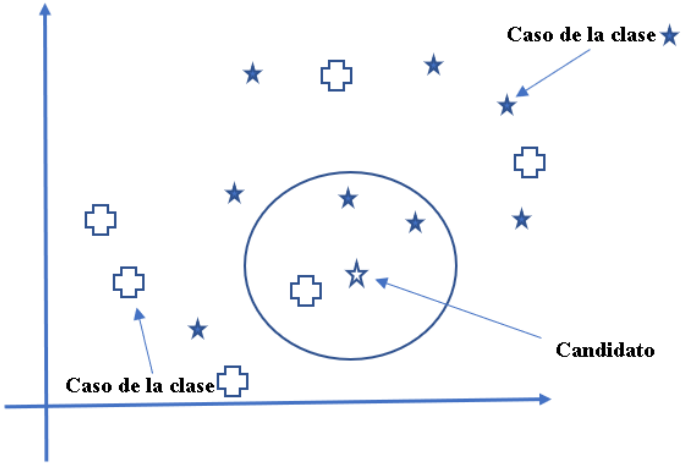


Figura 12. Ejemplo de k-NN

Fuente: (Lukač & Žalik, 2015)

Para la obtención de los k vecinos más cercanos, se calcula la distancia entre el dato a clasificar (candidato) $x = (x_1, \ldots . x_m)$ y todos los datos guardados $x_i = (x_{i1}, \ldots . x_{im})$. De acuerdo con (Benítez, 2013), la distancia euclidiana utilizada viene dada por la Ec.1.

$$de(x, x_i) = \sqrt{\sum_{i=1}^{n} (x_j - x_{ij})^2} \qquad \text{Ec.1}$$

2.4.2. Algoritmos de clasificación a base de probabilidad.

Los algoritmos de clasificación a base de probabilidad modelan un fenómeno mediante un conjunto de variables y las relaciones de dependencia entre ellas. Donde, a un objeto descrito por un conjunto de atributos se le asigna una de las clases posibles, donde la probabilidad de que la clase sea la correcta, se maximiza (Sucar, 2008). Así pues, el clasificador bayesiano parte del principio de probabilidad condicionada para estimar probabilidades de clasificación (Herrero, 2015), utilizando técnicas de minería de datos y aprendizaje automático. (Jahromi & Taheri, 2018). Este tipo de algoritmos de clasificación resuelve el problema de decisión utilizando una distribución probabilística (Mapayi & Tapamo, 2018). La Figura 13, muestra un ejemplo de una red bayesiana donde se representa la relación probabilística entre enfermedades y síntomas, donde dados los síntomas la red puede ser usada para predecir la presencia de alguna enfermedad.

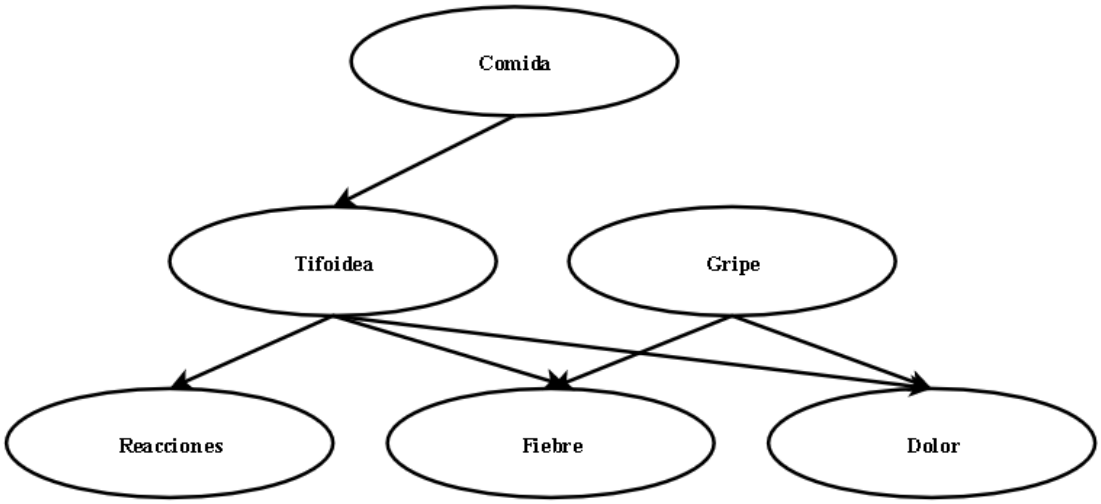


Figura 13. Ejemplo de una red bayesiana (Los nodos representan variables aleatorias y los arcos las relaciones de dependencia).

Fuente: (Araujo, 2006)

Este tipo de algoritmo clasifica nuevos datos $x=(x_1,..x_m)$ asignándole la clase k que maximiza la probabilidad condicional de la clase, dada la secuencia observada de atributos del

ejemplo. Como expresa (Benítez, 2013), en la Ec.2 los atributos P(k) y $P(x_i|k)$ se estiman a partir del conjunto de entrenamiento, utilizando las frecuencias relativas.

$$\boldsymbol{argmax\ P(k|x_1, \ldots . x_m) = argmax\ \frac{P(k|x_1, \ldots . x_m)P(k)}{P(k|x_1, \ldots . x_m)} \approx argmax\ P(k)} \qquad \text{Ec.2}$$

2.4.3. Algoritmos de clasificación a base de superplanos.

De acuerdo con Bohra & Palivela (2016), "los algoritmos de clasificación a base de superplanos son utilizados para analizar patrones y clasificar conjuntos de datos, construyendo un hiperplano que pueda separar las diferentes clases de los datos de entrenamiento". Para conseguir esto, es necesario contar con alguna función de *kernel* que transforme los datos de entrada, para posteriormente clasificarlos a partir de esas transformaciones (Araujo, 2006). Las máquinas de vectores de soporte o "*Support Vector Machines*" (SVM), son algoritmos de aprendizaje supervisado que pueden ser utilizados para analizar patrones y clasificar datos. La idea principal de como clasifica es construir un hiperplano que puede separar fácilmente los datos de entrenamiento dependiendo de la clase a la que pertenece (Bohra & Palivela, 2016). En la Figura 14, se presenta un ejemplo de hiperplano de separación con dos clases.

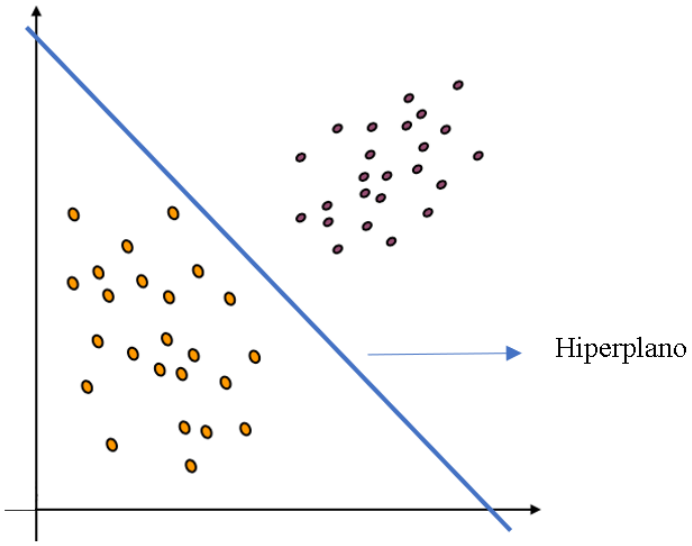


Figura 14. Hiperplano de separación con dos clases

Fuente: (Araujo, 2006)

2.5. Validación

Citando a Mathai (2019), “el desempeño y alcance de un algoritmo debe ser validado para verificar su nivel predictivo y si su utilización es factible o no”. La estimación del porcentaje de acierto de un algoritmo clasificador sirve para medir la capacidad de predicción sobre nuevos datos que lleguen en un futuro para que los pueda clasificar. La validación se realiza en los parámetros de exactitud, precisión y error (Araujo, 2006).

Un método de validación es la matriz de confusión, la cual es una tabla que muestra la relación entre los valores observados y los pronosticados en un problema de clasificación (Ruuska, 2018). Una matriz de confusión nos permite ver, mediante una tabla de contingencia la distribución de los errores cometidos por un clasificador en el proceso de predicción de nuevos datos (Araujo, 2006). Para el proceso de clasificación se necesita una validación para cualquier distribución de datos, y la matriz de confusión realiza una evaluación rigurosa de validez y error (Ruuska, 2018).

La validación cruzada, también es uno de los métodos de muestreo de datos que se utiliza para estimar el verdadero error de predicción de los modelos y así, poder ajustar los parámetros de clasificación (Berrar, 2018). Este método estima la precisión de predicción puntual fuera de la muestra a partir de un modelo probabilístico ajustado (Vehtari, 2017), basándose en la partición de la muestra en *k* subconjuntos del mismo tamaño y repitiendo el proceso de clasificación el mismo número de veces *k* de la partición, permitiendo así reducir el porcentaje de error de predicción de la muestra (Araujo, 2006).

Otro método de validación, son las curvas ROC, denominadas “*Receiver Operating Characteristic Curve*”, son una herramienta estadística que se utiliza para analizar la capacidad discriminante de una prueba al momento de clasificar (Cerda Lorca & Cifuentes, 2012). Es decir, una prueba, donde cuyo objetivo es clasificar a los individuos de una población basado

en diferentes clases. La curva que se grafica es el resultado de las medidas de sensibilidad y especificidad para cada valor umbral de las pruebas (Benavides, 2017). En la Figura 15, se observa un ejemplo de dos curvas ROC, trazadas mediante un método paramétrico y no paramétrico.

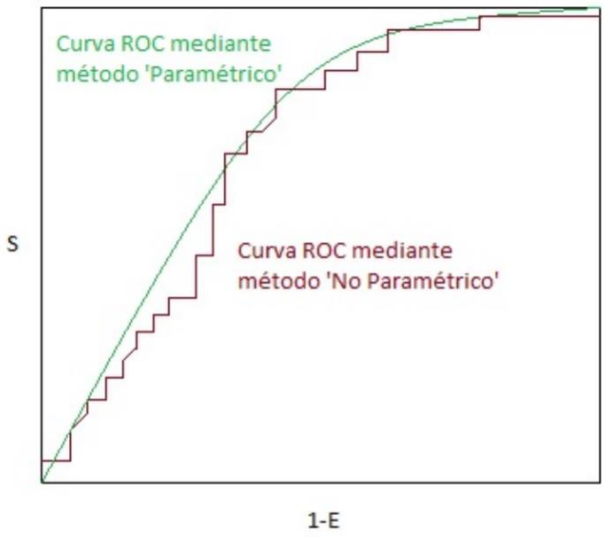


Figura 15. Ejemplo de curva ROC

Fuente: (Benavides, 2017)

El método paramétrico se enfoca en la distribución de la variable de decisión de la prueba (Kumar & Indrayan, 2011). Estos métodos tienen una gran capacidad discriminante permitiendo que las gráficas de las distribuciones no se solapen. Ya que, si estuviesen muy solapadas o, directamente fuesen iguales, la capacidad discriminante prácticamente sería nula (Benavides, 2017). Mientras, que los métodos no paramétricos para la construcción de la curva ROC, no hacen suposición alguna sobre la distribución de los resultados de la prueba en ambos grupos. Debido a esta característica este método tiene la propiedad de robustez, haciendo que la validación no sea exacta (Benavides, 2017).

Finalmente, con la implementación de las curvas ROC también se puede determinar el área bajo la curva, denominada AUC (*Area Under the Curve*), la cual se utiliza para evaluar el rendimiento del algoritmo de clasificación (Khan & Ali Rana, 2019). Midiendo la

probabilidad de clasificar correctamente un conjunto de datos pertenecientes a diversas clases, en donde, mientras el área sea mayor, el modelo tiene mejor porcentaje de clasificación (Khalid & Abrar, 2013). De acuerdo con (Khan & Ali Rana, 2019), el cálculo del área bajo la curva viene dado por la Ec. 3.

$$\boldsymbol{AUC = \left(\frac{B+b}{2}\right) * altura} \qquad \text{Ec.3}$$

2.6. Optimización de código

Como expresa Carrasco (2018.), "las técnicas de optimización de código mejoran el programa permitiendo que tengan un rendimiento mayor en tiempo de ejecución y espacio de memoria utilizado. Donde, las modificaciones que se realizan no deben cambiar el comportamiento u objetivo del código original". Como, por ejemplo, en la optimización de cadenas de caracteres, estas cadenas deben ser almacenadas en la memoria *Flash* y no en la memoria *RAM*; ya que al momento de mostrar mensajes en la ejecución del programa, se puede ocasionar consumo innecesario de la memoria RAM. (García, 2014). En la Figura 16, se muestra la manera optimizada de consumo de memoria utilizando la función F.

```
Serial.print("Hola Mundo");//Gasto de memoria RAM.
Serial.print(F("Hola Mundo"));//Gasto de Memoria Flash.
```

Figura 16. Función F en Arduino

Fuente: (García, 2014)

La optimización de bucles también es necesaria, ya que habitualmente un programa pasa la mayor parte del tiempo en la ejecución repetitiva de una parte pequeña del código. El objetivo principal de estas técnicas es disminuir estos bucles y mejorar el tiempo de ejecución del programa. El siguiente punto es la detección de código muerto, ya que este es el código que nunca se utilizará y que está ralentizando la memoria del sistema donde se está corriendo el programa (Carrasco, 2018); Como plantea Alabwaini (2018), "el objetivo principal es eliminar

expresiones redundantes porque estas declaraciones inútiles afectan negativamente a la calidad del código". El enfoque principal para el reconocimiento de código muerto consiste en un procedimiento iterativo manual, mediante el soporte de estructuras de control denominadas pseudocódigos, las cuales permiten identificar que clases pueden ser eliminadas por completo del algoritmo sin que afecte su funcionamiento (Streitel et al., 2014).

CAPÍTULO III. DISEÑO

En este capítulo, se llevará a cabo la revisión y el análisis de las bases de datos con las cuales se van a trabajar, en donde se evaluará el desempeño de los algoritmos k-NN, SVM y Clasificador Bayesiano presentados en el capítulo anterior. Para así tener una mejor idea de cómo están clasificando, a partir de un conteo de aciertos y errores,

3.1. Adquisición de bases de datos

Para el análisis de los Algoritmos de Clasificación Supervisada se hace el uso de cinco bases de datos, pertenecientes al muestreo realizado por SE, las cuales están aplicadas a diferentes áreas. Las bases de datos fueron seleccionadas del repositorio *IEEE DataPort*. En la Tabla 6, se muestra el nombre y la descripción de cada una de las bases de datos.

Tabla 6. Bases de datos

Nombre	**Descripción**	**Autor**
BDD1	Conjunto de datos obtenidos para la detección del tipo de pisada.	(Fuentes, 2019)
BDD2	Conjunto de datos de posición corporal obtenidos con un acelerómetro, para determinar la posición del cuerpo.	(Rosero, 2018a)
BDD3	Conjunto de datos obtenidos del ambiente de un cultivo de rosas de invernadero.	(Champutiz, 2019)
BDD4	Conjunto de datos obtenidos mediante sensores y actuadores de posición de presión colocados en una silla de ruedas, para la detección de posibles problemas de salud.	(Rosero, 2018b)

BDD5	Conjunto de datos obtenidos mediante sensores para la detección de alcohol en conductores.	(Portilla, 2018)

Fuente: Autoría

3.1.1. BDD1: Conjunto de datos del tipo de pisada.

Estos datos fueron obtenidos mediante sensores de presión que se encuentran colocados en tres zonas de cada pie. Los valores de cada sensor fueron leídos por las entradas analógicas de un arduino mega 2560 (Fuentes, 2019). En la Figura 17, se observa la ubicación de los sensores de presión para el pie izquierdo y pie derecho, los cuales están representados por círculos negros.

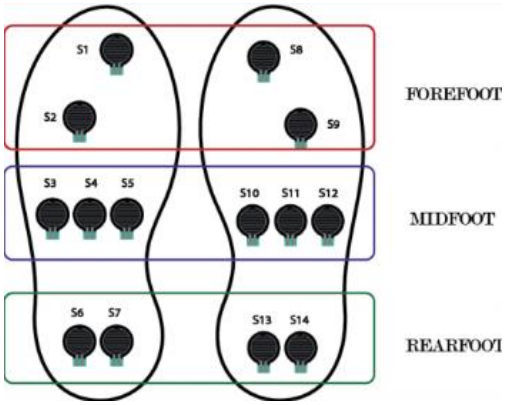


Figura 17. Diseño y ubicación de los sensores de presión

Fuente: (Fuentes, 2019)

3.1.2. BDD2: Conjunto de datos de posición corporal con un acelerómetro.

Este conjunto de datos fue obtenido mediante un sistema electrónico, diseñado para conocer la posición del cuerpo humano mediante un acelerómetro de tres ejes para detectar cinco posiciones comunes: decúbito ventral, decúbito lateral derecho, decúbito lateral izquierdo, decúbito supino y sentado. Los datos de este sensor se adquirieron con diez personas diferentes (Rosero, 2018a)). En la Figura 18, se observa el diseño del sistema de posición corporal, en donde el circuito se ajusta en el pecho de la persona.

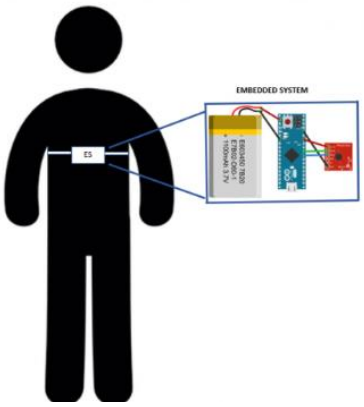


Figura 18. Diseño del Sistema de Posición Corporal

Fuente: (Rosero, 2018a)

3.1.3. BDD3: Conjunto de datos de un cultivo de rosas de invernadero.

La adquisición de datos se la realizó con un robot autónomo que incorpora diferentes sensores, con los cuales se recopiló información sobre el cultivo de rosas en invernaderos como: humedad del suelo, luz, temperatura y CO2 (Champutiz, 2019). En la Figura 19, se observa el diseño y ubicación de los sensores de humedad, luz, temperatura y CO2.

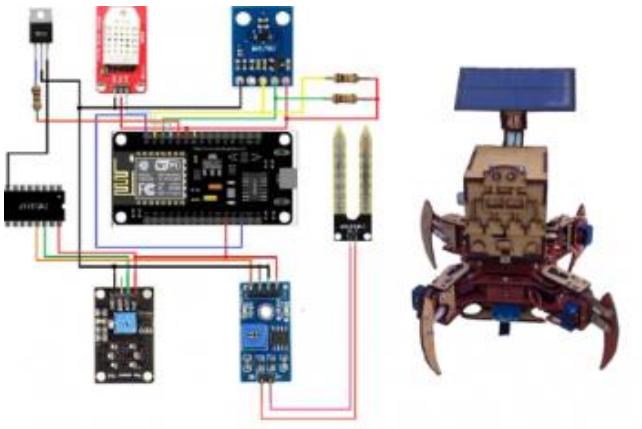


Figura 19. Diseño y ubicación de los sensores para la adquisición de datos de los cultivos de rosas de invernadero.

Fuente: (Champutiz, 2019)

3.1.4. BDD4: Conjunto de datos de sensores de una silla de ruedas.

Los datos fueron obtenidos mediante sensores y actuadores de posición de presión, los cuales están representados por círculos rojos grandes y azules en el asiento de la silla de ruedas,

respectivamente (Rosero, 2018). En la Figura 20, se observa que los círculos negros muestran sensores de ultrasonido en el respaldo, las flechas amarillas muestran el periférico de salida y las flechas rojas indican la entrada periférica en el convertidor analógico-digital (Rosero, 2018).

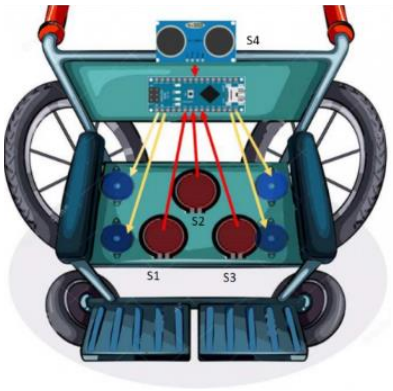


Figura 20. Diseño del sistema de sensores y actuadores de posición

Fuente: (Rosero, 2018)

3.1.5. BDD5: Conjunto de datos para la detección de alcohol en conductores.

Para la recopilación de datos, se utilizó un sensor de concentración de alcohol en el ambiente, un sensor que mide la temperatura de los puntos definidos en la cara del conductor y un sensor que permite identificar y reconocer el grosor de la pupila (Portilla, 2018). En la Figura 21, se observa el diseño del sistema de detección de alcohol en conductores.

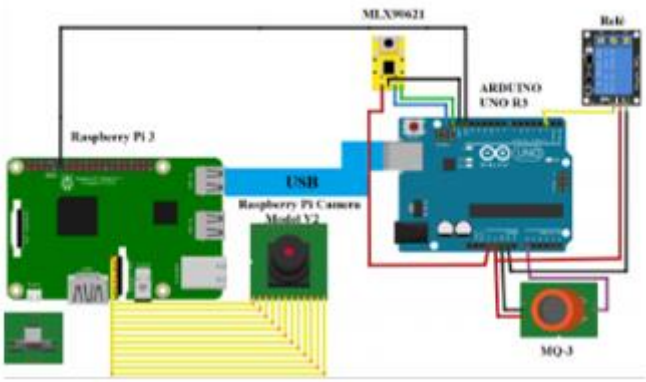


Figura 21. Diseño del sistema de detección de alcohol en conductores.

Fuente: (Portilla, 2018)

3.2. Pseudocódigos

Para la implementación de los Algoritmos de Clasificación Supervisada se hace el análisis de sus pseudocódigos. En las Figuras 22-24, se muestra el pseudocódigo del algoritmo clasificador k-NN, clasificador Bayesiano y SVM, respectivamente.

INICIO
Como entrada se dispone de:
El modelo, conteniendo M casos, uno por cada clase $(x_i, \theta_i), i = 1, \dots, M$,
y un nuevo caso (x, θ) que se desea clasificar

PARA cada caso del modelo (x_i, θ_i) **HACER**
INICIO
Calcular la distancia a **x** del caso
Sea D, dicha distancia
FIN
Como salida, dar la clase θ, cuya distancia D, es
la mínima de entre todas las clases
FIN

Figura 22.Pseudocódigo del algoritmo k-NN

Fuente: (Araujo, 2006)

INICIO
Como entrada se dispone de:
El modelo y un conjunto de datos de entrenamiento T, donde
$F = (f_1, f_2, f_3, .., f_n)$, son los valores de las variables predictoras en el
conjunto de datos de prueba

PARA cada conjunto de datos **HACER**
INICIO
Leer el conjunto de datos de entrenamiento T
Calcular la desviación estándar y media de la variable predictora en cada clase
REPETIR
Calcular la probabilidad de f_1 usando la ecuación de densidad de gauss en
cada clase
Hasta que la probabilidad de todas las variables predictoras $(f_1, f_2, f_3, .., f_n)$,
hayan sido calculadas
CALCULAR la probabilidad para cada clase
FIN
Como salida, se obtiene F, la cual es la mayor probabilidad
FIN

Figura 23. Pseudocódigo del algoritmo clasificador Bayesiano

Fuente: (Araujo, 2006)

INICIO

Como entrada se dispone de:

El modelo y un conjunto de datos T, donde "x" & "y", son los datos etiquetados de entrenamiento y vector de soporte $\alpha = 0$

PARA cada conjunto de datos **HACER**

INICIO

Leer el conjunto de datos de entrenamiento T

Calcular valores máximos y mínimos

REPETIR

Para todos los valores (x_i, y_i),

Optimizar α_i *y* α_j

Hasta que no se cumplan cambios en α

FIN

Como salida, se obtiene el vector de soporte $(\alpha_i > 0)$

FIN

Figura 24. Pseudocódigo del algoritmo SVM

Fuente: (Araujo, 2006)

3.3. Metodología de funcionamiento

En la Figura 25, se observa la metodología de funcionamiento para la validación de los algoritmos de clasificación. Donde, el proceso inicia cargando cada una de las cinco bases de datos las cuales se encuentran en formato "csv", luego se procede a dividir la base de datos en datos de entrenamiento y datos de prueba. A continuación, se procede a cargar los datos de entrenamiento en los algoritmos de clasificación, para así entrenarlos. Luego usando los algoritmos ya entrenados cargamos los datos de prueba para poder obtener basados en las métricas de rendimiento, los resultados de clasificación de cada uno de los algoritmos.

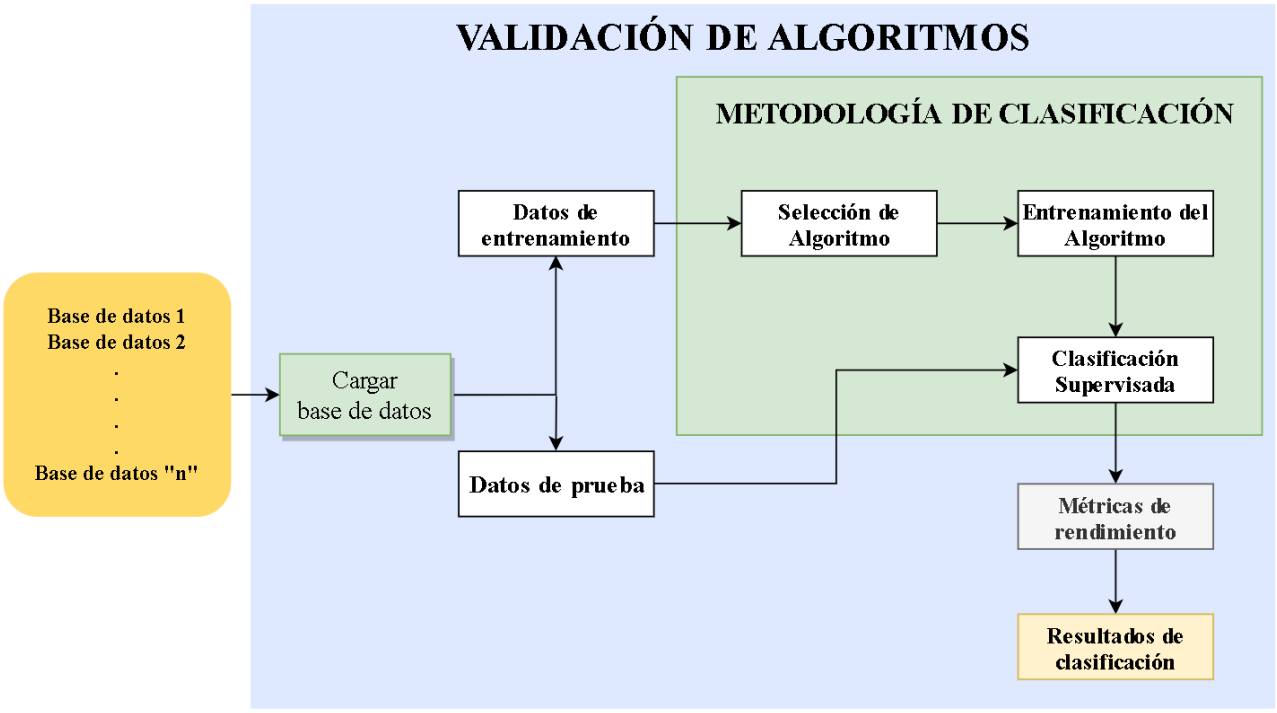


Figura 25. Pseudocódigo del algoritmo SVM

Fuente: Autoría

3.4. Algoritmo k-NN

De acuerdo con Liang (2015), "el algoritmo k-NN se encuentra entre los diez algoritmos de clasificación de minería de datos más eficaces y eficientes", pero la principal limitación del algoritmo k-NN es su tiempo de clasificación frente a grandes conjuntos de datos. Pero esto puede ser solucionado con una óptima implementación del algoritmo (Pawlovsky & Matsuhashi, 2017).

La optimización del algoritmo de clasificación k-NN empieza desde la importación de librerías, ya que en la opinión de (V. Sharma, 2019), las librerías necesarias para la implementación del algoritmo se muestran en la Figura 26, donde las librerías "scipy,spatial", "sklearn" y "sklearn.model_selection"; son de uso innecesario, ya que son librerías utilizadas para el uso de etiquetas en metadatos y además contienen otros algoritmos de clasificación que no están siendo utilizados. Sin embargo, estas librerías pueden ser reemplazadas por la librería

“math” y “random”, ya que solo se necesita la herramienta de raíz cuadrada y números randómicos en la implementación del algoritmo.

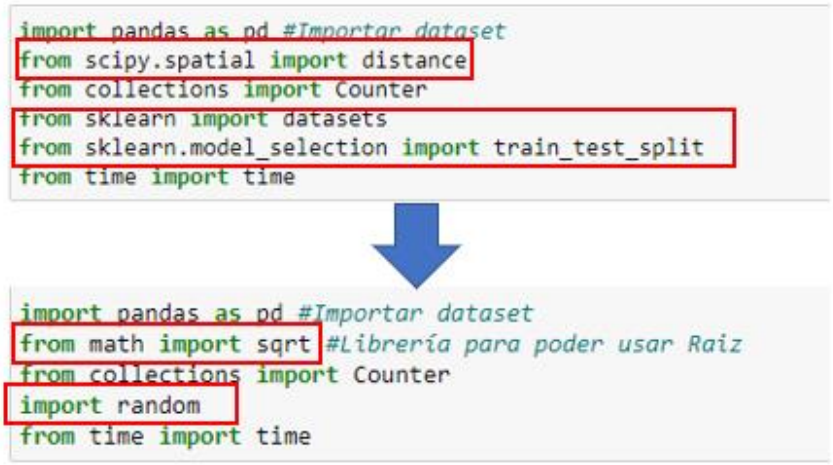


Figura 26. Importación de librerías algoritmo k-NN

Fuente: (V. Sharma, 2019)

Como plantea (V. Sharma, 2019), para la división de los datasets en datos de entrenamiento y en datos de prueba, se usa la librería “sklearn.model_selection” y “train_test_split”; la cuales cumplen con la función de dividir el modelo, pero ralentizan el procesamiento de las bases de datos porque ocuparán espacio innecesario en la memoria del sistema embebido. Sin embargo, esto puede optimizarse al implementar una clase que únicamente se encargue de la división de la base de datos y sustituya a las librerías antes mencionadas; esta implementación se presenta en la Figura 27.

Figura 27. División en datos de entrenamiento y datos de prueba

Fuente: (V. Sharma, 2019)

Finalmente, citando a (V. Sharma, 2019), la clasificación empieza con el llamado de la librería “sklearn”, la cual contiene varios algoritmos de clasificación. En este caso solamente se usa el algoritmo de vecinos más cercanos k-NN, pero para obtener una mayor rapidez al momento de procesar la información se procede a implementar solo la función del cálculo de distancias, la cual se aprecia en la Figura 28, con esto se evita que el algoritmo cargue con librerías innecesarias, optimizando así el tiempo de procesamiento.

Figura 28. Función cálculo de distancias k-NN

Fuente: (Benítez, 2013)**3.5. Clasificador Bayesiano**

El clasificador Bayesiano supera la precisión de clasificación de otros clasificadores estándar bien conocidos (Rahman & Qamar, 2016), ya que este algoritmo minimiza la probabilidad de error al momento de clasificar (Salih, 2019).

El clasificador Bayesiano, según (Seif, 2017), necesita de las librerías "sklearn.naive_bayes" y "GaussianNB"; las cuales contienen herramientas que no se usan en su implementación y se las considera como código muerto. Y esto puede ocasionar que el algoritmo no tenga un óptimo desempeño. De acuerdo con (Askaruly,2018), para la optimización de este código se procedió a la implementación solamente de la ecuación de probabilidad bayesiana, la cual se usará para la predicción al momento de clasificar. En la Figura 29, se observa la implementación de la ecuación.

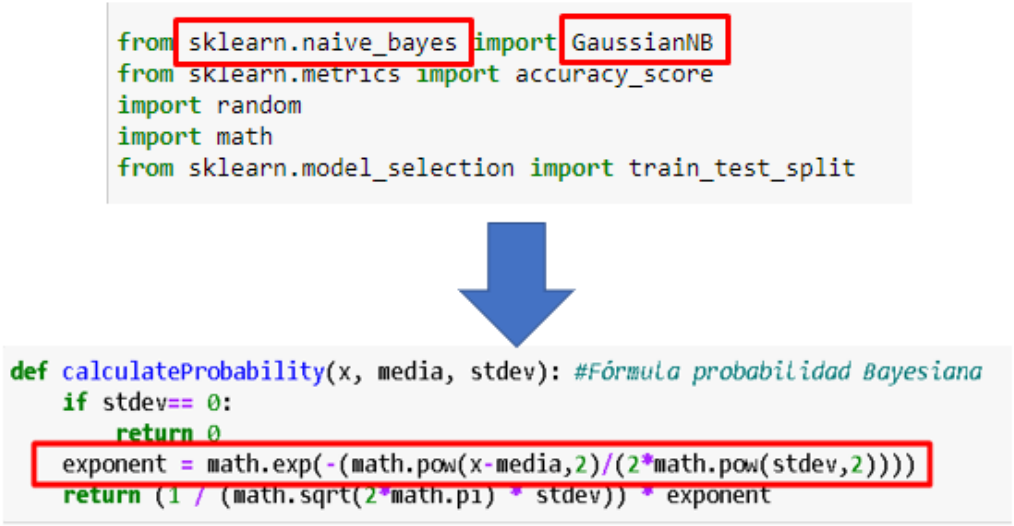


Figura 29. Fórmula para el cálculo de la Probabilidad Bayesiana

Fuente: (Askaruly, 2018), (Seif, 2017)

Citando a (V. Sharma, 2019), para la división de los datasets en datos de entrenamiento y en datos de prueba en el Clasificador Bayesiano, se usa la librería "sklearn.model_selection" y "train_test_split". Y de acuerdo con (Askaruly, 2018), estas se pueden reemplazar con la

implementación una clase, que únicamente se encargue de la división de la base de datos. Esta implementación se presenta en la Figura 27, ya que también se la hizo en el algoritmo k-NN:

Como plantea (Askaruly,2018), en el cálculo de la predicción se utilizó la técnica de optimación de bucles, separando este proceso en dos funciones. El objetivo principal de esta técnica es disminuir los bucles redundantes y mejorar el tiempo de ejecución del programa. Esto se puede observar en la Figura 30 y 31, donde se separó en una función para calcular la predicción y otra para calcular el porcentaje de predicción.

```
def ob_predicciones(summaries, testSet): #calculamos la predicción
    predictions = []
    for i in range(len(testSet)):
        result = predict(summaries, testSet[i])  #Llamamos a la función de predicción
        predictions.append(result)
    return predictions
```

Figura 30. Técnica de Optimización de Bucles -Función para calcular la predicción

Fuente: (Askaruly, 2018)

```
def tasa_pred(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100
```

Figura 31. Técnica de Optimización de Bucles -Función para calcular el porcentaje de predicción

Fuente: (Askaruly, 2018)

3.6. Algoritmo SVM

Dai (2018), plantea que “el algoritmo SVM acorta el tiempo de clasificación cuando la distancia euclidiana es muy grande entre los diferentes datos”. De manera que el porcentaje de error sería el mínimo, logrando así una mayor precisión al momento de clasificar (Tian & Wang, 2017).

Como lo hace notar (Syuriadi, 2018), en la implementación del algoritmo SVM se utiliza una de las librerías más robustas, como es el caso de “sklearn.svm” y “SVC”; las cuales permiten clasificar mediante la construcción de hiperplanos, esta función se la muestra en la Figura 32a. Mientras que para la optimización del algoritmo, se procedió a la detección de código muerto y solo se implementa las funciones necesarias para la clasificación. Esto se observa en la Figura 32b.

```
import csv
from sklearn import svm
from sklearn.metrics import classification_report
```

Figura 32a. Librería del algoritmo de clasificación SVM

Fuente: (Syuriadi, 2018)

```
#Función para calcular el costo de  SVM
def cost(theta,c,x,y):
    cost = 0.0
    for i in range(len(x)):
        z = ThetaTX(theta[c], x[i])
        cost += y[i]*LinearSVM_cost1(z) + (1 - y[i])*LinearSVM_cost0(z)
    return cost
```

```
def gradDescent(theta,c,x,y,learning_rate):
    oldTheta = theta[c]
    for Q in range(len(theta[c])):
        derivative_sum = 0
        for i in range(len(x)):
            derivative_sum += (sigmoid(ThetaTX(oldTheta,x[i])) - y[i])*x[i][Q]
        theta[c][Q] -= learning_rate*derivative_sum
```

Figura 32b. Función para calcular y entrenar el algoritmo SVM

Fuente: (Syuriadi, 2018)

Finalmente, como lo hace notar también (Thukral, 2017), así como en los casos anteriores (k-NN y Clasificador Bayesiano), la división de los datasets en datos de entrenamiento y prueba, es mucho más óptima con la implementación de una clase que con la

utilización de una librería de “sklearn.model_selection”. Esta implementación se muestra en la Figura 33.

```
def cross_val_split(data_X,data_Y,test_size,seed_val):
    data_x = data_X.tolist()
    data_y = data_Y.tolist()
    seed(seed_val)
    train_size = floor((1 - test_size)*len(data_x))
    train_x = []
    train_y = []
    while(len(train_x)<train_size):
        index = randrange(len(data_x))
        train_x.append(data_x.pop(index))
        train_y.append(data_y.pop(index))
    return train_x,train_y,data_x,data_y
```

Figura 33. División en datos de entrenamiento y datos de prueba

Fuente: (V. Sharma, 2019), (Thukral, 2017)

3.7. Cálculo del porcentaje de aciertos, tamaño del algoritmo y tiempo de ejecución.

Luego de la división de la data set en datos de entrenamiento y prueba se procede al entrenamiento del algoritmo y cálculo del porcentaje de aciertos. El funcionamiento del cálculo de acierto se explica en la Tabla 7.

Tabla 7.Cálculo del porcentaje de aciertos

Secuencia de la clase	**Descripción**
1	Carga de la base de datos `df = pd.read_csv("training_vision.csv")`

2	División de la base de datos en datos de entrenamiento y prueba. El 0.3 indica el porcentaje de la base de datos destinada a datos de prueba (30%).
	`splitRatio = 0.3`
	`trainingSet, testSet = dividir_datos(dataset, splitRatio)`

3	Se carga los datos divididos en el algoritmo. El cual empieza a entrenar con los datos de entrenamiento, después empiezan a ingresar los datos de prueba, y este clasifica y predice su etiqueta. Y la compara con los datos de entrenamiento. `trainingSet,testSet = algorithm(accuracy)`

4	Si la predicción fue correcta, el contador del total de aciertos aumenta en uno. Al finalizar la comprobación se procede a la división del total de acierto con el total de datos de prueba y se obtiene el porcentaje de clasificación del algoritmo.
	`for x in range(len(testSet)):`
	`  if testSet[x][-1] == predictions[x]:`
	`  correct += 1`
	`return (correct/float(len(testSet))) * 100`

Fuente: (V. Sharma, 2019), (Askaruly, 2018)

En cambio, para el cálculo del tiempo se colocó un contador al inicio y al final de la implementación, para después hacer la diferencia entre los dos tiempos, y obtener el tiempo de ejecución, el cual se muestra en la Figura 34.

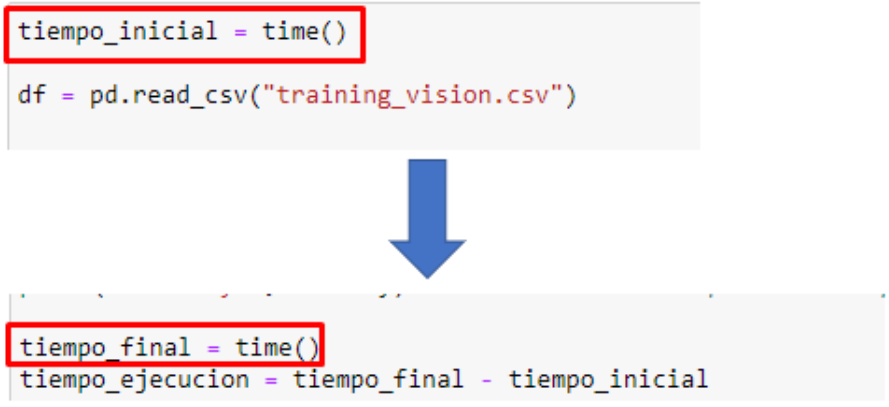


Figura 34. Cálculo tiempo de ejecución

Fuente: Autoría

En cuanto a la adquisición de datos del tamaño de cada algoritmo, este valor se obtiene midiendo el tamaño del archivo que contiene toda la programación, en la Figura 35, se presenta la manera de visualización del tamaño del algoritmo k-NN implementado y de librería sin optimizar.

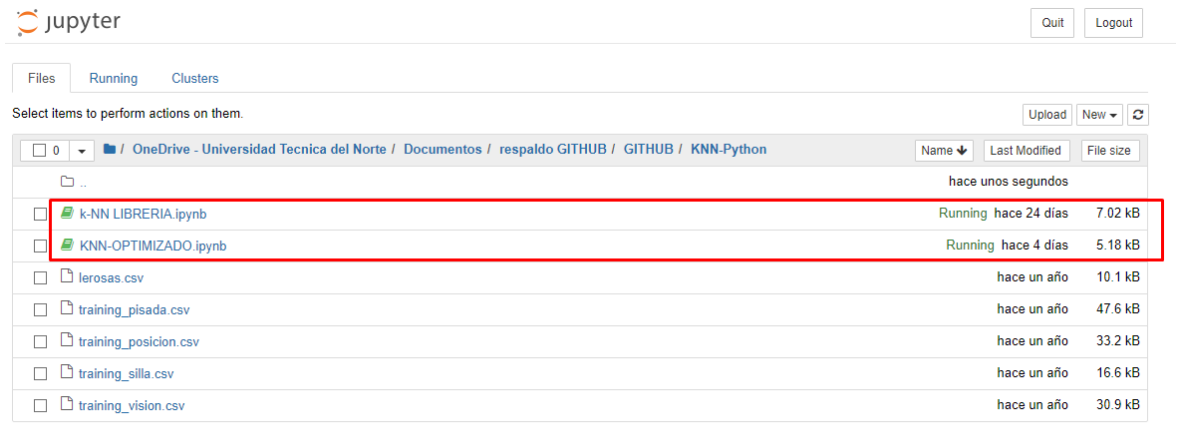


Figura 35. Tamaño de algoritmos k-NN optimizado y librería

Fuente: Autoría

Finalmente, en la Figura 36 se presenta el formato de visualización de los resultados de porcentaje de aciertos y tiempo de ejecución, como también el número de datos de entrenamiento y prueba.

```
Datos= 312 ,Datos de entrenamiento= 93 ,Datos de test= 219
Porcentaje de aciertos: 100.0
El tiempo de ejecucion fue: 62.99877166748047 milisegundos
```

Figura 36. Visualización de resultados

Fuente: Autoría

CAPÍTULO IV. RESULTADOS

En este capítulo, se lleva a cabo las pruebas de los algoritmos de clasificación con cada base de datos, además, de la tabla resumen en la que se encuentra el total de porcentajes de precisión obtenidos de las pruebas realizadas. La validación consta de 10 pruebas controladas, en donde, para la realización de cada una se utiliza un conjunto diferente de datos de entrenamiento por cada uno de los algoritmos implementados y de librería. Los criterios para evaluar a los algoritmos de clasificación son tasa de error, rapidez, tamaño y rendimiento, en donde se registra el porcentaje de aciertos, tiempo de clasificación al momento de realizar cada iteración y tamaño de cada uno de los algoritmos.

Posteriormente, para determinar el rendimiento se implementó las Curvas ROC y el cálculo del AUC, utilizando los resultados obtenidos de las pruebas controladas, donde estos se ordenaron dependiendo del porcentaje de clasificación. El capítulo finaliza con las conclusiones y recomendaciones del presente trabajo de titulación.

4.1. Criterios de Evaluación: Tasa de Error

Inicialmente, la validación empieza con la realización de 10 pruebas controladas, en donde, en cada iteración se obtiene el porcentaje de aciertos y errores al terminar la clasificación, el proceso se lo presenta en la Tabla 7. Para el porcentaje total de aciertos se emplea la media aritmética, aplicada en todas las iteraciones realizadas en cada validación.

4.1.1. Tasa de Error: Algoritmos implementados.

En la Tabla 8, se presenta los resultados obtenidos de las cinco bases de datos utilizando el algoritmo k-NN implementado.

Tabla 8. Tasa de error algoritmo k-NN

Base de Datos	Número de Datos	Porcentaje de acierto promedio	Tasa de Error
BDD1	319	99.06%	0.94%
BDD2	530	99.24%	0.76%
BDD3	240	99.58%	0.42%
BDD4	246	100%	0%
BDD5	312	99.36%	0.64%

Fuente: Autoría

En la Tabla 9, se puede ver los resultados obtenidos de las cinco bases de datos utilizando el algoritmo Clasificador Bayesiano.

Tabla 9. Tasa de error algoritmo Clasificador Bayesiano

Base de Datos	Número de Datos	Porcentaje de acierto promedio	Tasa de Error
BDD1	319	84.35%	15.65%
BDD2	530	100%	0%
BDD3	240	92.75%	7.25%
BDD4	246	99.6%	0.4%
BDD5	312	99.48%	0.52%

Fuente: Autoría

En la Tabla 10, muestra los resultados obtenidos de las cinco bases de datos utilizando el algoritmo SVM.

Tabla 10.Tasa de error algoritmo SVM

Base de Datos	Número de Datos	Porcentaje de acierto promedio	Tasa de Error
BDD1	319	93.18%	6.82%
BDD2	530	97.96%	2.04%
BDD3	240	94.57%	5.43%
BDD4	246	97.24%	2.76%
BDD5	312	89.92%	10.08%

Fuente: Autoría

4.4.2. Tasa de Error: Algoritmos de librería no optimizados.

En la Tabla 11, se puede ver los resultados obtenidos de las cinco bases de datos utilizando el algoritmo k-NN de librería.

Tabla 11. Tasa de error algoritmo k-NN

Base de Datos	Número de Datos	Porcentaje de acierto promedio	Tasa de Error
BDD1	319	98.82%	1.18%
BDD2	530	94.81%	5.19%
BDD3	240	96.87%	3.13%
BDD4	246	96.95%	3.05%
BDD5	312	99.6%	0.4%

Fuente: Autoría

En la Tabla 12, se presenta los resultados obtenidos de las cinco bases de datos utilizando el algoritmo Clasificador Bayesiano de librería.

Tabla 12. Tasa de error algoritmo Clasificador Bayesiano

Base de Datos	Número de Datos	Porcentaje de acierto promedio	Tasa de Error
BDD1	319	99.6%	0.4%
BDD2	530	100%	0%
BDD3	240	100%	0%
BDD4	246	100%	0%
BDD5	312	95.19%	4.81%

Fuente: Autoría

En la Tabla 13, se puede ver los resultados obtenidos de las cinco bases de datos utilizando el algoritmo SVM de librería.

Tabla 13. Tasa de error algoritmo SVM

Base de Datos	Número de Datos	Porcentaje de acierto promedio	Tasa de Error
BDD1	319	99.21%	0.79%
BDD2	530	99.76%	0.24%
BDD3	240	90.10%	9.9%
BDD4	246	95.43%	4.57%
BDD5	312	99.2%	0.08%

Fuente: Autoría

4.5. Criterios de evaluación: Rapidez

En este criterio de evaluación se mide el tiempo que el algoritmo emplea para construir el modelo y clasificar las bases de datos. La medición del tiempo empieza desde el momento en el que el algoritmo empieza a cargar las librerías hasta finalizar con cada clasificación y mostrar el porcentaje total de aciertos.

4.5.1. Rapidez: Algoritmos implementados.

En la Tabla 14, se evidencia el tiempo de ejecución en el proceso de clasificación realizada por los algoritmos k-NN, Clasificador Bayesiano y SVM.

Tabla 14. Tiempo de Clasificación algoritmos implementados

Base de Datos	k-NN	Clasificador Bayesiano	SVM
BDD1	46.9 ms	25.92 ms	23.12 ms
BDD2	94.75 ms	27.92 ms	24.58 ms
BDD3	37.9 ms	11.96 ms	9.86 ms
BDD4	25.93 ms	16.955 ms	12.08 ms
BDD5	46.87 ms	12.96 ms	18.06 ms

Fuente: Autoría

4.5.2. Rapidez: Algoritmos de librería no optimizados.

En la Tabla 15, se presenta el tiempo de ejecución en el proceso de clasificación realizada por los algoritmos k-NN, Clasificador Bayesiano y SVM.

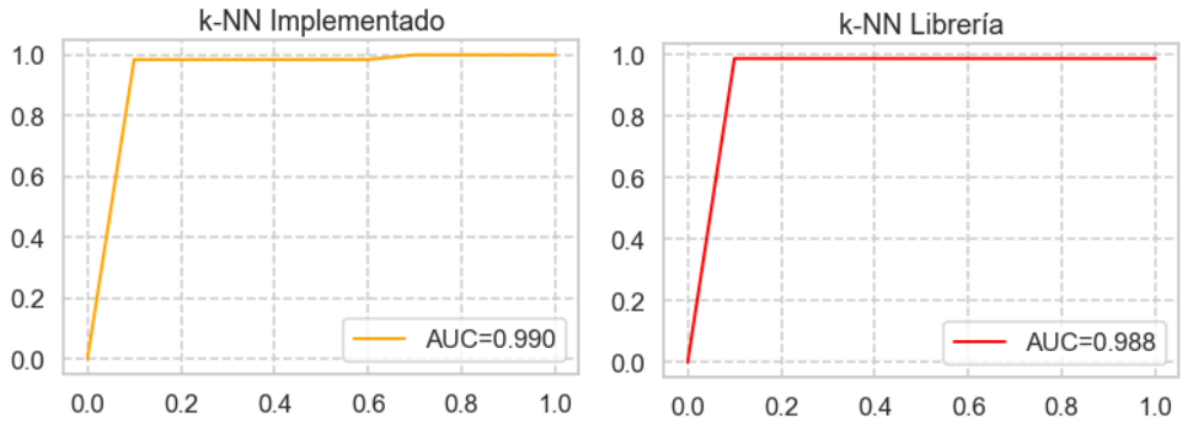
Tabla 15. Tiempo de Clasificación algoritmos de librería

Base de Datos	NN	Clasificador Bayesiano	SVM
BDD1	486.1 ms	10.12 ms	11.93 ms
BDD2	1307.8 ms	8.97 ms	11.96 ms
BDD3	383.95 ms	13.97 ms	10.97 ms
BDD4	322.47 ms	10.97 ms	10 ms
BDD5	428.79 ms	9.98 ms	7.97 ms

Fuente: Autoría

4.6. Curvas ROC

A continuación, en la Figura 37, se muestra la implementación de las curvas ROC y el cálculo del AUC de los algoritmos implementados y de librería sin optimizar, según los datos obtenidos en las pruebas de clasificación de la base de datos "BDD1", se observa que las curvas de los algoritmos k-NN tienen la mayor área, por lo que son los mejores clasificadores de cada grupo.



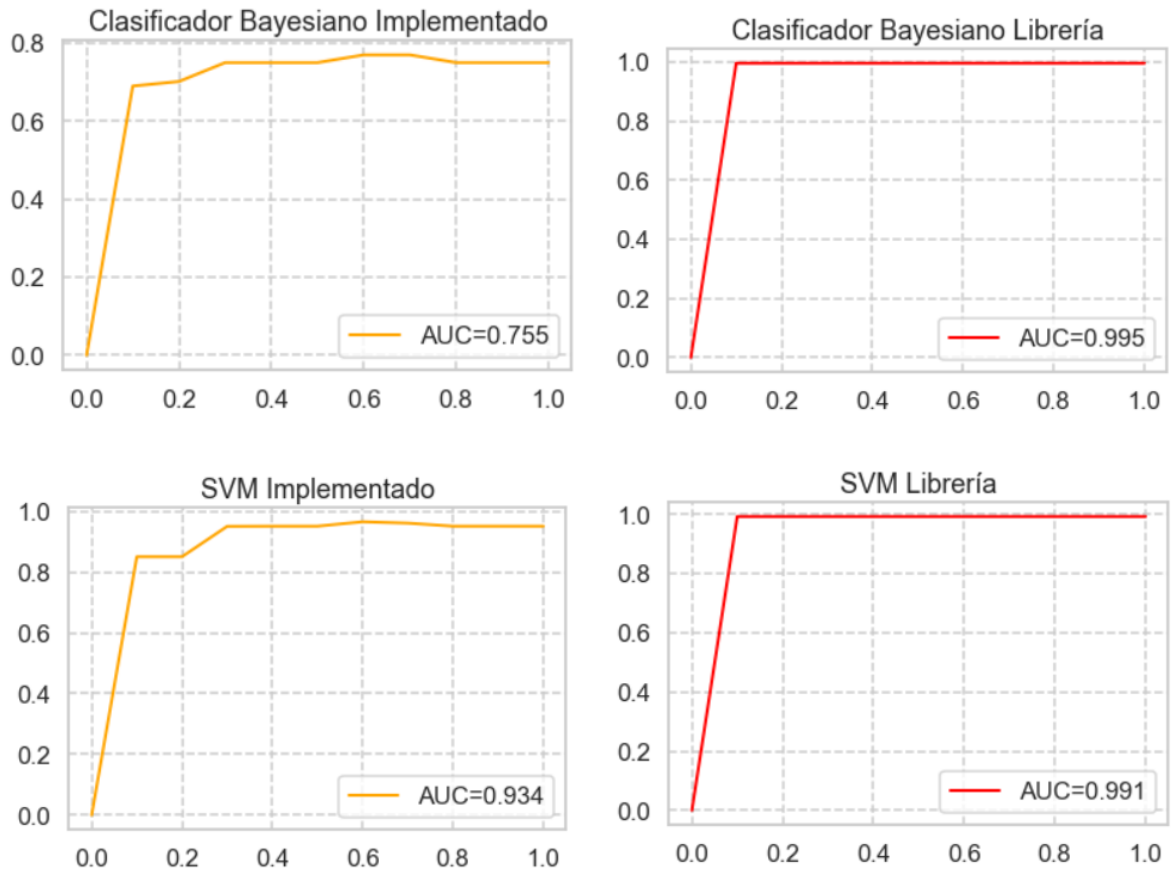


Figura 37.Curvas ROC - Análisis de algoritmos Implementados y de Librería con la base de datos "BDD1"

Fuente: Autoría

En la Figura 38, se muestra la implementación de las Curvas ROC y el cálculo del AUC de los algoritmos implementados y de librería sin optimizar, según los datos obtenidos en las pruebas de clasificación de la Base de Datos “BDD2”, muestra que las curvas de los algoritmos Clasificador Bayesiano son los mejores clasificadores de cada grupo, ya que tienen la mayor área bajo la curva.

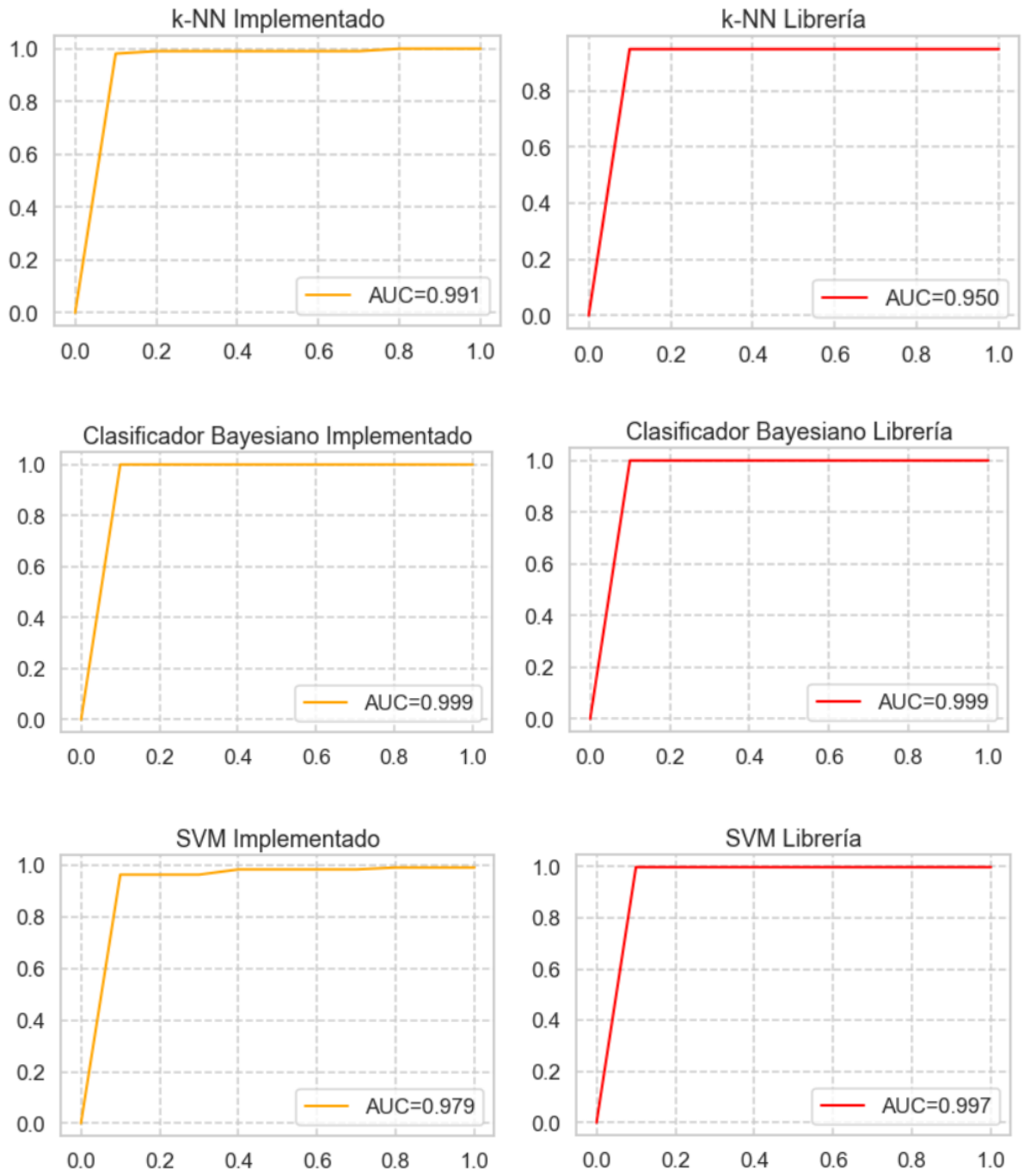


Figura 38. Curvas ROC - Análisis de algoritmos Implementados y de Librería con la Base de Datos "BDD2"

Fuente: Autoría

En la Figura 39, se muestra la implementación de las Curvas ROC y el cálculo del AUC de los algoritmos implementados y de librería sin optimizar , según los datos obtenidos en las pruebas de clasificación de la Base de Datos “BDD3”, se observa que las curvas de los

algoritmos k-NN implementado y Clasificador Bayesiano sin optimizar tienen la mayor área, por lo que son los mejores clasificadores de cada grupo.

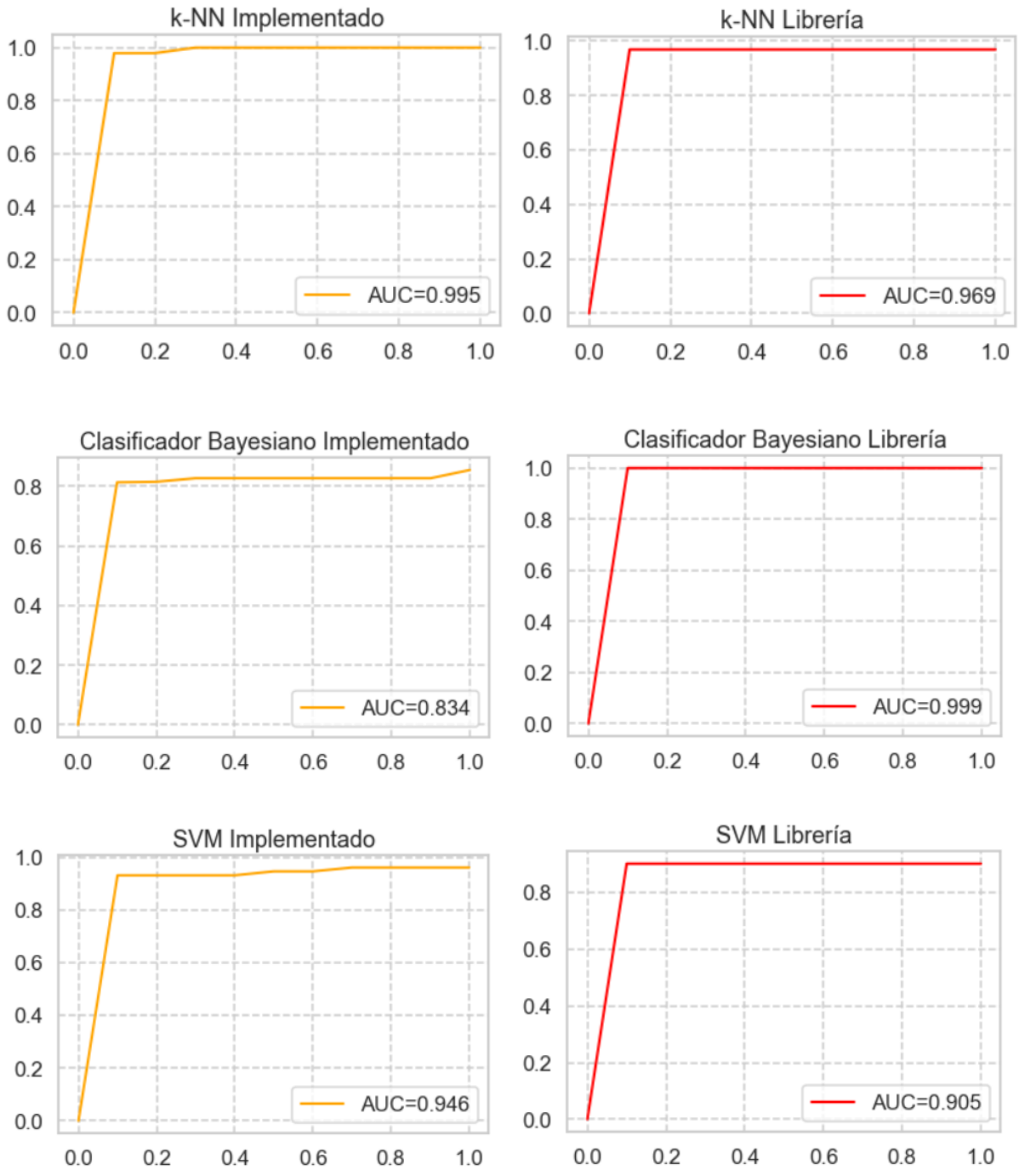


Figura 39. Curvas ROC - Análisis de algoritmos Implementados y de Librería con la Base de Datos "BDD3"

Fuente: Autoría

En la Figura 40, se muestra la implementación de las Curvas ROC y el cálculo del AUC de los algoritmos implementados y de librería sin optimizar , según los datos obtenidos en las pruebas de clasificación de la Base de Datos “BDD4”, se muestra que las curvas de los algoritmos k-NN implementado y Clasificador Bayesiano sin optimizar tienen la mayor área, por lo que son los mejores clasificadores de la base de datos.

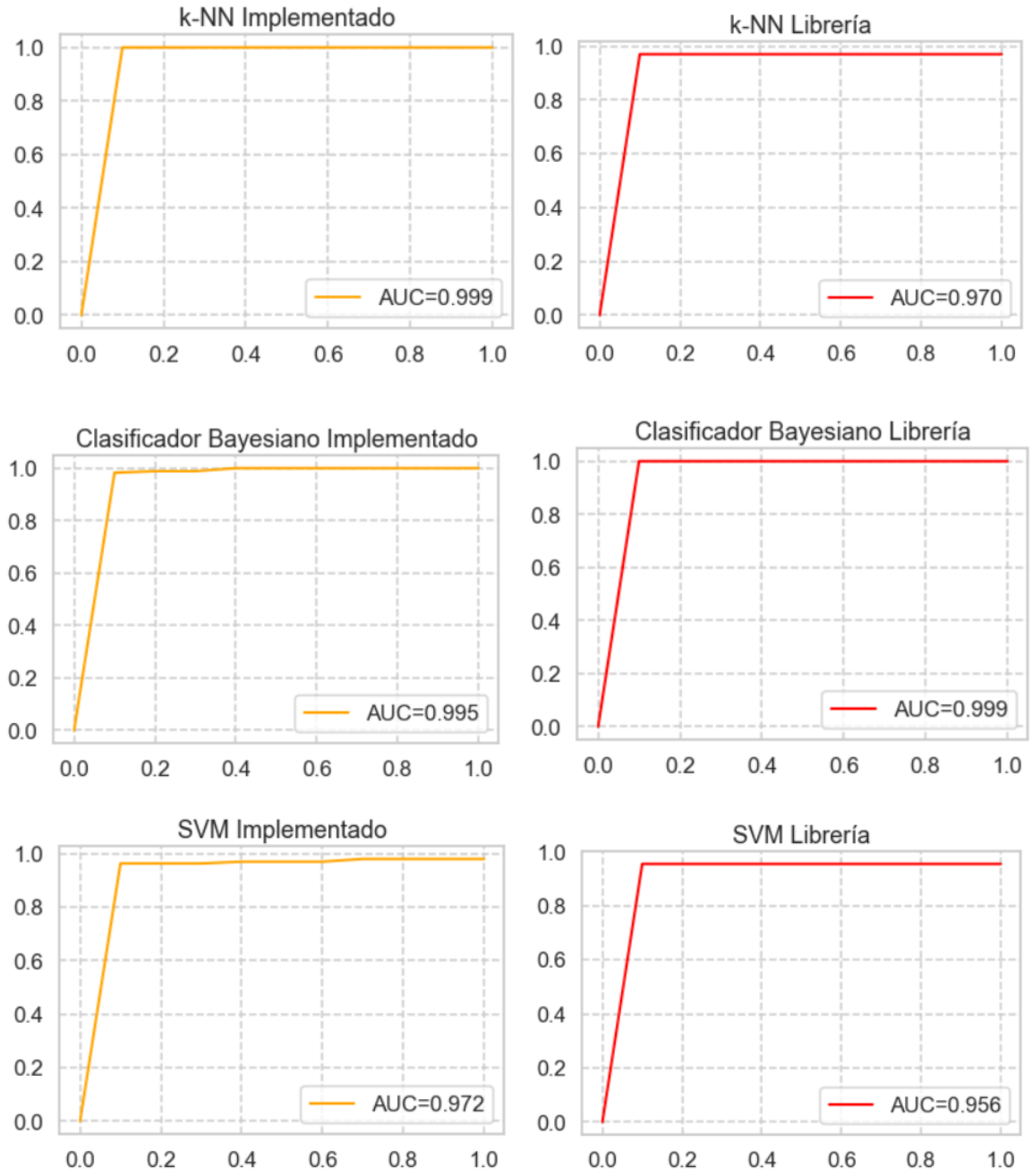


Figura 40. Curvas ROC - Análisis de algoritmos Implementados y de Librería con la Base de Datos "BDD4"

Fuente: Autoría

En la Figura 41, se muestra la implementación de las Curvas ROC y el cálculo del AUC de los algoritmos implementados y de librería sin optimizar , según los datos obtenidos en las pruebas de clasificación de la Base de Datos “BDD5”, se observa que las curvas de los algoritmos k-NN tienen la mayor área, por lo que son los mejores clasificadores de esta base de datos.

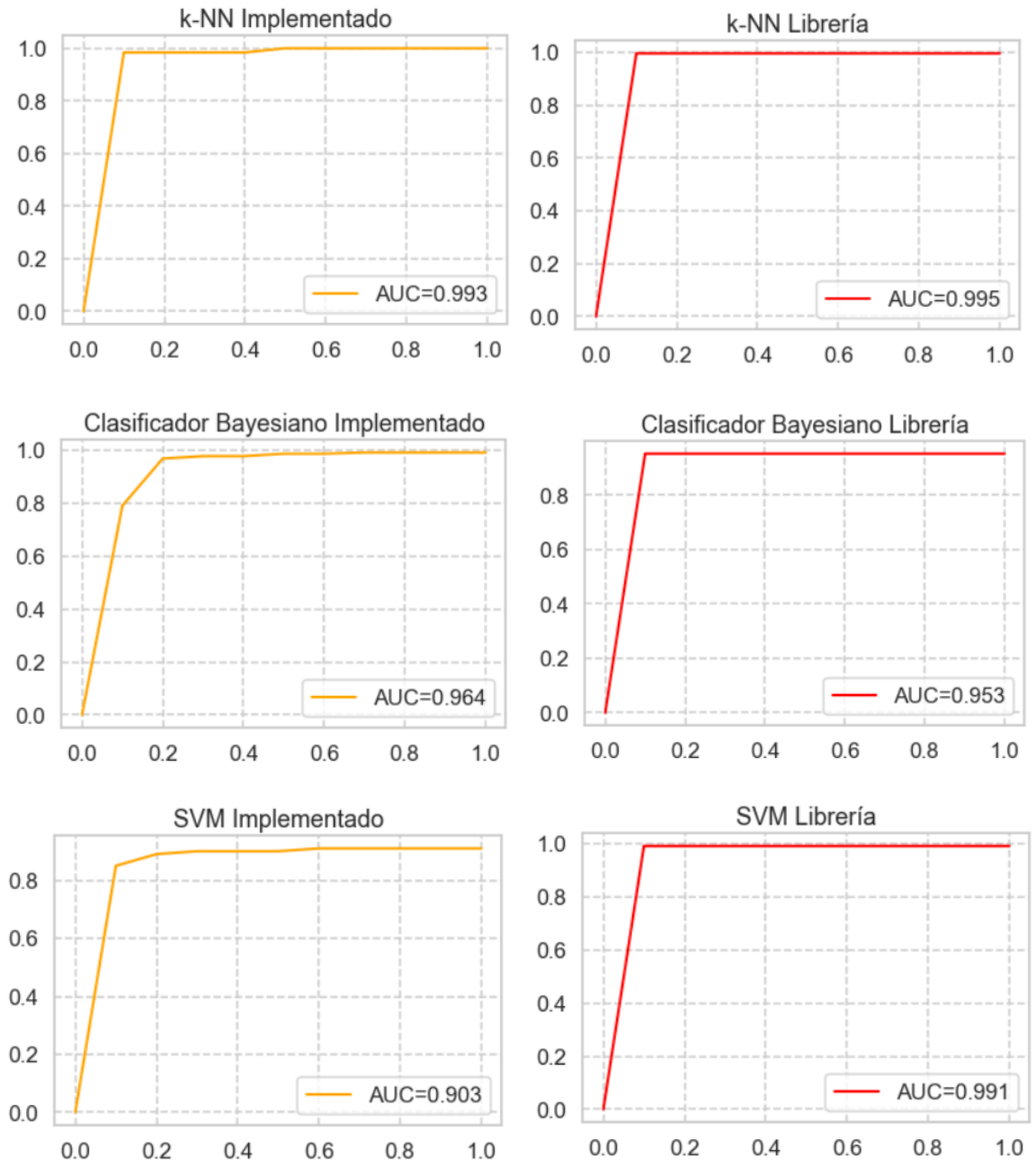


Figura 41. Curvas ROC - Análisis de algoritmos Implementados y de Librería con la Base de Datos "BDD5"

Fuente: Autoría

4.7. Resultados

En la Tabla 16-18, se evidencian los resultados obtenidos en las pruebas realizadas a los algoritmos implementados y de librería con la base de datos “BDD1”, la cual está conformada por 319 datos y 14 clases. Obteniendo como mejores clasificadores a los algoritmos **k-NN Implementado** (99.06 %) como el algoritmo de menor tamaño (5.6 KB) y a **Clasificador Bayesiano Librería** (99.6%) siendo el mejor en rapidez (10.12 ms).

Tabla 16. Resultados de clasificación algoritmos k-NN- Base de Datos "BDD1"

k-NN	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	5.6 KB	99.06%	0.94%	46.9 ms	0.990
Librería	6.99 KB	98.82%	1.18%	486.1 ms	0.988

Fuente: Autoría

Tabla 17. Resultados de clasificación algoritmos Clasificador Bayesiano- Base de Datos "BDD1"

Clasificador Bayesiano	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	6.09 KB	84.35%	15.65%	25.92 ms	0.755
Librería	7.36 KB	99.6%	0.4%	10.12 ms	0.995

Fuente: Autoría

Tabla 18. Resultados de clasificación algoritmos SVM- Base de Datos "BDD1"

SVM	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	10.04 KB	93.18%	6.82%	23.12 ms	0.934
Librería	11.6 KB	99.21%	0.79%	11.93 ms	0.991

Fuente: Autoría

En la Tabla 19-21, se evidencian los resultados obtenidos en las pruebas realizadas a los algoritmos implementados y algoritmos de librería con la base de datos “BDD2”, la cual está conformada por 530 datos y 4 clases. Obteniendo como mejores clasificadores a los algoritmos **Clasificador Bayesiano Implementado** (100 %) como el algoritmo de menor tamaño (6.09 KB) y a **Clasificador Bayesiano Librería** (100%) siendo el mejor en rapidez (8.97 ms).

Tabla 19. Resultados de clasificación algoritmos k-NN- Base de Datos "BDD2"

k-NN	**Tamaño**	**Porcentaje de Clasificación**	**Tasa de Error**	**Rapidez**	**AUC**
Implementado	5.6 KB	99.24%	0.76%	94.75	0.991
Librería	6.99 KB	94.81%	5.19%	1307.8	0.950

Fuente: Autoría

Tabla 20. Resultados de clasificación algoritmos Clasificador Bayesiano- Base de Datos "BDD2"

Clasificador Bayesiano	**Tamaño**	**Porcentaje de Clasificación**	**Tasa de Error**	**Rapidez**	**AUC**
Implementado	6.09 KB	100%	0%	27.92	0.999
Librería	7.36 KB	100%	0%	8.97	0.999

Fuente: Autoría

Tabla 21. Resultados de clasificación algoritmos SVM- Base de Datos "BDD2"

SVM	**Tamaño**	**Porcentaje de Clasificación**	**Tasa de Error**	**Rapidez**	**AUC**
Implementado	10.04 KB	97.96%	2.04%	24.58	0.979
Librería	11.6 KB	99.76%	0.24%	11.96	0.997

Fuente: Autoría

En la Tabla 22-24, se evidencian los resultados obtenidos en las pruebas realizadas a los algoritmos implementados y algoritmos de librería con la base de datos “BDD3”, la cual está conformada por 240 datos y 6 clases. Obteniendo como mejores clasificadores a los algoritmos **k-NN Implementado** (99.58 %) como el algoritmo de menor tamaño (5.6 KB) y a **Clasificador Bayesiano Librería** (100%) siendo el mejor en rapidez (19.97 ms).

Tabla 22. Resultados de clasificación algoritmos k-NN- Base de Datos "BDD3"

k-NN	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	5.6 KB	99.58%	0.42%	37.9	0.995
Librería	6.99 KB	96.87%	3.13%	383.95	0.969

Fuente: Autoría

Tabla 23. Resultados de clasificación algoritmos Clasificador Bayesiano- Base de Datos "BDD3"

Clasificador Bayesiano	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	6.09 KB	92.75%	7.25%	11.96	0.834
Librería	7.36 KB	100%	0%	13.97	0.999

Fuente: Autoría

Tabla 24. Resultados de clasificación algoritmos SVM- Base de Datos "BDD3"

SVM	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	10.04 KB	94.57%	5.43%	9.86	0.946
Librería	11.6 KB	90.10%	9.9%	10.97	0.905

Fuente: Autoría

En la Tabla 25-27, se evidencian los resultados obtenidos en las pruebas realizadas a los algoritmos implementados y algoritmos de librería con la base de datos “BDD4”, la cual está conformada por 246 datos y 4 clases. Obteniendo como mejores clasificadores a los algoritmos **k-NN Implementado** (100 %) como el algoritmo de menor tamaño (5.6 KB) y a **Clasificador Bayesiano Librería** (100%) siendo el mejor en rapidez (10.97 ms).

Tabla 25. Resultados de clasificación algoritmos k-NN- Base de Datos "BDD4"

k-NN	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	5.6 KB	100%	0%	25.93	0.999
Librería	6.99 KB	96.95%	3.05%	322.47	0.970

Fuente: Autoría

Tabla 26. Resultados de clasificación algoritmos Clasificador Bayesiano- Base de Datos "BDD4"

Clasificador Bayesiano	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	6.09 KB	99.6%	0.4%	16.955	0.995
Librería	7.36 KB	100%	0%	10.97	0.999

Fuente: Autoría

Tabla 27. Resultados de clasificación algoritmos SVM- Base de Datos "BDD4"

SVM	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	10.04 KB	97.24%	2.76%	12.08	0.972
Librería	11.6 KB	95.43%	4.57%	10	0.956

Fuente: Autoría

En la Tabla 28-30, se evidencian los resultados obtenidos en las pruebas realizadas a los algoritmos implementados y algoritmos de librería con la base de datos “BDD5”, la cual está conformada por 312 datos y 5 clases. Obteniendo como mejores clasificadores a los algoritmos **k-NN Implementado** (99.36 %) como el algoritmo de menor tamaño (5.6 KB) y a **Clasificador Bayesiano Implementado** (99.48%) siendo el mejor en rapidez (12.96 ms).

Tabla 28. Resultados de clasificación algoritmos k-NN- Base de Datos "BDD5"

k-NN	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	5.6 KB	99.36%	0.64%	46.87	0.993
Librería	6.99 KB	99.06%	0.4%	428.79	0.995

Fuente: Autoría

Tabla 29. Resultados de clasificación algoritmos Clasificador Bayesiano- Base de Datos "BDD5"

Clasificador Bayesiano	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	6.09 KB	99.48%	0.52%	12.96	0.964
Librería	7.36 KB	95.19%	4.81%	9.98	0.953

Fuente: Autoría

Tabla 30. Resultados de clasificación algoritmos SVM- Base de Datos "BDD5"

SVM	Tamaño	Porcentaje de Clasificación	Tasa de Error	Rapidez	AUC
Implementado	10.04 KB	89.92%	10.08%	18.06	0.903
Librería	11.6 KB	99.2%	0.08%	7.97	0.991

Fuente: Autoría

Finalmente, después de realizar las pruebas necesarias y analizar cada una de las tablas de resultado, se observa que los algoritmos de clasificación implementados superan el 84% de porcentaje de aciertos al momento de clasificar, además cabe destacar que, si se analiza el tamaño del algoritmo, los algoritmos implementados son los que tienen menor tamaño en comparación con los algoritmos de librería.

El algoritmo k-NN implementado tiene un porcentaje mínimo de aciertos del 99% y un tiempo máximo de clasificación de 47 milisegundos, su implementación destaca en escenarios donde se requiera el procesamiento de grupos pequeños de datos, a diferencia del clasificador Bayesiano implementado el cual destaca en escenarios donde se requiera el procesamiento de un gran número de datos ya que su porcentaje de clasificación en estas condiciones es del 100% y su tiempo máximo de procesamiento es de 28 milisegundos.

El algoritmo SVM implementado tiene un tiempo máximo de procesamiento de 25 milisegundos superando a los dos algoritmos antes mencionados, pero su deficiencia se encuentra en su tamaño siendo este mayor a los demás y en su porcentaje de aciertos ya que este en escenarios de bases de datos pequeñas es de 89% y en escenarios donde se requiera el procesamiento de un gran número de datos es 97%. Comprobando así los resultados expuestos en las tablas de resultado.

CAPÍTULO V. CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

- Mediante la revisión bibliográfica realizada para este proyecto, se logró determinar la importancia que tienen los algoritmos de clasificación implementados en SE, ya que estos en la actualidad se los utilizan en múltiples aplicaciones del sector médico y de las Telecomunicaciones, por lo que el uso del correcto algoritmo de clasificación es indispensable en estos dispositivos.
- En la etapa de selección de los algoritmos de clasificación supervisada se escogió a los algoritmos a base de distancias (k-NN), a base de probabilidad (Clasificador Bayesiano) y a base de Superplanos (SVM) ya que estos cumplían con una característica importante, que su implementación se la podía hacer en SE de bajos recursos.
- La utilización de un lenguaje de programación de uso libre como lo es Python facilita el desarrollo e implementación de software, y además al ser de fácil manejo, se convierte en una buena alternativa, para la creación de cualquier programa.
- En el proceso de implementación de los algoritmos seleccionados, la principal técnica de optimización de código que se utilizó fue la de “Detección de código muerto”, en donde se eliminó código que nunca que utilizaría ya que este podría ralentizar la memoria del sistema, donde el programa se estuviese ejecutando.
- El tamaño de los algoritmos implementados k-NN, Clasificador Bayesiano y SVM fue menor en comparación con el tamaño de los algoritmos de librería, esto fue un criterio fuerte a evaluar, ya que estos deben poder implementarse en SE de bajos recursos y un algoritmo de gran tamaño aumentaría el tiempo de procesamiento y uso de memoria de estos dispositivos, limitando su funcionamiento.
- Para que el proceso de validación de los algoritmos de clasificación tenga éxito, antes de cargar cada base datos se procedió a dividirlas en datos de entrenamiento y datos de

testeo. Con esto se consiguió entrenar a los algoritmos de una mejor manera obteniendo excelentes resultados al momento de clasificar.

- Las gráficas de la Curva ROC ayudaron a verificar mediante el cálculo de AUC la exactitud con la que los algoritmos estaban clasificando, en donde se comprobó que los algoritmos implementados estaban clasificando al mismo nivel que los algoritmos de librería, ya que los datos de AUC obtenidos no tenían mucha variación.
- La eficiencia del algoritmo de clasificación k-NN al trabajar con bases de datos pequeñas como es el caso de la base de datos "BDD3" y "BDD4" es de 99.58% y 100% respectivamente, concluyendo que este algoritmo tiene un óptimo funcionamiento en aplicaciones donde se requiera el procesamiento de grupos pequeños de datos.
- La eficiencia del algoritmo de clasificación clasificador Bayesiano al trabajar con bases de datos grandes como la base de datos "BDD2" es de 100%, concluyendo que este algoritmo tiene un óptimo funcionamiento en aplicaciones donde se requiera el procesamiento de grandes bases de datos.
- Los algoritmos k-NN y clasificador Bayesiano implementados debido a su tamaño, estos pueden ser implementados sin problemas en un SE Arduino y *Raspberry*. La única limitación existente se presenta en el SE arduino, ya que este puede solo puede compilar un limitado conjunto de datos, a diferencia del SE *Raspberry*, el cual tiene una mayor capacidad.
- El algoritmo de clasificación SVM a pesar de superar a los demás algoritmos en rapidez, este es ineficiente en tamaño y en porcentaje de aciertos. Ya que, al trabajar con bases de datos, este tiene como máximo un porcentaje de aciertos del 97%.
- Con el desarrollo de este proyecto de investigación se logró obtener un amplio conocimiento sobre la correcta implementación de software en SE, ya que de esto

depende el óptimo procesamiento de datos de las tecnologías enfocadas a aprendizaje automático actuales, las cuales buscan perfeccionarse con el tiempo.

5.2. Recomendaciones

- Es importante antes de la instalación, investigar y estudiar las herramientas del software a utilizar en la programación, para que de esta manera se pueda conseguir un correcto funcionamiento y así evitar posibles errores durante la ejecución del programa.
- Antes de evaluar las bases de datos se debe verificar que todas se encuentren en el mismo formato de archivos, ya que, si no es así, podría haber problemas al momento de compilar cada código.
- Es necesario que, al momento de dividir las bases de datos en datos de entrenamiento y datos de testeo, la mayor parte de los datos sean para el conjunto de entrenamiento, de esta forma podremos entrenar al sistema de una manera más exacta, mejorando así la precisión al momento de clasificar.
- Es recomendable el uso de un equipo con al menos 4GB de memoria RAM, para el correcto desempeño del Software (Anaconda), usado en la implementación de los algoritmos de clasificación, ya que se realiza varios procesos de cálculo en el procesamiento de las bases de datos y en la graficación de las curvas ROC.

BIBLIOGRAFÍA

Agarwal, T. (2015). *A Brief About Embedded System their Classifications and Applications*. https://www.efxkits.us/classification-of-embedded-systems/

AlAbwaini, N., Aldaaje, A., Jaber, T., Abdallah, M., & Tamimi, A. (2018). *Using Program Slicing to Detect the Dead Code*. 230–233. https://doi.org/10.1109/CSIT.2018.8486334

Alberri, M., Hegazy, S., Badra, M., Nasr, M., Shehata, O. M., & Morgan, E. I. (2018). Generic ROS-based Architecture for Heterogeneous Multi-Autonomous Systems Development. *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, 1–6. https://doi.org/10.1109/ICVES.2018.8519589

Alsibai, M. H. (2015). *A Smart Driver Monitoring System Using Android Application and Embedded System*. *November*, 27–29.

Araujo, B. S. (2006). *Aprendizaje Automático: conceptos básicos y avanzados*.

Arduino. (2020). *Ardunio*. https://www.arduino.cc

Askaruly, S. (2018). *Naive Bayes*. https://gist.github.com/tuttelikz/94f750ef3bf14f8a126a

Baek, N., & Lee, H. (2007). *A performance analysis for microprocessor architectures*.

Bahit, E. (2018). *Introduccion al Lenguaje Python*.

Benavides, A. R. del V. (2017). *Curvas ROC (Receiver-Operating-Characteristic) y sus aplicaciones*. 12–16. https://idus.us.es/xmlui/bitstream/handle/11441/63201/Valle Benavides Ana Rocío del TFG.pdf?sequence=1&isAllowed=y

Benítez, R. (2013). *Inteligencia artificial avanzada*.

Berrar, D. (2018). *Cross-Validation*. https://doi.org/10.1016/B978-0-12-809633-8.20349-X

Bohra, P., & Palivela, H. (2016). Understanding and formulation of various kernel techniques

for suport vector machines. *2015 IEEE International Conference on Computational Intelligence and Computing Research, ICCIC 2015*. https://doi.org/10.1109/ICCIC.2015.7435804

Carrasco, J. M. G. (n.d.). *LA OPTIMIZACION: UNA MEJORA EN LA EJECUCION De Programas*. 1–20.

Cerda Lorca, J., & Cifuentes, L. (2012). Uso de curvas ROC en investigación clínica: Aspectos teórico-prácticos. *Revista Chilena de Infectología*, *29*, 138–141. https://doi.org/10.4067/S0716-10182012000200003

Challenger-Pérez, Ivet. Díaz-Ricardo, Yanet. Becerra-García, R. A. (2014). El lenguaje de programación Python/The programming language Python. *Ciencias Holguín*, *XX*, 1–13. https://www.redalyc.org/pdf/1815/181531232001.pdf

Champutiz, W. (2019). *ROSES GREENHOUSE CULTIVATION DATABASE REPOSITORY (ROSESGREENHDB)*. https://ieee-dataport.org/open-access/roses-greenhouse-cultivation-database-repository-rosesgreenhdb

Dai, H. (2018). *Research on SVM Improved Algorithm for Large Data Classification*. *1*, 181–185.

Fiore, J. M. (2020). *Embedded Controllers*.

Fuentes, E. (2019). *FOOTSTEP ANALYSIS USING PRESSURE SENSORS*. https://ieee-dataport.org/open-access/footstep-analysis-using-pressure-sensors

Fuentes Gómez, M. D. C. (2013). Bases de Datos. In *Universidad Autónoma Metropolitana Unidad Cuajimalpa*.

García, V. S. (2014). *Optimización de Memoria Arduino*. https://booleanbite.com/web/optimizacion-de-memoria-de-nuestro-codigo-de-arduino/

Gupta, K., & Rakesh, N. (2018). IoT Based Automobile Air Pollution Monitoring System. *2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 14–15.

Halfacree, G. (2018). *Raspberry Pi Beginner's Guide*.

Herrero, J. G. (2015). *Clasificadores Bayesianos Métodos probabilísticos y numéricos de clasificación*.

Jahromi, A. H., & Taheri, M. (2018). A non-parametric mixture of Gaussian naive Bayes classifiers based on local independent features. *19th CSI International Symposium on Artificial Intelligence and Signal Processing, AISP 2017*, *2018-Janua*(1), 209–212. https://doi.org/10.1109/AISP.2017.8324083

Jayanth, Shourya, T. R., Anantharaman, A., Chinmay, C., Rao, K. U., & Salanke, G. R. (2018). and Control Algorithms for Application Specific Robot Designs. *2018 2nd International Conference on Inventive Systems and Control (ICISC)*, *Icisc*, 6–10.

Joyanes, L. (2013). Fundamentos Generales de Programación. *Вестник Казнму*, *№3*, с.30.

Khalid, S. S., & Abrar, S. (2013). Area under the ROC Curve of Enhanced Energy Detector. *2013 11th International Conference on Frontiers of Information Technology*, 131–135. https://doi.org/10.1109/FIT.2013.31

Khan, S. A., & Ali Rana, Z. (2019). Evaluating Performance of Software Defect Prediction Models Using Area Under Precision-Recall Curve (AUC-PR). *2019 2nd International Conference on Advancements in Computational Sciences (ICACS)*, 1–6. https://doi.org/10.23919/ICACS.2019.8689135

Kumar, R., & Indrayan, A. (2011). Receiver Operating Characteristic (ROC) Curve for Medical Researchers. *Indian Pediatrics*, *48*, 277–287. https://doi.org/10.1007/s13312-

011-0055-4

Liang, S., Liu, Y., Wang, C., & Jian, L. (2015). Neighbor Algorithm. *Architecture*, 291–296.

Lifelong Learning. (2011). *Ingenieria de los sistemas embebidos.* 1–19. http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE5_3_1.pdf

Lin, J. D., Cheng, A. M. K., & Gercek, G. (2016). Partitioning Real-Time Tasks with Replications on Multiprocessor Embedded Systems. *IEEE Embedded Systems Letters*, *8*(4), 89–92. https://doi.org/10.1109/LES.2016.2620473

Lukač, N., & Žalik, B. (2015). *Fast Approximate k-Nearest Neighbours Search Using GPGPU BT - GPU Computing and Applications* (Y. Cai & S. See (eds.); pp. 221–234). Springer Singapore. https://doi.org/10.1007/978-981-287-134-3_14

Lutful, Faizan, C., & Quavi, S. M. A. (2019). IOT Based Smart Garbage Monitoring System. *International Journal of Computer Sciences and Engineering*, *7*(2), 649–651. https://doi.org/10.26438/ijcse/v7i2.649651

Luyan, W., Zhangguo, S., & Long, C. (2013). The Performance Analysis for Embedded Systems using Statistics Methods. *TELKOMNIKA Indonesian Journal of Electrical Engineering*, *11*. https://doi.org/10.11591/telkomnika.v11i7.2864

Mapayi, T., & Tapamo, J. (2018). *Performance Comparison of Supervised Learning Methods for Retinal Vessel Tortuosity Characterisation*.

Mathai, N., Chen, Y., & Kirchmair, J. (2019). Validation strategies for target prediction methods. *Briefings in Bioinformatics*, *21*(3), 791–802. https://doi.org/10.1093/bib/bbz026

Mcguire, M. (2009). *Programming Language Notes*.

Nakov, S. (2013). *Fundamentals of Computer Programming with C# (The Bulgarian C# Programming Book) by Svetlin Nakov & Co. http://www.introprogramming.info*.

Nayyar, A., & Puri, V. (2015). Raspberry Pi-A Small, Powerful, Cost Effective and Efficient Form Factor Computer: A Review. *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)*, *5*, 720–737.

Nuratch, S. (2018). Design and Implementation of Real-time Embedded Data Acquisition and Classification with Web-based Configuration and Visualization. *2018 International Conference on Embedded Systems and Intelligent Technology & International Conference on Information and Communication Technology for Embedded Systems (ICESIT-ICICTES)*, 1–4.

Okoi, M. D. (2019). *Programming Languages for Embedded Systems*. https://www.fossmint.com/programming-languages-for-embedded-systems/

Pašalić, D., Cvijić, B., Bundalo, D., Bundalo, Z., & Kuzmić, G. (2017). Embedded systems for user identification in access to objects and services using mobile phone. *2017 6th Mediterranean Conference on Embedded Computing, MECO 2017 - Including ECYPS 2017, Proceedings, June*. https://doi.org/10.1109/MECO.2017.7977172

Patel, R., & Rajawat, A. (2014). Recent trends in embedded system software performance estimation. *Design Automation for Embedded Systems*, *17*. https://doi.org/10.1007/s10617-013-9125-2

Pawlovsky, A. P., & Matsuhashi, H. (2017). The use of a novel genetic algorithm in component selection for a kNN method for breast cancer prognosis. *Pan American Health Care Exchanges, PAHCE, 2017-March*. https://doi.org/10.1109/GMEPE-PAHCE.2017.7972084

Peter Kinz. (2017). *A Complete Guide to C Programmig*. http://www.lmpt.univ-tours.fr/~volkov/C++.pdf

Portilla, L. (2018). *ALCOHOL DETECTION IN DRIVERS BY SENSORS AND COMPUTER VISION*. https://ieee-dataport.org/open-access/alcohol-detection-drivers-sensors-and-computer-vision

Profentzas, C., Gunes, M., Nikolakopoulos, Y., Landsiedel, O., & Almgren, M. (2019). Performance of secure boot in embedded systems. *Proceedings - 15th Annual International Conference on Distributed Computing in Sensor Systems, DCOSS 2019*, 198–204. https://doi.org/10.1109/DCOSS.2019.00054

Rahman, A., & Qamar, U. (2016). *A Bayesian Classifiers based Combination Model for Automatic Text Classification*. 63–67.

Ray, S. (2019). A Quick Review of Machine Learning Algorithms. *Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Prespectives and Prospects, COMITCon 2019*, 35–39. https://doi.org/10.1109/COMITCon.2019.8862451

Rosero, P. (2018a). *BODY POSITION DATA WITH ACCELEROMETER SENSOR*. https://ieee-dataport.org/open-access/body-position-data-accelerometer-sensor

Rosero, P. (2018b). *DATA SET WHEELCHAIR SENSORS*. https://ieee-dataport.org/open-access/data-set-wheelchair-sensors

Ruuska, S., Hämäläinen, W., Kajava, S., Mughal, M., Matilainen, P., & Mononen, J. (2018). Evaluation of the confusion matrix method in the validation of an automated system for measuring feeding behaviour of cattle. *Behavioural Processes*, *148*, 56–62. https://doi.org/https://doi.org/10.1016/j.beproc.2018.01.004

Saguil, D., & Azim, A. (2020). A Layer-Partitioning Approach for Faster Execution of Neural Network-Based Embedded Applications in Edge Networks. *IEEE Access*, *8*, 59456–59469. https://doi.org/10.1109/ACCESS.2020.2981411

Sakurai, Y., Shimbo, K., Toba, T., & Osaka, H. (2015). The Network Performance Analysis Platform and Its Application to Network Buffer Evaluation of the Embedded System. *Proceedings - IEEE 9th International Symposium on Embedded Multicore/Manycore SoCs, MCSoC 2015*, 305–312. https://doi.org/10.1109/MCSoC.2015.32

Salih, D. M. (2019). Bayesian And Naive Bayesian Decision Boundaries For Multidimensional Cases. *2019 International Conference on Computing and Information Science and Technology and Their Applications (ICCISTA)*, 1–5.

Sandoval, A. G., Tijaro, O. J., & Moreno, Y. T. (2019). Acquisition and storage of optical interference fringes by means of an embedded system. *2019 22nd Symposium on Image, Signal Processing and Artificial Vision, STSIVA 2019 - Conference Proceedings*, 1–5. https://doi.org/10.1109/STSIVA.2019.8730246

Santana, J. S., & Farfán, E. M. (2014). El arte de programar en R. *Instituto Mexicano de Tecnología Del Agua*, 182. https://cran.r-project.org/doc/contrib/Santana_El_arte_de_programar_en_R.pdf

Seif, G. (2017). *Machine Learning Naive Bayes*. https://github.com/GeorgeSeif/Python-Machine-Learning/blob/master/Classification/naive_bayes.py

Selmani, A., Outanoute, M., Alaoui, M. A., El Khayat, M., Guerbaoui, M., Ed-Dahhak, A., Lachhab, A., & Bouchikhi, B. (2018). Multithreading design for an embedded irrigation system running on solar power. *Proceedings of the 2018 International Conference on Optimization and Applications, ICOA 2018*, 1–5. https://doi.org/10.1109/ICOA.2018.8370519

Sharma, A. (2013). *Applications Of Embedded System :* 1–3.

Sharma, K., & Nandal, R. (2019). A literature study on machine learning fusion with IoT. *Proceedings of the International Conference on Trends in Electronics and Informatics, ICOEI 2019, 2019-April*(Icoei), 1440–1445.

Sharma, V. (2019). *Importing and Splitting Data into Dependent and Independent Features for Machine Learning*.

Somvanshi, M., Chavan, P., Tambade, S., & Shinde, S. V. (2017). A review of machine learning techniques using decision tree and support vector machine. *Proceedings - 2nd International Conference on Computing, Communication, Control and Automation, ICCUBEA 2016*. https://doi.org/10.1109/ICCUBEA.2016.7860040

Srinivasan, A. (2018). IoT Cloud Based Real Time Automobile Monitoring System. *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, 231–235. https://doi.org/10.1109/ICITE.2018.8492706

Streitel, F., Steidl, D., & Jürgens, E. (2014). *Dead Code Detection On Class Level*.

Sucar, L. E. (2008). Clasificadores Bayesianos: de Datos a Conceptos. *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 1–2.

Surrel, G., Rincón, F., Murali, S., & Atienza, D. (2015). *Real-Time Probabilistic Heart Beat Classification and Correction for Embedded Systems*. 161–164.

Syuriadi, I. K. (2018). *Support Vector Machine*. https://github.com/ilhamksyuriadi/Support-Vector-Machine-using-scikit-learn/blob/master/svm.py

Tabari, A., Higgins, C., & Bale, P. (2015). Improved integration of medium scale PV plants on the distribution network. *2015 International Conference on Renewable Energy*

Research and Applications, ICRERA 2015, *5*, 461–466. https://doi.org/10.1109/ICRERA.2015.7418455

Thakkar, H. (2018). MED-IoT : A Medicine Confirmation System. *2018 International Conference on Smart City and Emerging Technology (ICSCET)*, 1–5.

Thukral, A. (2017). *SVM Classification*. https://github.com/AbhinavThukral97/LinearSVMClassification/blob/master/svm.py

Tian, Y., & Wang, X. (2017). *SVM ensemble method based on improved iteration process of Adaboost algorithm*. *2015*, 4026–4032.

Upadhyay, A., & Dhapola, A. (2015). *Embedded Systems And its Application in Medical Field*. https://doi.org/10.13140/2.1.1004.2406

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*(5), 1413–1432. https://doi.org/10.1007/s11222-016-9696-4

Velásquez, J. L. (2010). *UG Lenguaje de Programación : Introducción a C / C ++ (IDE)*. www.cimat.mx/~pepe/cursos/lenguaje_2010/slides/slide_17.pdf

Voudoukis, N. (2019). Arduino Based Embedded System and Remote Access Technologies of Environmental Variables Monitoring. *European Journal of Electrical Engineering and Computer Science*, *3*. https://doi.org/10.24018/ejece.2019.3.4.101

Wang, X., Seyfi, M., Chen, M., Ng, H. W., & Liang, J. (2020). *Age and Gender Estimation Using Small-Scale Convolutional Neural Network (CNN) Modules for Embedded Systems*.

Wilfrido, C. (2017). *PROGRAMACION R*.

Ya-lin Miao, Xiang-lin Miao, Zheng-Zhong Bian, & Yong-jie Zhang. (2006). *Design and Application of Embedded System Based on ARM7 LPC2104 Processor in Telemedicine*. *60271022*, 2187–2190. https://doi.org/10.1109/iembs.2005.1616896

Yamane, S. (2017). Deductively verifying embedded software in the era of artificial intelligence = machine learning + software science. *2017 IEEE 6th Global Conference on Consumer Electronics, GCCE 2017*, *2017-Janua*(Gcce), 1–4. https://doi.org/10.1109/GCCE.2017.8229475

Yan, B., & Nirenberg, S. (2017). *An Embedded Real-time Processing Platform for Optogenetic Neuroprosthetic Applications*. *XX*(XX), 1–10. https://doi.org/10.1109/TNSRE.2017.2763130

Yigit, H. (2013). A weighting approach for KNN classifier. *2013 International Conference on Electronics, Computer and Computation, ICECCO 2013*, *1*(2), 228–231. https://doi.org/10.1109/ICECCO.2013.6718270

Zai-ying, W., & Liu, C. (2015). *3.1 Design Of Video Acquisition Part*. *2440*, 5856–5859.

Zhou, Z.-H. (2017). A brief introduction to weakly supervised learning. *National Science Review*, *5*(1), 44–53. https://doi.org/10.1093/nsr/nwx106

Becker, D., Mcmullen, B., & King, T. D. (2015). BIG DATA , BIG DATA QUALITY PROBLEM, 2644–2653.

Hongxing, W., & Tianmiao, W. (2006). Curriculum of Embedded System for Software Colleges.

Mondkjar, J. A. R. (2002). A Tool for the Implementation of Heavy-Computational- Load Control Functions in Embedded Systems, 1057–1061.

Park, I., Lee, H., & Lee, H. (2013). Software Optimization for Embedded Communication,

676–679.

Feng, S., (2012). Supervised Classification Algorithms Based on Artificial Immune.

Mingyao, X., & Xiongfei, L.(2015).Embedded database query optimization algorithm based on particle swarm optimization.

Villalon, E. (2013).Classification Algorithm for Embedded Systems Using High-Resolution Multispectral Data.

Fuangkhon, P., & Tanprasert, T.(2009).An incremental learning algorithm for supervised neural network with contour preserving classification

Fouad, Hafez & Farouk, Hesham. (2016). Heart Rate Sensor Node Analysis for Designing Internet of Things Telemedicine Embedded System.

Plan Nacional de Desarrollo, (2017). Toda una Vida 1, 1–148.

ANEXOS

Anexo 1. Programación k-NN.

```
#Librerias

import pandas as pd
import math
import random
import time

#Algoritmo KNN

#Inicio tiempo de procesamiento
tiempo_inicial = time()

#Importar Dataset
df = pd.read_csv("DATASET")

#Convertimos todos los datos a float
full_data = df.astype(float).values.tolist()
splitRatio = 0.8

#Se llama a la funcion para dividir en datos de test y entrenamiento
train_set, test_set = dividir(full_data,splitRatio)

#Imprimir el total de datos de entrenamiento y test
print('Datos=', len(full_data)
print('Datos de entrenamiento=',len(train_set))
print('Datos de test=',len(test_set))

#Se llama a la funcion para separar clases
train, test = separar_clases(train_set,test_set)

#Se llama a la funcion para evaluar resultados
correct, total=evaluar()

#Fin tiempo de procesamiento
tiempo_final = time()

#Calcular tiempo de procesamiento
tiempo_ejecucion = tiempo_final - tiempo_inicial

#Imprimir porcentaje de aciertos
print('Porcentaje de aciertos:',(correct/total)*100)

#Imprimir tiempo de procesamiento en milisegundos
print ('El tiempo de ejecucion fue:',(tiempo_ejecucion)*1000 ,'ms')

#Funcion "dividir"

def dividir(full_data, splitRatio):
```

```
    trainSize = int(len(full_data) * splitRatio)
    train_set = []
    test_set = list(full_data)
    while len(train_set) < trainSize:
        index = random.randrange(len(test_set))
        train_set.append(test_set.pop(index))
    return train_set, test_set

#Funcion "separar_clases"

def separar_clases(train_set, test_set):
    train = {}
    for i in range(len(train_set)):
        vector = train_set[i]
        if (vector[-1] not in train):
            train[vector[-1]] = []
        train[vector[-1]].append(vector)

    test = {}
    for i in range(len(test_set)):
        vector2 = test_set[i]
        if (vector2[-1] not in test):
            test[vector2[-1]] = []
        test[vector2[-1]].append(vector2)
    return train, test

#Funcion "KNN"

def knn(data, predict, k):

#Creamos otro vector con las distancias, del cual
#tomaremos las mas cercanas
 distances = []

 for group in data:
    for features in data[group]:

    #Formula para calcular la distancia mas cercana
    mdistancia=sqrt((features[0]-predict[0])**2+
    (features[1]-predict[1])**2)
        distances.append([mdistancia,group])

    #Ordenamos las distancias para tomar los primeros k elementos
    votes = [i[1] for i in sorted(distances)[:k]]

    #Tomamos el dato mayoritario o que se repite mas veces
    vote_result = Counter(votes).most_common(1)[0][0]
    confidence = Counter(votes).most_common(1)[0][1] / k
```

```
    #Devolvemos el resultado
    return vote_result, confidence

#Funcion "evaluar"

def evaluar():
    correct=0
    total=0
    for group in test:
        for data in test[group]:
            vote,confidence = knn(train, data, k=5)
            if group == vote:
                correct += 1
            total += 1
    #Devolvemos el total de aciertos y el total de datos evaluados
    return correct,total
```

Anexo 2. Programación Clasificador Bayesiano.

```
#Librerias

import pandas as pd
import math
import random
import time

#Algoritmo Clasificador Bayesiano

 #Inicio tiempo de procesamiento
 tiempo_inicial = time()

 #Importar Dataset
 df = pd.read_csv("DATASET")

 #Convertimos todos los datos a float
 dataset = df.astype(float).values.tolist()
 splitRatio = 0.8

 #Se llama a la funcion para dividir en datos de test y entrenamiento
 trainingSet, testSet = dividir_datos(dataset, splitRatio)

 #Imprimir el total de datos de entrenamiento y test
 print('Datos=', len(dataset)
 print('Datos de entrenamiento=',len(trainingSet))
 print('Datos de test=',len(testSet))

 #Se llama a la funcion para sacar la media aritmetica de cada clase
 summaries = summarizeByClass(trainingSet)

 #Se llama a la funcion para hacer la prediccion
 predictions = ob_predicciones(summaries, testSet)

 #Se llama a la funcion para calcular el porcentaje de aciertos
 accuracy = tasa_pred(testSet, predictions)

 #Fin tiempo de procesamiento
 tiempo_final = time()

 #Calcular tiempo de procesamiento
 tiempo_ejecucion = tiempo_final - tiempo_inicial

 #Imprimir porcentaje de aciertos
 print('Porcentaje de aciertos:',accuracy)

 #Imprimir tiempo de procesamiento en milisegundos
 print ('El tiempo de ejecucion fue:',(tiempo_ejecucion)*1000 ,'ms')


#Funcion "dividir_datos"

 def dividir_datos(dataset, splitRatio):
```

```
    #Tamano de datos de entrenamiento
    trainSize = int(len(dataset) * splitRatio)
    trainSet = []
    test = list(dataset)
    while len(trainSet) < trainSize:

        #Se toma datos al azar
        index = random.randrange(len(test))

        #Se agrega datos al vector
        trainSet.append(test.pop(index))

    #Se devuelven los datos de entrenamiento y test
    return [trainSet, test]

#Funcion "separar_clases"

 def separar_clases(dataset):
    separar = {}

    #Separar los datos por clase
    for i in range(len(dataset)):
        vector = dataset[i]
        if (vector[-1] not in separar):
            separar[vector[-1]] = []
        separar[vector[-1]].append(vector)
    return separar

#Funcion "media"

 #Funcion para sacar la media
 def media(numbers):
    return sum(numbers)/float(len(numbers))

#Funcion "stdev"

 #Funcion para sacar la variacion
 def stdev(numbers):
    avg = media(numbers)
    variance =sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1)
    return math.sqrt(variance)

#Funcion "sumarizar"

 #Funcion para sumarizar los datos por clase
 def sumarizar(dataset):
    summaries = [(media(attribute), stdev(attribute)) for attribute
    in zip(*dataset)]
    del summaries[-1]
    return summaries

#Funcion "summarizeByClass"
```

```
 #Sumarizar los datos por clase (Juntar los datos por clase)
 def summarizeByClass(dataset):
     separar = separar_clases(dataset)
     summaries = {}
     for classValue, instances in separar.items():
         summaries[classValue] = sumarizar(instances)
     return summaries

#Funcion "calculateProbability"

 def calculateProbability(x, media, stdev):
     if stdev== 0:
         return 0

     #Formula probabilidad Bayesiana
     exponent = math.exp(-(math.pow(x-media,2)/(2*math.pow(stdev,2))))
     return (1 / (math.sqrt(2*math.pi) * stdev)) * exponent

#Funcion "calculateClassProbabilities"

 #Calcular la probabilidad de cada clase
 def calculateClassProbabilities(summaries, inputVector):
     probabilities = {}
     for classValue, classSummaries in summaries.items():
         probabilities[classValue] = 1
         for i in range(len(classSummaries)):
             media, stdev = classSummaries[i]
             x = inputVector[i]

             #Se llama a la funcion del calculo de la probabilidad
             probabilities[classValue] *=
              calculateProbability(x, media, stdev)
     return probabilities

#Funcion "predict"

 #Funcion para calcular la prediccion
 def predict(summaries, inputVector):
    probabilities = calculateClassProbabilities(summaries, inputVector)


    bestLabel, bestProb = None, -1
    for classValue, probability in probabilities.items():
        if bestLabel is None or probability > bestProb:
            bestProb = probability
            bestLabel = classValue
    return bestLabel

#Funcion "ob_predicciones"

 #calculamos la prediccion
 def ob_predicciones(summaries, testSet):
     predictions = []
     for i in range(len(testSet)):
```

```
            #Llamamos a la funcion de prediccion
            result = predict(summaries, testSet[i])
            predictions.append(result)
        return predictions

#Funcion "tasa_pred"

 #Funcion para calcular el porcentaje de aciertos
 def tasa_pred(testSet, predictions):
        correct = 0
        for x in range(len(testSet)):
            if testSet[x][-1] == predictions[x]:
                correct += 1
        return (correct/float(len(testSet))) * 100
```

Anexo 3. Programación SVM.

```
#Librerias

import pandas as pd
from random import seed
from random import randrange
from math import floor
from math import exp
from math import log
from time import time

#Algoritmo SVM

#Inicio tiempo de procesamiento
tiempo_inicial = time()

#Importar Dataset
df = pd.read_csv("DATASET")

#Convertir todos los datos a float
full_data = df.astype(float).values.tolist()

#Dividir la base de datos por caracteristicas (x) y clases (y)
x = data[:,:(data.shape[1]-2)]
y = data[:,(data.shape[1]-1)]

#Escalado de caracteristicas mediante el uso de maximos y minimas
stats = statistics(x)
scale(x,stats)

#Dividimos el dataset en datos de entrenamiento y datos de test
test_data_size = 0.2 #Volumen de datos de test
learning_rate = 0.01
validations = 1 #Total de validaciones
final_score = cross_validation(x,y,test_data_size,
        validations,learning_rate,epoch)

#Imprimir Resultados

print("Datos:_", (floor(len(x)*(1 - test_data_size)))+(len(x)
- floor(len(x)*(1 - test_data_size))))
print("Porcentaje_de_aciertos:_",final_score*100,"%")

#Fin tiempo de procesamiento
tiempo_final = time()

#Calcular tiempo de procesamiento
tiempo_ejecucion = tiempo_final - tiempo_inicial
```

```
#Imprimir tiempo de procesamiento en milisegundos
print ('El_tiempo_de_ejecucion_fue:',(tiempo_ejecucion)*1000 ,'ms')

#Clase para dividir el dataset en Entrenamiento y Test

 def cross_val_split(data_X,data_Y,test_size,seed_val):
     data_x = data_X.tolist()
     data_y = data_Y.tolist()
     seed(seed_val)
     train_size = floor((1 - test_size)*len(data_x))
     train_x = []
     train_y = []

     while(len(train_x)<train_size):
         index = randrange(len(data_x))
         train_x.append(data_x.pop(index))
         train_y.append(data_y.pop(index))

     return train_x,train_y,data_x,data_y


#Funcion para devolver las estadisticas de minimos
       # de la columna para el escalado

 def statistics(x):
     cols = list(zip(*x))
     stats = []
     for e in cols:
         stats.append([min(e),max(e)])
     return stats

#Funcion para escalar las caracteristicas
 def scale(x, stat):
     for row in x:
         for i in range(len(row)):
             row[i] = (row[i] - stat[i][0])/(stat[i][1] - stat[i][0])

#Funcion para calcular Theta transpose x Feature Vector
 def ThetaTX(Q,X):
     det = 0.0
     for i in range(len(Q)):
         det += X[i]*Q[i]
     return det

#Funcion para calcular el costo por clase negativa (classs = 0)
 def LinearSVM_cost0(z):
     if(z < -1): #Ensuring margin
         return 0


     return z + 1

#Funcion para calcular el costo por clase positiva (classs = 1)
 def LinearSVM_cost1(z):
     if(z > 1): #Ensuring margin
         return 0
     return -z + 1
```

```
#Funcion pra calcular la sigmoid
 def sigmoid(z):
     return 1.0/(1.0 + exp(-z))

#Funcion para calcular el costo de SVM
 def cost(theta,c,x,y):
    cost = 0.0
    for i in range(len(x)):
       z = ThetaTX(theta[c], x[i])
       cost += y[i]*LinearSVM_cost1(z) + (1 - y[i])*LinearSVM_cost0(z)
       #cost += -1*(y[i]*log(sigmoid(z))+(1 - y[i])*log(1-sigmoid(z)))
    return cost

#Funcion para realizar el descenso del gradiente
        #en los pesos de cada clase

 def gradDescent(theta,c,x,y,learning_rate):
     oldTheta = theta[c]
     for Q in range(len(theta[c])):
       derivative_sum = 0
       for i in range(len(x)):
         derivative_sum +=(sigmoid(ThetaTX(oldTheta,x[i]))-y[i])*x[i][Q]
       theta[c][Q] -= learning_rate*derivative_sum


#Funcion para devolver predicciones utilizando pesos entrenados
 def predict(data,theta):
    predictions = []
    count = 1
    for row in data:
        hypothesis = []
        multiclass_ans = [0]*len(theta)
        for c in range(len(theta)):
            z = ThetaTX(row,theta[c])
            hypothesis.append(sigmoid(z))
        index = hypothesis.index(max(hypothesis))
        multiclass_ans[index] = 1
        predictions.append(multiclass_ans)
        count+=1
    return predictions


#Funcion para devolver la precision
 def accuracy(predicted, actual):
     n = len(predicted)
     correct = 0
     for i in range(n):
         if(predicted[i]==actual[i]):
             correct+=1
     return correct/n

#Funcion para realizar validacion cruzada
 def cross_validation(x,y,test_data_size,validations,
                learning_rate,epoch):

     accuracies = []
     for valid in range(validations):
             x_train, y_train, x_test, y_test =
               cross_val_split(x,y,test_data_size,valid+1)
```

```
        #Convirtiendo y_train a columnas de clase con valores 0/1
        classes = []
        for i in range(len(y)):
            classes.append([row[i] for row in y_train])

        #Iniciando Theta (Pesos)
        theta = [[0]*len(x_train[0]) for _ in range(len(classes))]

        #Entrenando el modelo
        for i in range(epoch):
            for class_type in range(len(classes)):
              gradDescent(theta,class_type,x_train,classes[class_type],
                      learning_rate)

            if(i%(epoch/10)==0):
              print("Procesando", i*100/epoch,"%")

        print("Completado")
        #Prediciendo usando el test de datos
        y_pred = predict(x_test,theta)
        #Calculando la exactitud
        accuracies.append(accuracy(y_pred,y_test))

    return sum(accuracies)/len(accuracies)
```