Universidad Autónoma de Baja California

Facultad de Ciencias Químicas e Ingeniería
Maestría y Doctorado en Ciencias e Ingeniería

Modelos Inteligentes Difusos para Big Data

Tesis para obtener el grado de Doctor en Ciencias

Presenta:

M.C. Sukey Sayonara Nakasima López

Bajo la dirección de:

Dr. Mauricio Alonso Sánchez Herrera

Co-dirigido por:

Dr. Juan Ramón Castro Rodríguez

Tijuana, Baja California septiembre 2022

Universidad Autónoma de Baja California
FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA

Folio No.330
Tijuana, B.C., a 25 de agosto del 2022

C. Sukey Sayonara Nakasima López
Pasante de: Doctorado en Ciencias
Presente

El tema de trabajo y/o tesis para su examen profesional, en la Opción TESIS.

Es propuesto, por las C. Dr. Mauricio Alonso Sánchez Herrera y Dr. Juan Ramón Castro Rodríguez.

Quienes serán las responsables de la calidad del trabajo que usted presente, referido al tema "Modelos Inteligentes Difusos Para Big Data".

El cual deberá usted desarrollar, de acuerdo con el siguiente orden:

I. INTRODUCCIÓN
II. MARCO TEÓRICOS
III. SISTEMA NEURO-DIFUSO BASADO EN MAMDANI CON DEFUSIFICACION EN CENTRO DE CONJUNTOS
IV. ESTUDIO EXPERIMENTAL

Dra. Ana Alejandra Ramírez Rodríguez
Sub-Directora

Dr. Juan Ramón Castro Rodríguez
Co - Director de Tesis

M.C. Roberto Alejandro Reyes Martínez
Director

Dr. Mauricio Alonso Sánchez Herrera
Director de Tesis

UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA
FACULTAD DE CIENCIAS QUÍMICAS E INGENIERÍA
COORDINACIÓN DE POSGRADO E INVESTIGACIÓN

Agradecimientos

Quiero agradecer a mis padres por su gran apoyo y acompañamiento en todo lo que me he propuesto a lo largo de mi vida, a mi hermana por siempre tener las palabras adecuadas cuando la frustración se hacía presente y su apoyo incondicional, a mis profesores Dra. Olivia Mendoza y Dr. Antonio Rodríguez por compartir de forma generosa su conocimiento y experiencias, a mis compañeros y amigos de laboratorio por su ayuda y por los momentos tan amenos que pasamos después de clase, a mis directores de tesis al Dr. Mauricio Sánchez y al Dr. Juan Ramón Castro por abrirme las puertas en esta área tan compleja pero tan bonita, por creer que podía y ayudarme a creer en mí, por guiarme y siempre acompañarme. Finalmente le doy gracias a Dios por poner en mi camino a personas con una gran calidez humana y un entorno que siempre propiciaba mi mejora y desarrollo continuo, a todos y cada uno de ustedes muchas gracias por ayudarme a lograr mi objetivo, concluir mi tesis doctoral.

Resumen

La rápida evolución y desarrollo de tecnología computacional, su asequibilidad, ubicuidad , hiperconectividad por parte de los usuarios y nuevos modelos de negocio contribuyen entre otras cosas a la generación de datos, la cual se estima que para el 2020 su velocidad de generación fue a una tasa de 1.7 Mbps cada día por usuario, considerando que los usuarios conectados al internet generando y consumiendo contenido representaron para ese año el 53% de la población global (más de 4 billones de personas).

La masificación de los datos trae consigo una serie de retos tanto en el proceso de generación (captación, recuperación, procesamiento y almacenamiento) así como en su análisis y visualización, por tal razón los datos ya no pueden ser tratados con los métodos, técnicas y tecnologías tradicionales.

Esta tesis está enfocada al desarrollo de algoritmos inteligentes que puedan procesar altos volúmenes de datos, y desempeñarse en un entorno tanto paralelizable como distribuido, buscando tomar provecho de las capacidades computacionales que brinda el ecosistema de Big Data, donde el computo intensivo en memoria es uno de sus potenciales más fuertes, tratando también hacer frente a los retos que su tratamiento conlleva, como son: volumen, velocidad, variedad, veracidad, valor, entre otras.

Tabla de contenidos

Índices de figuras

Índices de tablas

CAPÍTULO 1. Introducción

1.1 Problemática y justificación

Debido al creciente desarrollo tecnológico, facilidades de comunicación, acceso y ubicuidad, estamos generando cantidades inconmensurables de datos heterogéneos provenientes de fuentes diversas que ya no son posible de almacenar, procesar, analizar y visualizar a partir de herramientas tradicionales, a este fenómeno se le ha denominado Big Data. Akoaka et. al. [7] enfatiza que el 95% de los datos están constituidos por los no estructurados, agregando así mayor complejidad en su procesamiento y análisis.

Se ha reportado que cada día se generan 2.5 quintillones de bytes de datos, para dimensionar este volumen podemos decir que 1 exabyte es igual a 1 quintillón de bytes, lo que a su vez equivale a 1 billón de gigabytes [67], se estima que para el 2020 se habrán generado y consumido más de 50 zettabytes de datos, por lo que se requerirá de nuevas tecnologías, técnicas, metodologías computacionales y pensamiento estadístico para poder hacer frente a los desafíos que este fenómeno presenta [12].

Si bien se cuenta con la infraestructura tecnológica para el almacenamiento y procesamiento distribuido, paralelizable y escalable de estos grandes volúmenes de datos, existe un alto interés por desarrollar nuevos algoritmos que hagan frente a los desafíos de acumulación de ruido por alta dimensionalidad, datos incompletos e imprecisos, alta variabilidad en calidad y tasa de cambio por latencia en red.

Para ello, el Cómputo Inteligente ofrece un conjunto de métodos bioinspirados los cuales combinados entre sí pueden potenciar sus ventajas, ofreciendo capacidades de aprendizaje, adaptación de entornos cambiante y razonamiento aproximado, de forma similar a como el ser humano afronta la complejidad, permitiendo simplificar la representación de grandes volúmenes de datos, extracción de conocimiento de forma optimizada, que puedan garantizar tanto su valor como su utilidad, aprovechando a su vez las ventajas tecnológicas que las plataformas para el procesamiento de Big Data nos brindan.

1.2 Hipótesis

El modelado de algoritmos inteligentes basados en métodos de inferencia difusa y aprendizaje automático para un ambiente de Big Data, permiten el análisis de poblaciones heterogéneas de alta dimensionalidad y volumen, con la finalidad de descubrir patrones ocultos, correlaciones, tendencias y/o predicciones, proporcionando resultados altamente interpretables y de valor para la toma de decisiones informada y basada en conocimiento.

1.3 Objetivos

Diseñar e implementar un Sistema Neuro-Difuso basado en Mamdani con Defusificación Center-of-set, sobre una plataforma de Big Data, para el procesamiento de grandes volúmenes de datos.

1.3.1 Objetivos específicos

i. Diseño de la arquitectura para un Sistema Neuro-Difuso basado en Mamdani con Defusificación Center-of-set.

ii. Implementación del Sistema Neuro-Difuso basado en Mamdani con Defusificación Center-of-set en un entorno de desarrollo integrado que permita el procesamiento y análisis de un volumen de datos de medio a bajo.

iii. Implementación de una red neuronal artificial prealimentada (feedforward) y retro-propagación (backpropagation), utilizando el método de aprendizaje full batch y el método de optimización de gradiente descendente con momentum y tasa de aprendizaje adaptativa para acelerar la búsqueda de los valores iniciales óptimos de los hiperparámetros de entrenamiento y agilizar el ajuste de los parámetros de diseño sobre una plataforma en clúster de Big Data.

1.3.2 Metas

I. Realizar una revisión de literatura acerca de los fundamentos teóricos del fenómeno de Big Data y las metodologías de cómputo inteligente.
II. Estudio de los conceptos y elementos fundamentales de las arquitecturas de sistemas híbridos inteligentes, enfoque principalmente en Neuro-Difusos basado en Mamdani con Defusificación en Center-of-set.
III. Evaluación y análisis del desempeño de diferentes heurísticas para el óptimo ajuste de hiperparámetros de entrenamiento, enfoque principal en el ajuste de la tasa de aprendizaje.
IV. Evaluación y análisis de los métodos de aprendizaje full batch, online y mini-batch sobre el Sistema Neuro-Difuso basado en Mamdani con Defusificación en Center-of-set, afín de valorar las distintas formas en que se pueda llevar a cabo un aprendizaje con precisión y eficiencia.
V. Estudio del entorno de trabajo en clúster open-source Apache Spark para el procesamiento y analítica de Big Data, enfoque principal en su arquitectura, API pyspark y librerías MLlib.
VI. Evaluación y análisis experimental de la red neuronal artificial prealimentada (feedforward) y retro-propagación (backpropagation) utilizando el método de aprendizaje full batch y el método de optimización de gradiente descendente con momentum y tasa de aprendizaje adaptativa sobre la plataforma en clúster de Apache Spark.
VII. Validación de los resultados a partir de métricas de desempeño y evaluación de la bondad de ajuste, utilizando; suma de los errores al cuadrado, error cuadrático medio y raíz del error cuadrático medio.
VIII. Corrección y/o ajuste de la red neuronal artificial desarrollada para la plataforma en clúster para Big Data, utilizando como referente los datos obtenidos de la experimentación.

1.4 Organización del documento

Esta tesis está dividida en cuatro principales secciones. En la sección 2 contextualizaremos la situación actual de la analítica y procesamientos de datos masivos, los fundamentos teóricos requeridos para la comprensión de la investigación conducida en esta tesis. La sección 3 está dedicada a la descripción de la arquitectura del modelo neuro-difuso basado en Mamdani con Defusificación en Center-of-set y método de aprendizaje full batch que puede ser ejecutado sobre un clúster de Big Data. La sección 4 está dedicada a mostrar una serie de experimentos y resultados ejecutados en un ambiente Big Data, sobre el sistema neuro-difuso antes comentado. En la sección 5, finalizamos con conclusiones, hallazgo y trabajos futuros sobre la investigación realizada.

1.5 Contribuciones de investigación

1.5.1 Capítulos de libros

i. Nakasima-López, S., Sanchez, M. A., & Castro, J. R. (2018). Big Data and Computational Intelligence: Background, Trends, Challenges, and Opportunities. In Computer Science and Engineering—Theory and Applications (pp. 183-196). Springer, Cham.

ii. Nakasima-López, S., Sanchez, M. A., & Castro, J. R. (2020). Evaluation and Analysis of Performances of Different Heuristics for Optimal Tuning Learning on Mamdani Based Neuro-Fuzzy System, Studies in Computational Intelligence, vol 862., Springer, Cham.

1.5.2 Artículos

i. Nakasima-López, S., Castro, J. R., Sanchez, M. A., Mendoza, O., & Rodríguez-Díaz, A. (2019). An approach on the implementation of full batch, online and mini-batch learning on a Mamdani based neuro-fuzzy system with center-of-sets defuzzification: Analysis and evaluation about its functionality, performance, and behavior. PLOS ONE, 14(9), e0221369.

CAPÍTULO 2. Marco teórico

Con la llegada de la tercera revolución industrial a mediados de 1990, se inició la era de la automatización y digitalización de la información basados en la tecnología de cómputo e internet, consolidando poderosas estructuras que cambiarían el mundo de la comunicación y generación de conocimiento, teniendo un impacto significativo en la sociedad moderna y la transición hacia una nueva economía basada en datos y en la toma de decisiones informada [11] [62].

En la actualidad se puede observar un crecimiento considerable en la cantidad de datos que se pueden capturar y utilizar, así como la velocidad exponencial con la que se generan, al respecto Helbing, D. [26] hace una observación muy adecuada, pues comenta que hay más máquinas conectadas al internet que usuarios humanos. De acuerdo con Sivarajah, U. [67] se producen alrededor de 2.5 quintillones de bytes de datos en el mundo cada día, desde diversas tecnologías, tales como; internet de las cosas, redes sociales, dispositivos móviles, entre otras, por lo que según comenta [12] se estima que para el 2020 se habrán generado y consumido más de 50 zettabytes de datos, lo cual dificultará las actividades de organización, interpretación, almacenamiento y análisis de grandes volúmenes de datos gestionados y tratados con herramientas tradicionales, por lo que se requerirá de nuevas tecnologías, técnicas, metodologías computacionales y pensamiento estadístico para poder hacer frente a los desafíos que este fenómeno presenta.

2.1 Evolución del análisis de datos

Con la progresiva evolución de la informatización, hemos transitado de la necesidad de solo almacenar y gestionar nuestros datos a la posibilidad vincular distintas fuentes y extraer valor de ellos, para lo cual las prácticas de análisis y analítica se han tenido que ir transformando en el tiempo.

Se destaca como primera referencia de inteligencia de negocios en el campo del análisis de datos, la cual fue hecha por Hans Peter Luhn en 1958, sin embargo, fue en 1980 cuando Howard Dresner consolidó dicho término, hablando de la inteligencia de negocios como un conjunto de software que respalda la toma de decisiones empresariales en función de datos recopilados [50][70].

A finales de 1980 surgió la expresión de ***minería de datos y el proceso de descubrimiento de conocimiento en base de datos***, que tuvo origen en la inteligencia artificial, los cuales se define como el proceso de descubrimiento de patrones en los datos y se centra en el pronóstico y análisis predictivo. Pronto estas técnicas en conjunto con aprendizaje maquinan permitirían construir modelos para determinar los resultados de un evento que pudieran ocurrir en el futuro, lo cual puede conducir a la identificación tanto de oportunidades como de riesgos [50][70][73].

En la actualidad, la integración de la analítica de Big Data como soporte a la toma de decisiones por grandes corporaciones como por ejemplo Walmart ha sido de gran relevancia, pues ha permitido pronosticar con éxito el volumen de aprovisionamiento necesario cuando un desastre natural se acerca, como fue el caso del huracán Katrina en el 2005 [28]. Otro de los casos relevantes que mostraron el poder predictivo del Big Data fue para el proyecto Google Flu Trends, donde a través de las consultas que los usuarios realizaban en el buscador de Google con términos como “síntomas de gripe” y “tratamientos para gripe”, pudieron predecir con antelación el aumento de pacientes que llegarían con gripe a un hospital de alguna región, sin embargo, más adelante la revista de investigación nature evidenciaría algunas fallas en dichas predicciones [46].

La información que los buscadores pueden revelar acerca de sus usuarios es realmente valiosa, ya que a través de sus búsquedas o por las secuencias de clics en sus navegaciones, los usuarios pueden compartir sus necesidades, preocupaciones, intereses, patrones de comportamiento, entre otras, dicha información puede convertirse en ventaja competitiva, estrategia de mercado e incluso en la base de un nuevo modelo de negocios [64].

Existe un gran potencial y alto valor ocultos en grandes volúmenes de datos, los cuales exigen de nuevas tecnologías, técnicas y metodologías de innovación informática para modelar una gran variedad de fenómenos [45].

2.2 Big Data

Roger Magoulas acuñó el término de Big Data en el 2005, al observar una amplia variedad de grandes conjuntos de datos casi imposibles de administrar y procesar utilizando herramientas y técnicas tradicionales tanto para su almacenaje como para su procesamiento, no solo por su enorme volumen sino también por su alta complejidad [16][69]. Big Data es un término en evolución el cual empezó a ganar popularidad a principio del año 2000 con el auge del comercio electrónico, dicho fenómeno presenta exigencias y requerimientos de arquitecturas distribuidas y paralelizables, así como de poderosos y avanzados algoritmos para su cómputo, visualización y análisis [25][34][52].

En 2001, Doug Laney describió tres dimensiones características de Big Data también conocidas como 3V's [39][57][60], a continuación, se describen cada una de ellas [17][25][41][42][52]:

1. Volumen: atributo principal que caracteriza al fenómeno de Big Data, se refiere a la cantidad, tamaño y escala de los datos, lo cual agrega complejidad computacional. No hay una medida estándar que fije cual es el tamaño mínimo de los datos para ser considerado Big Data [12], sin embargo, algunos investigadores consideran que el tamaño mínimo es a partir de 1 terabyte (TB), se estima que Facebook ha almacenado 260 mil millones de fotografías utilizando más de 20 petabytes (PB) de almacenamiento, corporaciones transnacionales como Walmart y la tienda electrónica Alibaba generaron lotes de más de un millón de transacciones por hora y docenas de transacciones en TB por día, respectivamente.

2. Velocidad: se refiere al flujo de datos que es creado, capturado, agregado y a la tasa de análisis de datos, la cual debe realizarse de manera rápida y oportuna, afín de conservar su pertinencia y obtener valor de ellos. Un enlace de red simple puede generar más de un TB de datos durante el día y un PB de datos en un año a una velocidad de 1 Gbps.

3. Variedad: alude a los diferentes tipos de datos, a su heterogeneidad, siendo estos del tipo; estructurado (bases de datos relacionales, hojas de cálculos, entre otras) el cual representa el 5% de los datos existentes [34], semi-estructurado (tales como; correos electrónicos, XML, redes sociales, todos aquellos que no se ajustan a una estricto estándar) y no estructurados (imágenes, audios, videos y demás) el cual corresponde al 85% de todos los datos existentes [20].

En el tiempo se han agregado otras dimensiones, que complementan la descripción compleja de este fenómeno, de acuerdo con Gandomi, A. and Haider, M. [25] identificaron y describieron las siguientes V's:

4. Valor: característica introducida por Oracle, los datos sin procesar tienen una densidad de bajo valor con respecto al volumen, sin embargo, al ser procesados y transformados, podemos obtener alto valor del análisis de esta enorme cantidad de datos, pudiendo producir estrategias de negocios que nos sirva como apoyo a la toma de decisiones, los cuales pueden ayudar a obtener incremento de ingresos, reducción de costos operativos, mejorar el servicio al cliente, entre otras.

5. Veracidad: IBM acuño esta característica, la cual representa la desconfianza y la incertidumbre latente sobre la fuente de los datos generados por la incompletitud, vaguedad, inconsistencias y subjetividades presentes en los datos. Un ejemplo de ello sería el analizar los sentimientos de los usuarios en las redes sociales para definir una posición o una tendencia con respecto a algo, son de naturaleza incierta, ya que implican un juicio humano, pero con un contenido en información realmente valiosos si se logra descifrarlo. Lidiar con datos imprecisos e inciertos es otra faceta del Big Data, los cuales son enfrentados utilizando técnicas y modelos de análisis basados en inteligencia computacional.

6. Variabilidad: se refiere a la variabilidad en la tasa de transmisión de datos dado sus inconsistencias por intermitencia, o picos de tráfico por largos periodos, esto denota complejidad por conexión, combinación, limpieza, y por la conversión de datos colectados por muchas fuentes.

2.3 Desafíos del Big Data

Debido a la compleja naturaleza del entorno de Big Data, una gran variedad de desafíos es presentados principalmente en actividades de procesamiento y análisis de altos volúmenes de datos, a continuación, se listan alguno de los desafíos más representativos [17][32]:

- **Complejidad en datos:** fuerte aumento en la carga computacional debido a sus difíciles interrelaciones y una variada calidad en los datos, lo cual limita la capacidad para diseñar modelos y métodos computacionales altamente eficientes. Se observa una acumulación de ruido por la alta dimensionalidad y enorme volumen de datos, lo que provoca un elevado costo computacional e inestabilidad algorítmica. Es necesario profundizar en el estudio y conocimiento acerca de la teoría de la complejidad de Big Data, para tener una mayor comprensión de sus características, lo que permitirá simplificar su representación, mejorar la abstracción de conocimiento, y garantizar tanto su valor como su utilidad, disminuyendo la redundancia y facilitando su manejo.

- **Complejidad computacional:** se dificulta el procesamiento y análisis efectivo de los datos, debido a que provienen de múltiples fuentes, en una gran variedad de formatos, en enormes volúmenes y con una alta tasa de cambio. Por ello se hace necesario, un marco informático que este orientado a Big Data el cual se centre en solidas estructuras de comunicación, almacenamiento y cómputo que sea estable, escalable, bien integrado y sobre todo optimizado. De acuerdo con Sivarajah et al [67] las actividades que presentan mayores desafíos con respecto a las cargas computacionales son; agregación e integración de datos, modelado, análisis e interpretación.

- **Complejidad del sistema:** se requiere el diseño de arquitecturas de sistemas que pueda procesar Big Data con un alto rendimiento en la adquisición de datos, bajo consumo de energía y cómputo altamente eficiente, tomando en cuenta las condiciones reales de carga de trabajo, recursos distribuidos y capacidad de cómputo iterativo y paralelizables.

2.4. Áreas de aplicación de Big Data

Los sectores donde Big Data ha tenido aplicación, así como un fuerte impacto positivo, enfrentando desafíos de almacenamiento y análisis, han sido:

- **Internet de las cosas:** actualmente representa uno de los mayores mercados, sus dispositivos y sensores producen grandes cantidades de datos y tienen el potencial para generar tendencias e investigar el impacto de ciertos eventos o decisiones. Su desarrollo y aplicación ha sido llevado a cabo en edificios inteligentes, sistemas ciber-físicos, así como en los sistemas de gestión de tráfico [6].
- **Redes inteligentes:** la analítica de Big Data permite la identificación de transformadores de red eléctrica en riesgo, así como la detección de comportamientos irregulares de dispositivos conectados. Esto permite establecer estrategias preventivas con el propósito de reducir costos por corrección, así como la generación de pronósticos más aproximados de la demanda energética, lo que permite realizar y ejercer un mejor balance de las cargas eléctricas [52].
- **Aerolíneas:** emplea cientos de sensores en cada avión para generar datos sobre toda su flota de aviones, su objetivo es monitorear su desempeño y aplicar mantenimiento preventivo, lo que resulta en ahorros significativos para la empresa [9].
- **E-salud:** utilizado para personalizar servicios de salud, los médicos pueden monitorear los síntomas de sus pacientes con el objetivo de ajustar sus prescripciones. Es considerado muy útil para la optimización de operaciones administrativa de un hospital y reducir costos, CISCO ofrece una de estas soluciones [52].
- **Servicio:** las herramientas de Big Data permiten analizar y monitorear el comportamiento de sus clientes, para cruzar información histórica y actuales de sus preferencias, con el objetivo de ofrecerle un servicio efectivo y personalizado, mejorando así sus estrategias de mercadeo, tal como lo ha hecho el parque Disneyland quienes han introducido un brazalete equipado con radiofrecuencia, la cual permite a sus visitantes evitar esperar en líneas y paseos, esto les permite tener una mejor experiencia de su visita al parque, atrae muchos más clientes e incrementan sus ingresos [9].

- **Usos públicos:** han sido utilizados en complejos sistemas de suministro de agua para monitorear su flujo y detectar fugas en tiempo real, conexiones ilegales y controlar válvulas para un suministro más equitativo en diferentes partes de la ciudad, otro ejercicio también se ha llevado a cabo en el ayuntamiento de Dublín, donde uno de sus servicios más importantes es el transporte, y para ello ha equipado a sus autobuses con sensores GPS para recopilar datos geoespaciales en tiempo real y con esto, a través de su análisis, este puede optimizar sus rutas y el uso de su transporte, lo que permite el ahorro de combustible y la disminución del nivel de contaminación del aire [9][52].

2.5. Inteligencia computacional

Debido a la generación excesiva de los datos en un corto periodo de tiempo, se ha observado la necesidad tanto de tecnología con poder de cómputo como de algoritmos robustos que puedan extraer conocimiento y valor de ellos. Bajo este contexto, una solución que puede abarcar las características representativas y desafiantes del fenómeno de Big Data es la inteligencia computacional (IC).

De acuerdo con Hill, R. [27], IC se centra en replicar el comportamiento humano más que en los mecanismos que generan dicho comportamiento a través del cómputo, son considerados como una subclase del enfoque de Aprendizaje Máquina (AM). Los algoritmos basados en IC permiten modelar la experiencia humana sobre específicos dominios, con el objetivo de proporcionarles la capacidad de aprender y luego adaptarse a nuevas situaciones o entornos cambiantes [22][31][37]. Alguno de estos comportamientos es; abstracción, jerarquías, agregación de datos para la construcción de nuevas conclusiones, información conceptual, representación y aprendizaje a partir de símbolos. El uso de IC requiere centrarse sobre los problemas más que en el desarrollo tecnológico [27]. Sus técnicas brindan un medio para generar modelos que descubran patrones y correlaciones en los datos basados en eventos no vistos, combina elementos de aprendizaje, adaptación, evaluación, razonamiento aproximado, entre otras.

Los algoritmos bio-inspirados se están utilizando cada vez más para trabajar y brindar soluciones a problemas con un alto nivel de complejidad, ya que al ser algoritmos inteligentes poseen la capacidad de aprender del ambiente y adaptarse a nuevos entornos como lo harían los organismos biológicos, otros atributos característicos de este tipo de algoritmos es que pueden tolerar datos incompletos e imprecisos. También pueden aumentar el rango de potenciales soluciones con una mejor aproximación, manejabilidad y solidez [22][36][37][38].

2.6. Sistemas Híbridos Inteligentes

La combinación de diferentes métodos de IC proporciona la oportunidad de construir una gran variedad de Sistemas Híbridos Inteligentes (SHI), lo cual permite extraer las ventajas que cada uno nos ofrece y fusionarlas, para el desarrollo de poderosas herramientas que conduzcan a la solución de problemas de forma efectiva, con las capacidades de aprendizaje e interpretabilidad, obteniendo un buen desempeño y manejo del error muy aproximado al razonamiento humano [56].

IC tiene elementos de aprendizaje, adaptación evolución, y percepción, también integran métodos estadísticos y probabilísticos para un mejor soporte [15]. A continuación, revisaremos algunas de las técnicas asociadas al IC y SHI, y los conceptos sobre los cuales se han basados, con la finalidad de tenerlos como referencia para cuando el modelo desarrollado sea descrito.

2.6.1 Algoritmos Genéticos (AG)

Inspirados en los principios de la genética y selección natural, tienen el objetivo de encontrar la solución óptima de un problema, fue propuesto por John Holland en 1960 [47]. A través de una función de costo, trata de encontrar los valores óptimos, ya sea en máximos o mínimos de un conjunto de parámetros dados [36]. Este tipo de algoritmos han logrado optimizar los sistemas de búsqueda que son difíciles de cuantificar, tal como en aplicaciones financieras, sector industrial, climatología, ingeniería biomédica, control, teoría del juego, diseño electrónico, manufactura automatizada, minería de datos, optimización combinatoria, diagnósticos erróneos, clasificación, calendarización y aproximación de serie de tiempo [22].

En Big Data, los AG han sido aplicados para generar agrupamientos, con el objetivo de lograr un mejor manejo del volumen de datos, dividiendo los datos en pequeños grupos que son considerados su población. También, uno de sus grandes beneficios es que son altamente paralelizables. Pueden combinarse con algoritmos K-means (creado por Stuart Lloyd [44]), la combinación de GK-means toma menos tiempo en memoria y procesa grandes volúmenes de datos en menos tiempo alcanzando muy buenos resultados [30].

2.6.2 Cómputo Granular

Paradigma informático para el procesamiento de la información, trata de imitar la forma en que los humanos procesan la información obtenida de su entorno para comprender un problema. Pueden ser modelados bajo los principios de conjunto difusos, conjuntos aproximados, cómputo de palabras, redes neuronales, análisis de intervalos, entre otras. En 1979, Zadeh introdujo la noción de información granular y sugirió que la teoría de conjuntos difusos podría encontrar posibles aplicaciones con respecto a ello. Sus poderosas herramientas son vitales para la manipulación y entendimiento de la complejidad de Big Data, permitiendo múltiples vistas para el análisis de datos, generando una gran variedad de niveles de granularidad [53].

os gránulos pueden ser representados por subconjuntos, clases, objetos, agrupamientos, y elementos de un universo. Estos conjuntos son construidos a partir de sus distinciones, similitudes, o funcionalidades. Se ha convertido en uno de los paradigmas de información de más rápido crecimiento en el campo de la inteligencia computacional y sistemas centrados en el ser humano [72].

Las técnicas de lógica difusa junto con los conceptos de cómputo granular son considerados una de las mejores opciones para el proceso de la toma de decisiones. Un granulo difuso está definido por restricciones generalizadas, dichos gránulos pueden ser representados por palabras del lenguaje natural. La fusificación de sus gránulos junto con los valores que lo caracterizan, es la forma en la cual los humanos construimos nuestros conceptos, los organizamos y manipulamos. Pueden ser utilizados para reconstruir problemas con un cierto nivel de granularidad (desde los más finos, los cuales pueden estar al nivel de un individuo, hasta los más gruesos que pudieran ser a nivel comunidad), el objetivo se centra sobre el manejo del volumen, característica principal de Big Data, reduciendo su tamaño y creando diferentes perspectivas que pueden ser analizadas y convertirse en indicadores de relevancia para la toma de decisiones [53][72].

El computo granular ha representado una solución alternativa para obtener utilidad y valor de Big Data a pesar de su complejidad. Porque debido a su integración con teorías de inteligencia computacional, estos pudieran ser efectivamente soportados en todos los niveles operacionales, incluidos; adquisición, extracción. Limpieza, integración, modelado, análisis, interpretación y desarrollo [53].

2.6.3 Aprendizaje Máquina

Rama de la inteligencia artificial, la cual se centra sobre la teoría, desempeño, y propiedades de los algoritmos y sistemas de aprendizaje. Forma parte de un campo interdisciplinario el cual está relacionado a la inteligencia artificial, teoría de la optimización, teoría de la información, estadística, ciencia cognitiva, control optimo y otras disciplinas de la ciencia, ingeniería y matemáticas. Su campo está dividido en tres subdominios, los cuales son [74]:

- **Aprendizaje supervisado:** requiere entrenamiento de los datos de entrada y estableciendo una salida deseada. Alguna de las tareas realizadas en el procesamiento de datos es; clasificación, regresión y estimación. Algunos de sus algoritmos más representativos son; máquina de vectores de soporte que fue propuesto por Vladimir Vapnik en 1982 [18], modelo oculto de Markov el cual fue propuesto en 1966 por Leonard E. Batum y Ted Petrie [13], naives bayes [63], redes bayesianas [76], entre otras.

- **Aprendizaje no supervisado:** solo requiere los datos de entrada, sin indicarle el objetivo deseado. Las tareas de procesamiento de dato que ejecuta son de agrupación y predicción, de las cuales algunos de los algoritmos existentes son; modelos de mezcla Gaussiana que fue creado por Karl Pearson's en 1984 [23], X-means [58], entre otros.

- **Aprendizaje por reforzamiento:** permite el aprendizaje desde la retroalimentación recibida a través de la interacción obtenida del ambiente externo. Están orientados a la toma de decisiones y algunos de sus algoritmos son; Q-learning que fue introducido por Zdzislaw Pawlak en 1981 [54], TD learning propuesto por R.D. Sutton en 1988 [19], y Sarsa learning que fue propuesto por Sutton y Barton en 1998 [5].

Las aplicaciones del aprendizaje máquina pueden ser conducidos a través de tres fases primarias [77]:

- **Preprocesamiento:** ayuda a preparar los datos crudos, los cuales por su naturaleza están conformados por; no estructurados, incompletos, inconsistentes y con ruido, a través de la limpieza de los datos, extracción, transformación y fusión pueden ser utilizados en la etapa de aprendizaje como datos de entrada.

- **Aprendizaje:** utiliza algoritmos de aprendizaje para ajustar los parámetros del modelo y estimar la salida de datos deseada, partiendo del preprocesamiento de los datos de entrada.

- **Evaluación:** el desempeño de los modelos de aprendizaje se determina en esta fase, los atributos a los cuales se les presta mayor atención son; selección de conjuntos de datos, medidas de desempeño, estimación del error, pruebas estadísticas para su validación, lo cual permitirá durante el

entrenamiento y aprendizaje ir ajustando de forma apropiada los parámetros del modelo.

El objetivo del aprendizaje máquina es poder descubrir el conocimiento y servir a los tomadores de decisiones para poder generar estrategias informadas e inteligentes. En la vida real, se tienen aplicaciones en motores de búsqueda para recomendación, sistemas de reconocimiento de voz y faciales, minería de datos para el descubrimiento de patrones y extracción de valor, sistemas de control autónomos, entre otras [52]. Google utiliza algoritmos de aprendizaje máquina para la manipulación de grandes volúmenes de datos desordenados.

2.6.4 Redes neuronales artificiales (RNA)

Disciplina que trata de imitar los procesos de aprendizaje del cerebro, replicando la interpretación imprecisa de información obtenida desde los sentidos, aprovechando los beneficios del rápido procesamiento ofrecido por la tecnología informática. Los primeros pasos a la inteligencia artificial vino del neurofisiólogo Warren McCulloch y el matemático Walter Pitts que en 1943 escribieron un artículo acerca de cómo funcionan las neuronas y establecieron el precedente de la creación de un modelo computacional para una red neuronal [38].

Las RNA son también definidas como algoritmos adaptativos de procesamiento no lineal y autoorganizados, con múltiples unidades de procesamiento interconectadas conocidas como neuronas, en una red con diferentes capas, las cuales tienen la capacidad de aprender con base a sus entradas y adaptarse conforme se retroalimenta de su entorno [36].

Su arquitectura está compuesta por tres principales capas, donde cada capa realiza un proceso de mapeo de entradas a salidas. La primera capa está dedicada a organizar las entradas (conocida como capa de entrada), y estas podrían ser cualquier combinación de variables que son importantes para predecir las salidas. Las señales de salida entre los nodos de las capas son generadas a través una función de activación o función de transferencia.

Las capas intermediarias son conocidas como capas ocultas y estas son consideradas como esenciales, debido a que dotan a la RNA de la capacidad de aprender de las relaciones existentes entre los datos. Finalmente, en la capa de salida, los resultados obtenidos son comparados con las muestras de datos objetivos o salidas deseadas, para conocer si la tarea ha sido realizada satisfactoriamente o bien conocer el alcance y desempeño que se logró durante el entrenamiento, las tareas que comúnmente puede realizar una RNA son; clasificación, agrupamiento, predicción, estimación, tiempos de series, entre otras. La arquitectura general de una RNA puede ser observada en la Fig. 1.

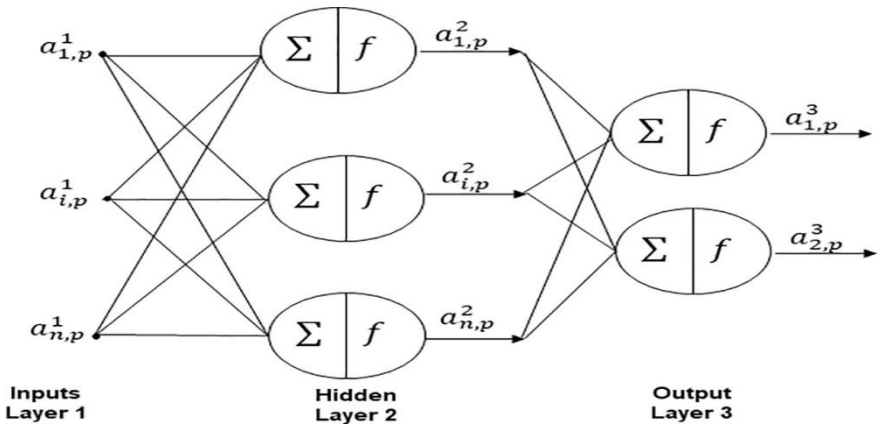


Fig. 1. Arquitectura general de una red neuronal artificial.

Las arquitecturas de las RNA han evolucionado en el tiempo, a continuación, se muestran algunas de ellas son:

- **Redes neuronales Hopfield**, es la más simple de todas, ya que es una red neuronal con una simple capa. Este modelo fue propuesto en 1982 por John Hopfield [21].

- **Red neuronal multicapa prealimentada (feedforward)**, ejecuta su paso hacia adelante, su estructura base está constituida por una capa de entrada, las capas intermedias son conocidas como capas ocultas, de las cuales pueden definirse N número de ellas, y finalmente la capa de salida. Este diseño fue hecho por Broomhead y Lowe in 1988 [49].

- **Redes neuronales autoorganizadas**, tal como los mapas de funciones autoorganizados de Kohonen y el cuantificador vectorial del aprendizaje. Un artículo de Kohonen fue publicado en 1982 [49].

- **Máquinas de aprendizaje extremo,** son redes neuronales de avance. Su principio de aprendizaje es esencialmente un modelo lineal. Tienen un buen desempeño de generalización y el aprendizaje son más rápidas que las redes que utilizan entrenamiento de propagación hacia atrás [14].

- **Redes neuronales convolucionales,** tal como en las RNA, sus neuronas se auto-optimizan a través del aprendizaje. Una de sus diferencias significativas con respecto a las tradicionales RNA es que sus neuronas están organizadas en capas tridimensionales, que se compone de dimensionalidad de entrada (altura y anchura) y profundidad [8].

- **Redes de aprendizaje profundo (deep learning)**, considera dos factores importantes el procesamiento se lleva a cabo en múltiples capas con funciones no lineales, utilizando tanto aprendizaje supervisado como no supervisado [51].

Las propiedades funcionales de las RNA son; el proceso de mapeo es hacia adelante, los pesos se ajustan de forma iterativa después de cada entrenamiento y son almacenadas hasta que el error deseado es alcanzado o se ejecuta el total de épocas definido. Para calcular la medida de error, el algoritmo de retro-propagación (backpropagation) es implementado, el cual tiene una dirección inversa al proceso de pre-alimentación (feedforward). Uno de los algoritmos de optimización más utilizados es el Gradiente Descendente el cual tiene el propósito de minimizar la medida de error [29].

Las RNA son caracterizadas por su adaptabilidad, procesamiento en paralelo y cómputo distribuido, las funciones de procesamiento de la entrada a la salida pueden tener un comportamiento lineal, semi-lineal o incluso no lineal, razonamiento aproximado, aprendizaje autoorganizado, tolerancia a fallas y tiene la capacidad para ajustarse a datos no lineales de forma rápida.

Las neuronas pueden ser definidas y distribuidas de acuerdo con las necesidades del problema [36]. También tienen la capacidad para acercarse a diferentes grados de precisión y reconocer patrones ocultos de datos complejos e inexactos. Son ampliamente utilizados en problemas de control, clasificación agrupación, minado, predicción y reconocimiento de patrones, entre otras [33]. Las RNA son entrenadas y no programadas, son de fácil adaptación a nuevos problemas y pueden inferir relaciones no reconocidas por los programadores [14].

Algoritmo Gradiente Descendente: es uno de los más utilizados para la optimización de la función de error y para ajustar los parámetros de diseño de un modelo en la etapa de entrenamiento de una RNA. Son considerados de primer orden, lo cual se refiere a qué tanto la función decrementa o incrementa desde su primera derivada y un punto de inicio específico, trazando así una línea tangente sobre la superficie de error a partir del punto inicial establecido como puede observarse en la Fig 2.

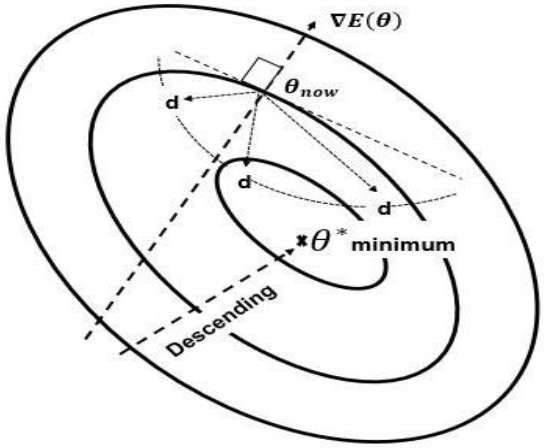


Fig. 2. Posibles direcciones de descenso más pronunciadas obtenidas cuando el descenso de gradiente se calcula desde un punto definido.

Este método tiene la capacidad para definir un vector de direcciones descendentes confiables a través de un proceso iterativo basado sobre información derivada de una función objetivo, donde principalmente busca ajustar los parámetros (pesos) y minimizar el error del modelo durante su proceso de aprendizaje en un espacio de entradas multidimensionales para acercarse a una solución pseudo-óptima. Sus componentes fundamentales para su implementación son:

- **Función de error:** es identificada como una función de costo, el cual resulta de la diferencia entre la respuesta estimada $\hat{\boldsymbol{y}}$ con respecto a la respuesta conocida $\boldsymbol{y}$, una de las medidas de error más utilizadas es la suma de los errores al cuadrado (SSE por sus siglas en ingles), la cual es expresada en la siguiente ecuación (1).

$$E = SSE = \sum_{p=1}^{q} \left(y_p - \hat{y}_p\right)^2 \forall p = 1, \dots, q \tag{1}$$

- **Vector gradiente:** se construye a través de un método eficiente conocido como algoritmo de retro-propagación (backpropagation), en el cual, la derivada parcial de una función de error con respecto a todos los parámetros por cada capa es propagados de forma iterativamente de forma inversa al cálculo de las señales de salida entre capas en la etapa de pre-alimentación (feedforward), esto puede expresarse como se muestra en la ecuación (2).

$$g(\xi) = \nabla E(\xi) \stackrel{\text{def}}{=} \left[\frac{\partial E(\xi)}{\partial \xi_1}, \frac{\partial E(\xi)}{\partial \xi_i}, \frac{\partial E(\xi)}{\partial \xi_n}\right]^T \forall i = 1, .., n \tag{2}$$

  Donde $g(\xi)$ es el vector gradiente para todos los parámetros, E representa a la función de error y ξ son todos los parámetros que serán ajustados durante el entrenamiento.

- **Generalización de la regla delta:** es también conocido como regla de aprendizaje de retro-propagación, el cual es el cambio aplicado a todos los parámetros que serán actualizados o ajustados. El cambio se realiza aplicando una tasa de aprendizaje (este permite mantener el control del descenso, es decir, ajustar su velocidad) al vector gradiente, como se muestra en la ecuación (3).

$$\nabla \xi = -\eta g(\xi) = -\eta \nabla E(\xi) = -\eta \frac{\partial E}{\partial \xi} \tag{3}$$

  Donde $\nabla \xi$ es el cambio direccional, $-\eta$ representa a la tasa de aprendizaje y $g(\xi)$ es el vector gradiente. El cambio direccional se aplica a los parámetros actuales, para obtener nuevos parámetros ajustados y continuar con el proceso de aprendizaje iterativo, como se muestra en la ecuación (4).

$$\xi^{new} = \xi^{old} + \nabla \xi \tag{4}$$

- **Tasa de aprendizaje:** considerado como un hiperparámetro crítico durante el entrenamiento de una RNA, su abreviación es lr (por sus siglas en ingles *learning rate*). Como se mencionó previamente, tiene la bondad de controlar los pasos del descenso en dirección a un pseudo-óptimo valor de mínimo global cuando se utiliza este algoritmo de optimización basados en gradientes. Un valor adecuado de este hiperparámetro podría representar un mejor rendimiento y alcanzar una temprana convergencia evitando el alto costo de procesamiento por excesivas iteraciones o quedarse atascado en algún mínimo local [43]. A continuación, se presentan los comportamientos más representativos de una tasa de aprendizaje cuando su valor es establecido [40]:

  Como puede observarse en la Fig 3. Cuando lr es inicializado con un valor bajo, podría tomar muchos pasos para alcanzar el nivel óptimo y podría tender a caer en mínimos locales, debido a que los pasos en el descenso serían muy pequeños.

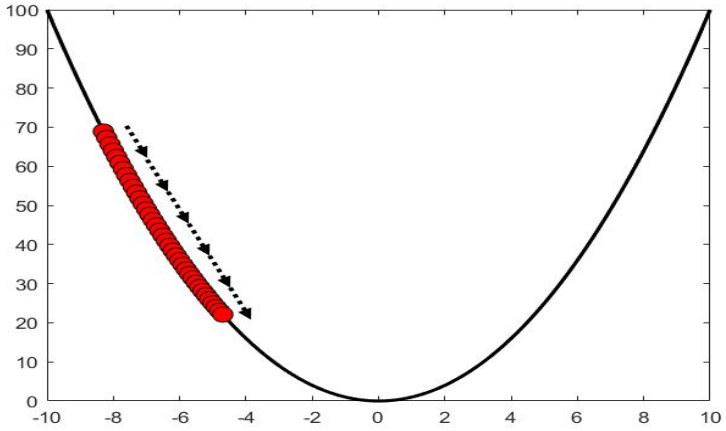


Fig. 3. Comportamiento del gradiente descendente, cuando lr es inicializado con un valor muy bajo.

Otro comportamiento observable en la Fig 4. es el que se da cuando el valor de lr es muy alto, la dirección del gradiente pudiera tornarse irregular, sería mucho más rápido, sin embargo, con altas posibilidades de perder el mínimo global e incluso divergir.

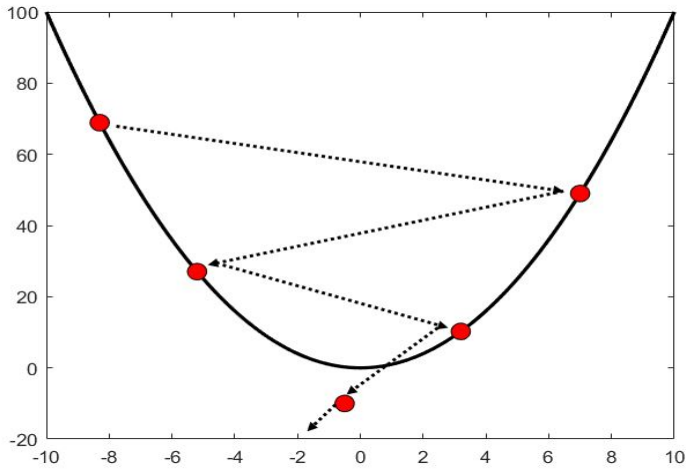


Fig. 4. Comportamiento del gradiente descendente, cuando lr es inicializado con un valor muy alto.

Como puede observarse, es de fácil implementación. Ofrece buenos resultados cuando se trata de optimización no lineal, sin embargo, algunas de sus desventajas se presentan en su rendimiento por su carga computacional, pues a mayor volumen de datos y/o alta dimensionalidad de atributos, su procesamiento iterativo se vuelve lento.

Dada esta problemática, se requiere de mecanismos que puedan controlar la velocidad del descenso de este hiperparámetro, para obtener una adecuada velocidad del gradiente descendente, tratar de encontrar un mínimo global o al menos un buen mínimo local y una temprana convergencia, estos objetivos pueden ser representado por el comportamiento que se muestra en la Fig 5.

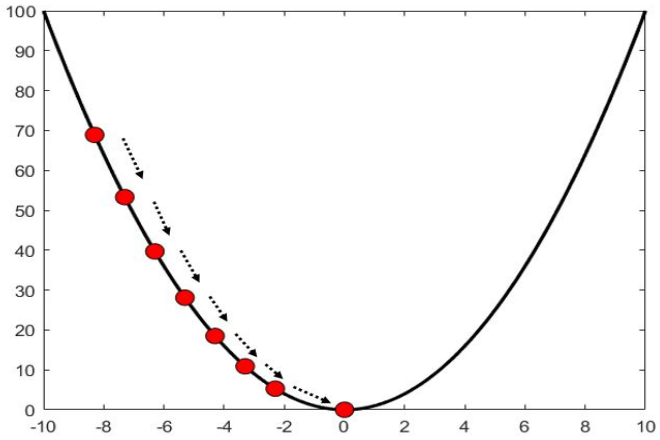


Fig. 5. Comportamiento del gradiente descendente, cuando lr es inicializado con un adecuado valor.

Métodos de aprendizaje basados en el tamaño de la muestra para su procesamiento durante la etapa de entrenamiento

Con el objetivo de aligerar la carga computacional, tratar de encontrar mejores mínimos globales y además lograr una mejor convergencia, existen diferentes métodos de aprendizaje basados en el tamaño, partición de la muestra y la forma en cómo se procesará el vector gradiente y la aplicación de la regla de aprendizaje, estos son [24][29][65]:

- **Método de aprendizaje por lote completo (full batch / offline):** también conocido como entrenamiento por lote (batch training), en este método las derivadas parciales del error son calculadas y acumuladas con respecto a los parámetros de cada dato procesado, esto es, se requiere el procesamiento completo de la muestra de entrenamiento, con la finalidad de construir el vector gradiente, a partir del cual, se puede aplicar la regla de aprendizaje que permitirá actualizar o ajustar los parámetros del modelo.

  Para muestras de tamaño pequeño (número de instancias), se observa un desempeño estable y convergencia aceptable, sin embargo, tiende a estancarse en mínimos locales, además de que, al incrementar el número de instancias, el tiempo de cómputo es excesivo, se vuelve impráctico, por lo que el gasto en cómputo y memoria se vuelve prohibitivo. El comportamiento antes mencionado se puede expresar como se muestra en la ecuación (5):

  $$g(\xi, b) = \nabla E(\xi, b) = \sum_{p=1}^{q} \left(y_p - \hat{y}_p\right)^2 \quad (5)$$

  Donde y_p corresponde a la salida deseada, $\hat{y}_p$ es la salida estimada, y $g(\xi, b)$ representa el vector gradiente construido a partir de la derivada parcial de la función de error con respecto a todos sus parámetros ξ y sus sesgos b, finalmente la ecuación (6) representa la regla de aprendizaje:

  $$\xi^{new} = \xi^{old} - \eta g(\xi, b) \quad (6)$$

  Donde η representa a la tasa de aprendizaje, ξ^{old} corresponde a los parámetros de diseño actuales, y para el cual el cambio obtenido de la tasa de aprendizaje aplicado al vector gradiente acumulado, permitirá actualizar dichos parámetros de diseño representado por ξ^{new}.

El siguiente pseudocódigo trata de representar el comportamiento funcional de este método de aprendizaje:

1. ***Mientras*** que el número de épocas ***no alcance*** el máximo definido
2. ***Para cada*** *dato en la muestra de entrenamiento*
3. *Se calcula el gradiente para todos los parámetros* $g(\xi_p, b)\ \forall\ p =$ *número de dato*
4. *Se obtiene el gradiente y este es acumulado* $g(\xi, b) = \sum_{p=1}^{q} g(\xi_p)\ \forall\ p = 1, .., q$
5. ***Fin***
6. ***Fin***
7. *El gradiente acumulado y la tasa de aprendizaje son utilizados para actualizar los parámetros de diseño* $\xi^{new} = \xi^{old} - \eta g(\xi, b)$

- **Método de aprendizaje en línea (online):** también conocido como entrenamiento en línea (online training), en este método, la actualización de los parámetros de diseño se realiza cada vez que un dato es procesado.

  Este tipo de procedimiento conduce al cálculo de un gradiente aproximado, su principal ventaja es el aumento de la velocidad, es adaptable ya que no depende de la distribución de los datos, a diferencia del método de aprendizaje por lote completo, disminuye en gran medida el costo de cómputo y la carga de datos en memoria, puede trabajar con facilidad en entornos en tiempo real, sin embargo, su descenso se muestra inestable, debido al ruido implícito en cada dato de la muestra (alta variabilidad), lo que a su vez podría ser beneficioso, ya que sus saltos podrían interpretarse como potenciales mínimos locales e incluso alcanzar un mejor mínimo global. Dicho comportamiento puede ser representado como se muestra en la ecuación (7):

$$\xi^{new} = \xi^{old} - \eta g\left(\xi, b, x^{(p)}, y^{(p)}\right) \tag{7}$$

Donde $g(\dots)$ representa el vector gradiente, los parámetros dentro de la función corresponde a; ξ parámetros de diseño, b sesgo, $x^{(p)}$ entradas, $y^{(p)}$ salida deseada, finalmente p representa el índice o la posición del dato que será procesado.

El siguiente pseudocódigo trata de representar el comportamiento funcional de este método de aprendizaje:

1. ***Mientras*** que el número de épocas ***no alcance*** el máximo definido
2. ***Para cada*** *dato en la muestra de entrenamiento*
3. *Se calcula el gradiente para todos los parámetros* $g\left(\xi, b, x^{(p)}, y^{(p)}\right) \forall$
4. $p =$ *número de dato*
5. *El gradiente obtenido y la tasa de aprendizaje son utilizados para actualizar*
6. *los parámetros de diseño* $\xi^{new} = \xi^{old} - \eta g\left(\xi, b, x^{(p)}, y^{(p)}\right)$
7. ***Fin***
8. ***Fin***

- **Método de aprendizaje por mini-lotes (mini-batch):** también conocido como entrenamiento por mini-lotes (mini-batch training). Este método mejora las dificultades presentadas tanto en el método de aprendizaje por lotes completo como en el método de aprendizaje en línea, por un lado, trata de reducir la alta variabilidad generada al calcular el gradiente por cada dato, comportamiento dado en el entrenamiento en línea, por otra parte, el alto costo computacional y la excesiva carga de datos en memoria cuando tiene que procesarse la muestra completa como es el caso del entrenamiento por lotes completo.

  El método de aprendizaje mini-lotes trata de tomar ventajas de los métodos antes mencionados, a partir de la partición de la muestra de datos de entrenamiento en particiones más pequeñas, calculando y acumulando por cada partición pequeña el gradiente como en el entrenamiento por muestra completa y actualizando los parámetros de diseño como se hiciera en el entrenamiento en línea dentro de la misma época y no hasta la finalización de esta.

  Esta combinación puede conducir a un descenso estable, mejorar la velocidad del descenso y reducir la variabilidad, este método puede trabajar con un buen desempeño en ambientes paralelizables y distribuidos, recomendado para utilizarse en entornos de Big Data y Deep Learning.

  Este método es el resultado de la acumulación de sus derivadas parciales por cada mini-lote procesado, donde el vector gradiente es construido desde la función de error promediada entre el mini-lote en proceso, que posteriormente se utilizará para actualizar y ajustar los parámetros de diseño. Dicho comportamiento puede ser representado bajo la siguiente ecuación (8):

$$g\left(\xi, b, x^{(p:bs)}, y^{(p:bs)}\right) = \frac{1}{bs}\sum_{p=0}^{bs} E\left(\xi, b, x^{(p)}, y^{(p)}\right) \quad (8)$$

Donde $g(\dots)$ representa el vector gradiente, los parámetros dentro de la función por cada mini-lote corresponde a; ξ parámetros de diseño, b sesgo, $x^{(p:bs)}$ entradas del mini-lote, $y^{(p:bs)}$ salida deseada del mini-lote, finalmente p representa el índice o la posición del dato que será procesado y bs representa el tamaño del mini-lote que será procesado durante el entrenamiento. Finalmente, la ecuación (9) representa la regla de aprendizaje, la cual permitirá actualizar y ajustar los parámetros de diseño.

$$\xi^{new} = \xi^{old} - \eta g\left(\xi, b, x^{(p:bs)}, y^{(p:bs)}\right) \tag{9}$$

El siguiente pseudocódigo trata de representar el comportamiento funcional de este método de aprendizaje:

1. ***Mientras*** *que el número de épocas* ***no alcance*** *el máximo definido*
2. ***Mientras*** *el número de mini-lotes* ***no alcance*** *el límite total de mini-lotes*
3. ***Para cada*** *dato en la muestra de entrenamiento del mini-lote*
4. *Se calcula el gradiente para todos los parámetros* $g\left(\xi, b, x^{(p:cbs)}, y^{(p:cbs)}\right)$
5. $\forall\, p =$ *número de dato;* $cbs =$ *tamaño actual del mini-lote*
6. *Por cada dato procesado del mini-lote, se obtiene el gradiente y este es*
7. *Acumulado*
8. $g\left(\xi, b, x^{(p:bs)}, y^{(p:bs)}\right) = \frac{1}{bs}\sum_{p=1}^{bs} g\left(\xi, b, x^{(p:bs)}, y^{(p:bs)}\right)$
9. $\forall\, p =$ *número de dato;* $bs =$ *total de mini-lotes*
10. ***Fin***
11. *El gradiente acumulado y la tasa de aprendizaje son utilizados para actualizar los parámetros de diseño* $\xi^{new} = \xi^{old} - \eta g\left(\xi, b, x^{(p:bs)}, y^{(p:bs)}\right)$
12. $\forall\, p =$ *número de dato;* $bs =$ *total de mini-lotes*
13. ***Fin***

2.6.5 Lógica Difusa

Todas las actividades humanas tienen incertidumbre implícita, nuestra comprensión está basada en gran medida sobre el razonamiento humano impreciso, dicha vaguedad es de utilidad cuando se requiere tomar decisiones, pero por otro lado resulta en procesamientos complejos para las computadoras.

A partir de 1920, Lukasiewiscz habló sobre el hecho de que los valores en los sistemas lógicos no eran más que una lógica con valores continuos [78]. En 1965, Zadeh logró cristalizar su idea de la indefinibilidad cointensiva, por la cual el argumentaba que era una medida cualitativa de la proximidad de los significados de precisión, a partir del cual creo su concepto de grado de pertenencia, que ha sido una de las bases fundamentales para el desarrollo de la teoría de conjuntos difusos [66][68].

La lógica difusa fue introducida en 1975 por Zadeh en su artículo titulado “Lógica difusa y el razonamiento aproximado”. Su inspiración está basada en el razonamiento de la mente humana que es aproximada más que exacta, dando más importancia al significado que a la precisión de la información resultante [71]. Por ejemplo, cuando un objeto está a punto de caer sobre la cabeza de una persona, la información importante para esta persona es saber que un objeto caerá sobre él (razonamiento aproximado) y no el peso, la forma, trayectoria y velocidad con la que el objeto caerá sobre el (precisión).

La definición de un comportamiento complejo no puede ser expresado con precisión o exactitud, en su lugar, necesitamos un sistema que pueda tolerar las inexactitudes, información incompleta, percepción, experiencia, y juicios, por esta razón, la lógica difusa requiere conceptos tales como; conjuntos difusos, variables lingüísticas, y reglas si-entonces (if-then) para construir sistemas robustos.

Estructura General de un Sistema Difuso

A continuación, describiremos cada uno de los elementos que componen la arquitectura general de un sistema difuso, los cuales se observan en la Fig.6.:

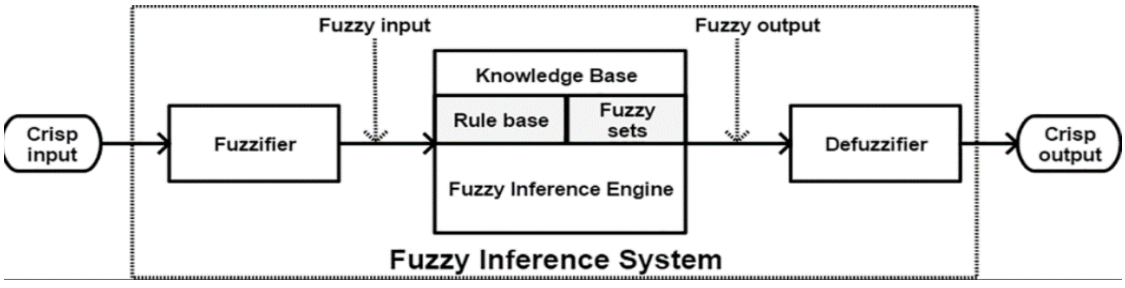


Fig. 6. Arquitectura general de un sistema difuso.

(1) Entradas Crisp: son las mediciones recolectadas del dominio numérico o subyacente de la cual se compondrá nuestro universo de discurso. Se puede visualizar como un conjunto de datos sobre particiones bien definida, donde no hay incertidumbre sobre su pertenencia y donde la transición de una partición a otra es a través de un salto brusco, los cuales han sido definido por sus límites, como se observa en la Fig. 7.

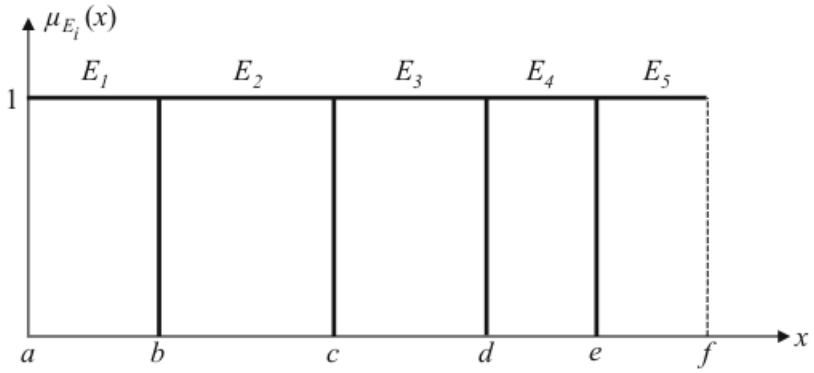


Fig. 7. Conjuntos y particiones CRISP.

No existe incertidumbre sobre la pertenencia de **x** a alguna de las particiones **E_1 a E_5**, si **x** pertenece a una partición tendrá un **valor de membresía de 1 para dicha partición, de lo contrario será 0**, pero de ninguna manera podrá pertenecer a más de una partición.

(2) Fuzzificación: es el proceso que se encarga de convertir (mapear) una **entrada CRISP** a un **valor difuso,** el cual se realiza mediante el uso de la información en la base de conocimiento. Se considera como el paso para determinar el grado de pertenencia de los datos de entrada a cada uno de los conjuntos difusos definidos. Se puede representar de la siguiente manera; vector de entrada $x\left(x_{1,\dots}, x_p\right)^T \in X_1 x X_2 x \dots x X_p \equiv X$. Existen dos tipos de fuzzificadores, estos son:

- **Singleton:** un conjunto difuso que contiene un **único elemento,** es aquel para el cual cada uno de los valores del universo discurso $(i = 1, \dots, p)$ es mapeado sobre la función de membresía asociada al conjunto difuso, y en el punto donde toca a la función de membresía es considerado su fuerza de disparo, siendo su valor máximo 1 , como se representa a continuación $mf_{X_i}(\acute{x_\iota}) = 1$ y para aquellos que no presentan un nivel de activación su valor será 0, $mf_{X_i}(x_i) = 0$ para $x_i \in X_i \; y \; x_i \neq \acute{x_\iota}$, como se puede observar en la Fig.8.

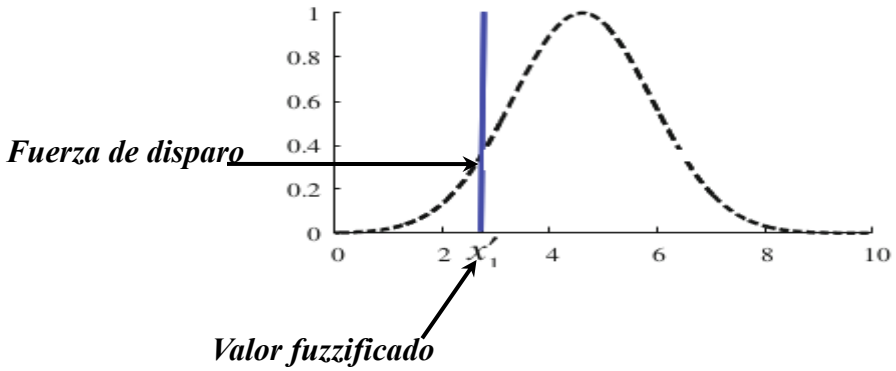


Fig. 8. Fuzzificador singleton.

Al mapear la entrada crisp en un conjunto difuso que solo tiene un punto único en su soporte, el fuzzificador singleton no modelará ninguna incertidumbre sobre la entrada, por lo que no hacen completo uso del modelado de las capacidades del fuzzificador.

- **Non-singleton:** implica que dado un valor de entrada $\acute{x}_1$es el valor más probable para ser el correcto de todos los valores en su vecindad inmediata, cuando las señales son corrompidas por el ruido, puntos adyacentes pudieran ser también ser valores correctos, pero en un menor grado, una amplia extensión indica que hay más incertidumbre inherente en los datos, como puede observarse en la Fig. 9.

El fuzzificador non-singleton mapeará las mediciones x_i donde $(i = 1, \ldots, p)$, y $x_i = \grave{x}_\iota$ se convertirá en un número difuso de tipo-1, para el cual si presenta un nivel de activación su fuerza de disparo podrá alcanzar como valor máximo 1 $mf_{X_i}(\grave{x}_\iota) = 1$ y para $mf_{X_i}(x_i)$ decrementará desde la unidad $x_i$alejándose de $\grave{x}_\iota$**.** Ya que la función de membresía para $\grave{x}_\iota$será el de un número difuso de tipo-1, expresado como $mf_{X_i}(x_i|\grave{x}_\iota)$.

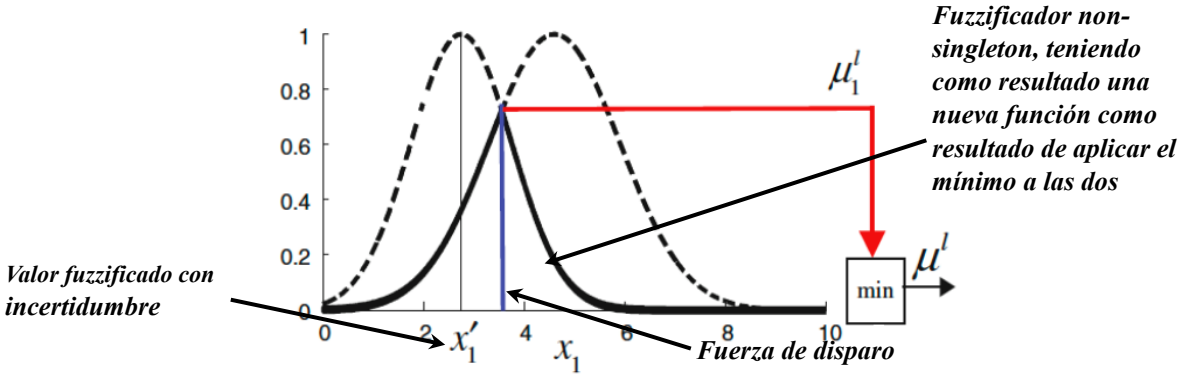


Fig. 9. Fuzzificador non-singleton.

(3) Regla Difusa: consideradas como la base de conocimiento, donde se almacenan las reglas **SI-ENTONCES** (IF-THEN), las cuales son obtenidas del conocimiento de los expertos, su estructura general se muestra a continuación en la Fig. 10.

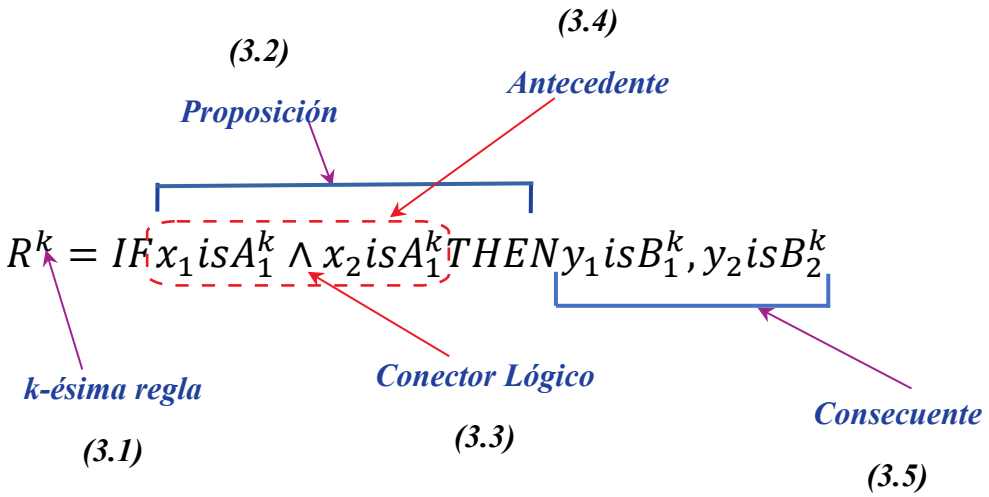


Fig. 10. Estructura general de una regla difusa.

Descripción de los elementos en la regla difusa:

(3.1) k-ésima regla: podemos generar tantas reglas sea necesaria, pudiendo estas ser simples o compuestas, tanto en antecedentes como en consecuentes.

(3.2) Proposición: En filosofía y lógica, el término de proposición se utiliza para referirse a:

- Las entidades portadoras de los valores de verdad, en lógica difusa serían grados de verdad.
- Expresa un contenido semántico, al cual es posible asignarle un valor de verdad.

(3.3) Conector lógico: permitirá **asociar dos o más proposiciones**, así como también decidir el **tipo de operación que se evaluará sobre dichas proposiciones**, generando las siguientes acciones, según el **conector lógico invocado**:

- **And** $(\wedge)$**:** $min\left(mf_A(x), mf_B(y)\right)$ ejecuta la **función mínima**, de los **grados de membresía** obtenidos de los conjuntos **A** y **B.**

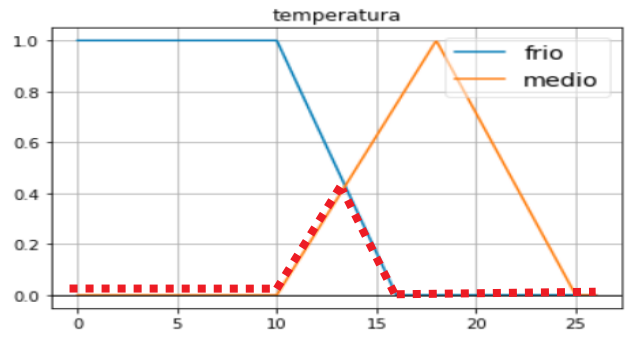


Fig. 11. Muestra gráfica del operador AND.

Esta técnica es útil, cuando se desea **identificar el mínimo denominador común** para la pertenencia de todos los criterios de entrada. **También se conoce como intersección o t-norm.** En la Fig. 11, se muestra gráficamente la representación del conector AND.

- **OR** ∨**:** $max(mf_A(x), mf_B(y))$ ejecuta la **función de máximo**, de los grados de membresía obtenidos de los conjuntos difusos **A** y **B**, para un valor (x, y) dados. En la Fig. 12, se muestra gráficamente la representación del conector OR.

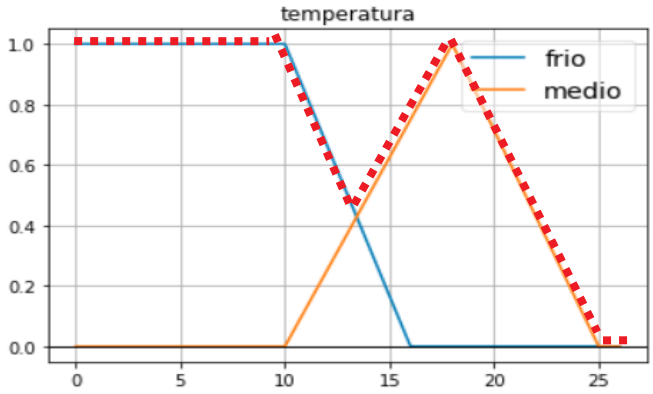


Fig. 12. Muestra gráfica del operador OR.

Esta técnica es útil, cuando se desea ***identificar el máximo denominador común*** para la pertenencia de todos los criterios de entrada. ***También se conoce como unión o t-conorm.***

- ***NOT* ¬*:*** $mf_{notA}(x) = 1 - mf_A(x)$ es el resultado de la **diferencia del universo (1) menos el grado de membresía del conjunto difuso A**, para un valor dado en x. En la Fig. 13, se muestra gráficamente la representación del conector NOT.

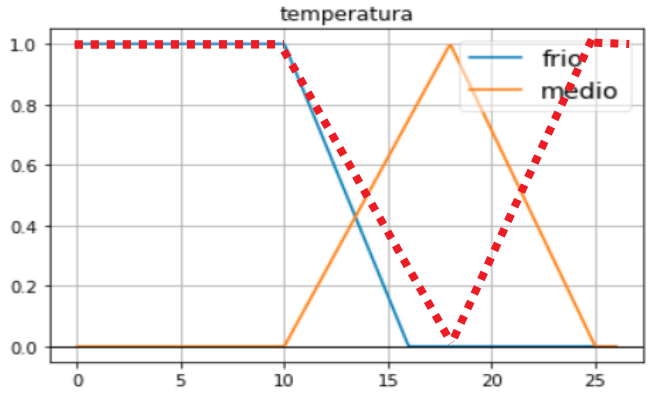


Fig. 13. Muestra gráfica del operador NOT, para función de membresía triangular.

T-norma y T-conorma son consideradas como disyunción y conjunción generalizada respectivamente y son conocidas como implicación difusa. La implicación es una de las principales conectivas en cualquier sistema lógico, y tiene influencia muy seria en el desempeño de los sistemas en los que se emplean técnicas de lógica difusa [75].

Propiedades de las operaciones para conjuntos difusos.

A continuación, se describen las propiedades de las operaciones de los conjuntos difusos en la tabla 1.

Tabla 1. Propiedades de las operaciones para conjuntos difusos.

Propiedades de Operación	**Descripción**	**Unión**	**Intersección**
Conmutativa	No importa el orden de valores de membresía los conjuntos difusos sean comparados, siempre se podrá extraer sus valores máximos o mínimos.	$A \cup B = B \cup A$ ***Función máxima*** ***(operador lógico OR)*** $max(mf_A, mf_B) = max(mf_B, mf_A)$	$A \cap B = B \cap A$ ***Función mínima*** ***(operador lógico AND)*** $min(mf_A, mb_B) = min(mf_B, mf_A)$
Asociativa	La operación máximo y mínimo son asociativas, porque nos permite cambiar el orden de asociación.	$A \cup (B \cup C) = (A \cup B) \cup C$ ***(1)*** $max\big(mf_A, max(mf_B, mf_C)\big)=$ ***(2)*** $max(mf_A, mf_B, mf_C)=$ ***(3)*** $max(max(mf_A, mf_B), mf_C)$	$A \cap (B \cap C) = (A \cap B) \cap C$ ***(1)*** $min\big(mf_A, min(mf_B, mf_C)\big)=$ ***(2)*** $min(mf_A, mf_B, mf_C)=$ ***(3)*** $min(min(mf_A, mf_B), mf_C)$
Distributiva	Se refiere a como se distribuye la unión con respecto a la intersección y viceversa, en conjuntos difusos y en términos de máximos y mínimos.	$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ ***(1)*** $max\big(mf_A, min(mf_B, mf_C)\big)=$ ***(2)*** $min\big(max(mf_A, mf_B), max(mf_A, mf_C)\big)$	$A \cap (B \cup C) = (A \cup B) \cap (A \cup C)$ ***(1)*** $min\big(mf_A, max(mf_B, mf_C)\big)=$ ***(2)*** $max\big(min(mf_A, mf_B), min(mf_A, mf_C)\big)$
Identidad	Se toma el valor de membresía	$A \cup \emptyset = A$	$A \cap \emptyset = \emptyset$

	máximo del conjunto difuso y el valor de membresía máximo del conjunto vacío que sería 0, por lo tanto, el valor máximo siempre será el del conjunto de difuso que no está vacío, sea cual sea su valor de membresía. Lo mismo para la unión del conjunto difuso con el universo, dado que el universo equivale a 1, al obtener su valor máximo entre el conjunto difuso y el universo, siempre será igual al universo, es decir, 1. La misma lógica, pero de forma inversa se aplicaría al mínimo (intersección).	***(1)*** $max(mf_A, 0) = mf_A$ $mf_A \cup X = X$ ***(2)*** $max(mf_A, 1) = 1$	***(1)*** $min(mf_A, 0) = 0$ $mf_A \cap X = X$ ***(2)*** $min(mf_A, 1) = mf_A$
Transitiva	Nos dice que si para los conjuntos difusos de: A es subconjunto de B, y B subconjunto de C, por lo tanto, A es subconjunto de C.	$SiA \subseteq ByB \subseteq CentoncesA \subseteq C$ $Simf_A \leq mf_B ymf_B \leq mf_C entoncesmf_A \leq mf_C$	
Idempotencia	Si se realiza la unión de un conjunto difuso con el mismo, el resultado sería dicho conjunto difuso. De la misma forma sería para la intersección.	$A \cup X = X$ $max(mf_A, mf_A) = mf_A$	$A \cap A = A$ $min(mf_A, mf_A) = mf_A$

Leyes DeMorgan	El complemento de la unión de dos conjuntos difusos es igual a la intersección de sus complementos. Y de forma viceversa en la intersección.	$\overline{A \cup B} = \bar{B} \cap \bar{A}$ **(1)** $1 - max(mf_A, mf_B) =$ $min(1 - mf_B, 1 - mf_A)$	$\overline{A \cap B} = \bar{B} \cup \bar{A}$ **(1)** $1 - min(mf_A, mf_B) =$ $max(1 - mf_B, 1 - mf_A)$
Involutiva	Si se saca el complemento del complemento de un conjunto, se llegará al mismo complemento.	$\bar{\bar{A}} = A$ $mf_{\bar{\bar{A}}} = 1 - mf_{\bar{A}} = 1 - (1 - mf_A) = mf_A$	

La **superposición difusa permite analizar la posibilidad de que un fenómeno pertenezca a varios conjuntos difusos**, además permite **analizar las relaciones entre las pertenencias de los diversos conjuntos.** Como se mencionó anteriormente, también se pueden realizar las siguientes **operaciones entre los pares de proposiciones:**

- **Producto algebraico:** $mf_{A\cdot B}(x) = mf_A(x).mf_B(x)$ este tipo de superposición de producto algebraico **multiplicará cada uno de los valores de membresía de los diferentes conjuntos difusos,** como se observa en la Fig. 14. El producto resultante será menor que cualquiera de las entradas, más aún cuando dicha entrada pertenece a más de un conjunto difuso.

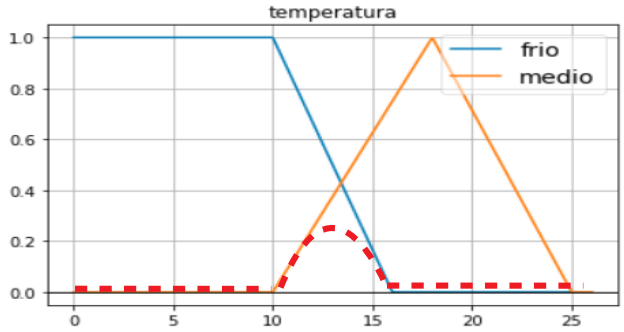


Fig. 14. Producto Algebraico.

Nota: Es difícil correlacionar el producto de todos los criterios de entrada con la relación relativa de los valores. **Esta opción no se utiliza con frecuencia**, sin embargo, **cuando se requiere optimizar los parámetros basado en gradientes, se utiliza tanto producto como suma algebraica, ya que son funciones derivables.**

- **Suma algebraica:** $mf_{A+B}(x) = mf_A + mf_B - mf_A.mf_B$ **sumará cada uno de los valores de membresía de los diferentes conjuntos difusos, así como también restará la multiplicación de los grados de membresía de tales conjuntos difusos**, como se muestra en la Fig. 15.

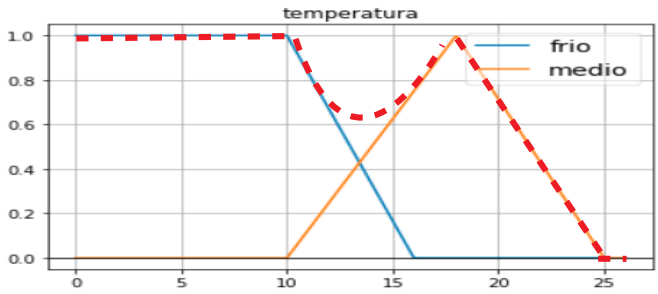


Fig. 15. Suma algebraica.

(3.4) Antecedentes: es la primera parte de una proposición que va seguido de la palabra **SI**, la cual **nos permitirá generar una condición asociando la variable difusa con un término lingüístico a través de la palabra ES**, en la Fig. 16 se muestra un ejemplo de este:

Fig. 16. Ejemplo de la composición de una proposición antecedente simple.

A continuación, se explican los elementos que componen una proposición antecedente:

(3.4.1) Variable Difusa: es una **variable cuyos posibles valores son palabras que describen su comportamiento**, y pueden ser representados mediante conjuntos difusos. **Permite la representación de aquello que no podamos representar en términos numéricos.**

- Está caracterizado por una **quíntupla:**

(X, T (X), U, G, M)

(1) **X**: nombre de la **"variable lingüística" (variable difusa)**

(2) **T(X):** es el **"conjunto de términos lingüísticos" (conjunto difuso).**

(3) **U: "universo de discurso" o dominio.**

(4) **G:** es una **gramática libre de contexto** mediante la que se generan los **términos en T(X),** como podría ser; **muy alto, alto, medio…**

(5) **M:** es una **regla semántica que asocia a cada valor lingüístico** de **X** su **significado M(X)**, este último denota un subconjunto difuso en **U**.

- Los símbolos terminales de las gramáticas incluyen:
  - **Términos primarios**, tales como; **bajo, alto…**
  - **Modificadores,** tales como; **muy, más, menos…**

- **Conectores lógicos,** tales como; **NOT, AND y OR.**

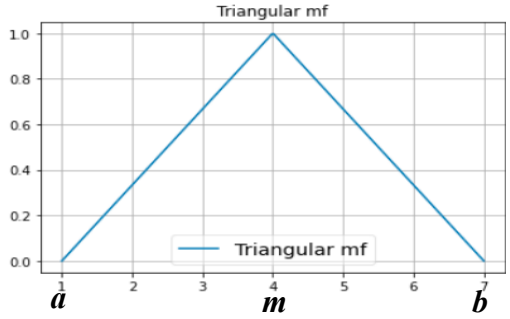
- Un uso habitual de las variables lingüísticas es en **las reglas difusas.**

(3.4.2) Término Lingüístico: son **descripciones vagas** de una **variable difusa**, y al **conjunto de este término lingüístico** que describen el comportamiento de la variable lingüística, **se le denomina conjuntos difusos**, los cuales están asociados a un dominio numérico o subyacente.

Cada **término lingüístico** es **asociado** a una **función de membresía**, las cuales **ajustan** su **dominio subyacente** a una **función de distribución de probabilidad de cada término lingüístico en un conjunto difuso**, cada uno de los términos lingüístico pudiera tener una función de membresía distinta a la otra. Las funciones de membresía más utilizadas son:

- **Triangular:** permite **describir una población** sobre la cual **existen datos de muestra limitados,** es una **distribución continua y de forma triangular** (pudiendo ser simétrica o asimétrica), la cual está **descrita por sus valores mínimo, máximo y moda**. Aumenta de forma lineal hasta alcanzar el valor pico de la moda y luego disminuye de forma lineal hasta alcanzar el punto máximo, como se observa en la Fig. 17.

$$mf_A(x) \begin{cases} 0, six \leq a \\ \frac{x-a}{m-a}, sia < x \leq m \\ \frac{b-x}{b-m}, sim < x < b \\ 0, six \geq b \end{cases}$$



$$triangular(x; a, m, b) = max\left(min\left(\left(\frac{x-a}{m-a}\right), \left(\frac{b-x}{b-m}\right)\right), 0\right)$$

Fig. 17. Definición de una función triangular.

- **Gaussiana:** distribución continua para la cual sus parámetros son **centro** (pico o centro de la curva en forma de campana) y **sigma** (desviación estándar la cual es la dispersión de la distribución, cuanto más pequeña más estrecha la campana), como se muestra en la Fig. 18.

$$Gaussiana(x; sigma, centro) = e^{-\left(\frac{x-centro}{sigma}\right)^2}$$

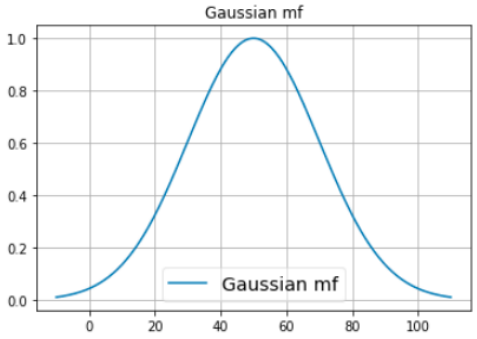


Fig. 18. Definición de una función gaussiana.

- **Logística:** utilizada para modelar distribuciones de datos que tengan colas más grandes y curtosis más alta que la distribución normal. Sus parámetros son **escala** (**a)** y **distribución (c)**. Se parece a la forma de distribución gaussiana a diferencia que esta tiene colas más grandes, como se observa en la Fig. 19.

$$logistica(x; a, c) = \frac{2}{1 + e^{\left(\frac{x-c}{a}\right)^2}}$$

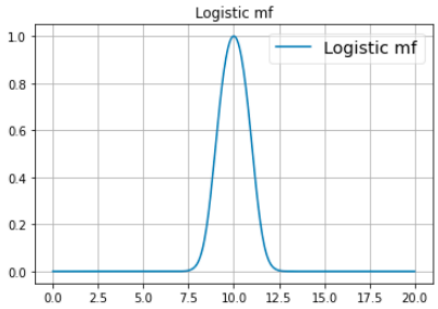


Fig. 19. Definición de una función logística.

- **Tangente:** La tangente hiperbólica de un ángulo **x** es la **relación del seno y coseno hiperbólico**. Puede ser utilizada para definir una medida de distancia en cierto tipo de geometría no euclidiana. Distribución parecida a la logística y de igual forma sus parámetros son **escala (a)** y **distribución (c)**, como se observa en la Fig. 20.

$$tangente(x; a, c) = \frac{e^{2x}-1}{e^{2x}+1} = 1 + tanh\left(-1 * \left(\frac{x-c}{a}\right)^2\right)$$

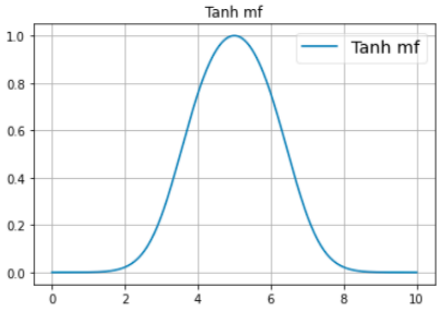


Fig. 20. Definición de una función tangente.

- **Sigmoide:** distribución que **permite mostrar progresión temporal iniciando en un nivel bajo hasta acercarse a un clímax transcurrido en cierto tiempo**. Los parámetros (**a)** corresponde a la **magnitud** que controla el **ancho del área** de transición y (**c)** define el **centro del área de transición**, como se muestra en la Fig. 21.

$$sigmoide(x; a, c) = \frac{1}{(1+e^{-a*(x-c)})}$$

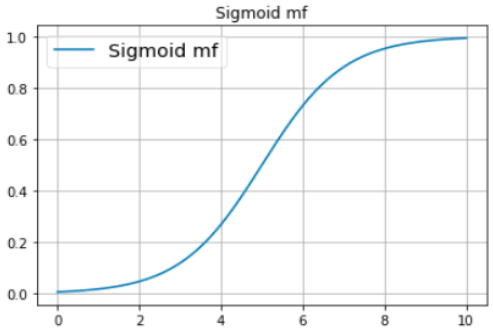


Fig. 21. Definición de una función sigmoide.

- **Cauchy:** distribución continua que se define por sus **parámetros de ubicación y escala**, muy utilizada en física. Es representada con una curva en forma de campana, similar a una distribución normal, sin embargo, **bajo esta distribución sus colas se aproximan a cero con mayor lentitud que las colas de distribución normal**. Los parámetros **(a)** corresponde a la **ubicación del pico** de la distribución y **(e) escala** el cual representa la **dispersión de la distribución**, como se observa en la Fig. 22.

$$cauchy(x; a, c) = \frac{1}{1+\left(\frac{x-c}{a}\right)^2}$$

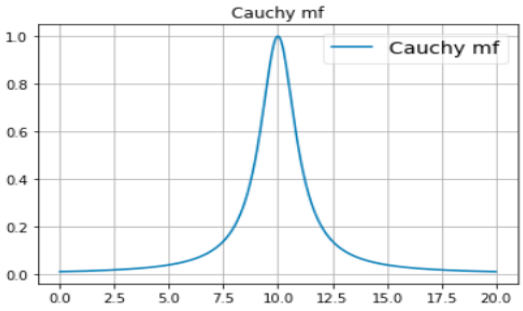


Fig. 22. Definición de una función cauchy.

- **Trapezoidal:** es definida por un **límite inferior (a)**, **límite superior (d)**, **un límite de soporte inferior (b)** y un **límite de soporte superior (c)**, tal que, $a < b < c < d$, como se muestra en la Fig. 23.

$$trapezoidal(x; a, b, c, d) = max\left(min\left(\frac{x-a}{b-a}, 1, \frac{d-x}{d-c}, 0\right)\right)$$

$$mf_A(x) \begin{cases} \frac{x-a}{b-a}, Si(a \leq x \leq b) \\ 1, Sib \leq x \leq c \\ \frac{d-x}{d-c}, Sic \leq x \leq d \end{cases}$$

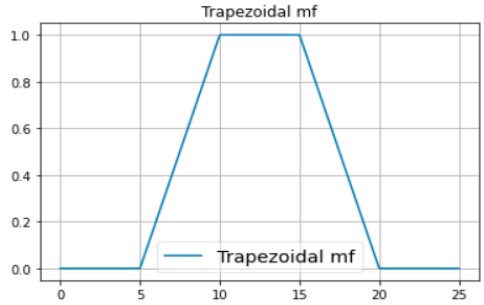


Fig. 23. Definición de una función trapezoidal.

- **Campana generalizada:** se especifica mediante los siguientes parámetros; **(a)** define el **ancho de la función de membresía**, mientras más grande el valor más ancho la función de membresía, **(b)** define la **forma de la curva** a cada lado de la meseta central, donde un valor mayor crea una transición más pronunciada y **(c)** define el **centro de la función de membresía**, como se muestra en la Fig. 24.

$$gbell(x; a, b, c) = \frac{1}{1 + \left|\frac{x-c}{a}\right|^{2b}}$$

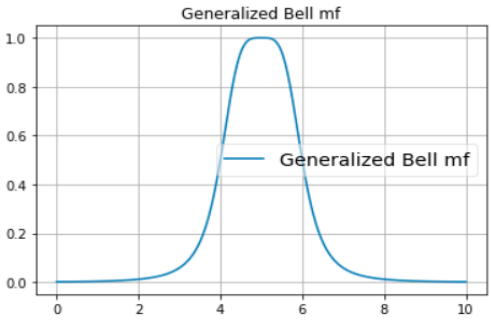


Fig. 24. Definición de una función de campana generalizada.

(3.5) Consecuentes: es la segunda parte de una proposición hipotética seguida de la palabra **ENTONCES**, el consecuente es una **proposición que evalúa el grado de verdad de una variable difusa de salida**, si la proposición antecedente evaluada es verdadera. Igualmente, como en la parte consecuente, los conjuntos difusos relacionado a la variable difusa de salida asignada al consecuente, estará asociada a una función de membresía, a continuación, se muestra un ejemplo en la Fig. 25.

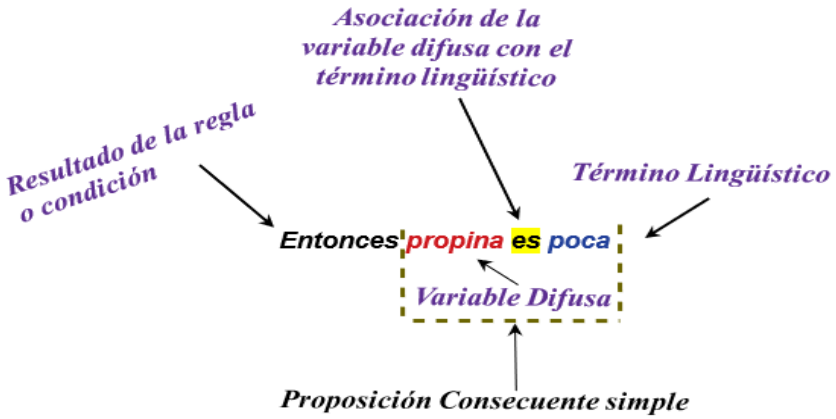


Fig. 25. Ejemplo de la composición de una proposición consecuente simple.

(4) Inferencia difusa: es el **proceso de obtener un valor de salida dado un valor de entrada empleando la teoría de conjuntos difusos**. Simula el razonamiento humano haciendo inferencias sobre las entradas recibidas y las reglas difusas almacenadas. A continuación, veremos dos tipos de inferencias:

(4.1) Modelo de Mamdani: en este modelo la salida de cada regla será un conjunto difuso derivado de la función de membresía de salida y los métodos de implicación del sistema de inferencia difuso, este tipo de modelo se considera más intuitivo y adecuado para el manejo de entradas lingüísticas, por lo que facilita el entendimiento de las reglas bases. A continuación, se presenta un sistema difuso basado en el modelo de inferencia Mamdani, el cual calcula el porcentaje de propina con base a la calificación otorgada al servicio y a la comida, iniciamos describiendo cada uno de los elementos necesarios para la construcción del sistema de inferencia difusa:

(4.1.1) Definición de las variables difusas de entrada/salida y los términos lingüísticos que componen sus respectivos conjuntos difusos, se muestran en las tablas 2 - 4. Estos son:

Tabla 2. Variable difusa de entrada "servicio" y la definición de su conjunto difuso.

Variable Difusa de Entrada	Términos Lingüísticos	Función de Membresía y sus parámetros (sigma y centro)
Servicio	Pobre	Gaussiana ([1.5,0])
	Bueno	Gaussiana ([1.5,0])
	Excelente	Gaussiana ([1.5,0])

Así como la distribución gráfica de las variables difusas de entrada y salida, se muestran en las Fig. 26 – 28, respectivamente.

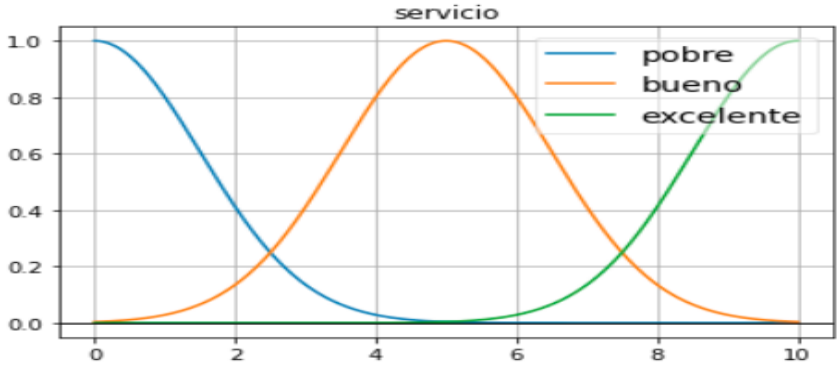


Fig. 26. Distribución gráfica de la variable difusa "servicio" y sus conjuntos difusos.

Tabla 3. Variable difusa de entrada "comida" y la definición de su conjunto difuso.

Variable Difusa de Entrada	Términos Lingüísticos	Función de Membresía y sus parámetros (a,b,c,d)
Comida	Rancio	Trapezoidal ([0,0,1,3])
	Delicioso	Trapezoidal ([7,9,10,10])

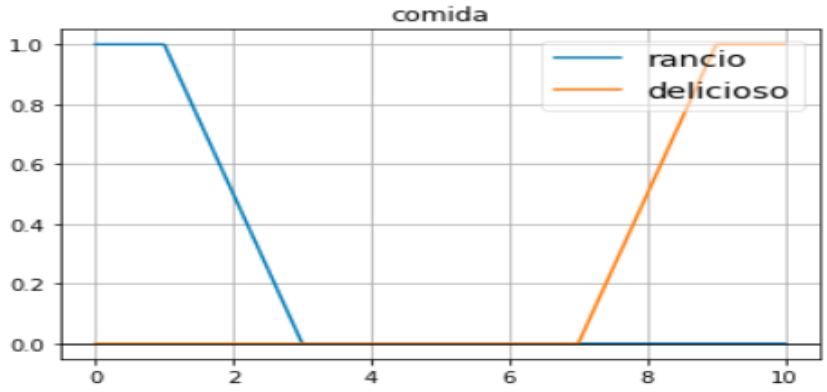


Fig. 27. Distribución gráfica de la variable difusa "comida" y sus conjuntos difusos.

Tabla 4.Variable difusa de salida "propina" y la definición de su conjunto difuso.

Variable Difusa de Salida	Términos Lingüísticos	Función de Membresía y sus parámetros (a,m,b)
Propina	Poco	Triangular ([0,5,10])
	Promedio	Triangular ([10,15,20])
	Generoso	Triangula ([20,25,30])

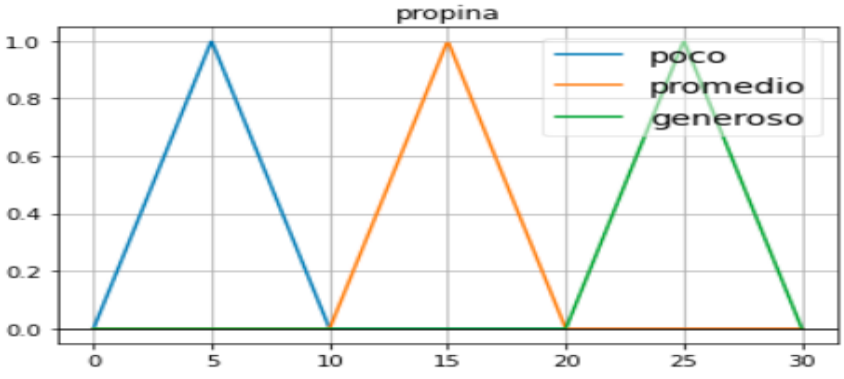


Fig. 28. Distribución gráfica de la variable difusa "propina" y sus conjuntos difusos.

(4.1.2) Creamos las reglas base, de las cuales se puede observar su composición en la tabla 5.

Tabla 5. Elementos que componen las reglas bases.

Número de Regla	Antecedente	Consecuente	Conector
R^1	Servicio is pobre	Propina is poco	OR
	Comida is rancio		
R^2	Servicio is bueno	Propina is promedio	No aplica
R^3	Servicio is excelente	Propina is generoso	OR
	Comida is delicioso		

Las reglas generadas fueron las siguientes:

***R^1: IF** servicio is pobre **or** comida is rancio **THEN** propina is poco*

***R^2: IF** servicio is bueno **THEN** propina is promedio*

***R^3: IF** servicio is excelente **or** comida is delicioso **THEN** propina is generoso*

(4.1.3) Se realiza las fuerzas de disparo sobre las funciones de membresía de las variables difusas de entrada, como se muestra en la Fig. 29.

inputs={'servicio':3,'comida':8}

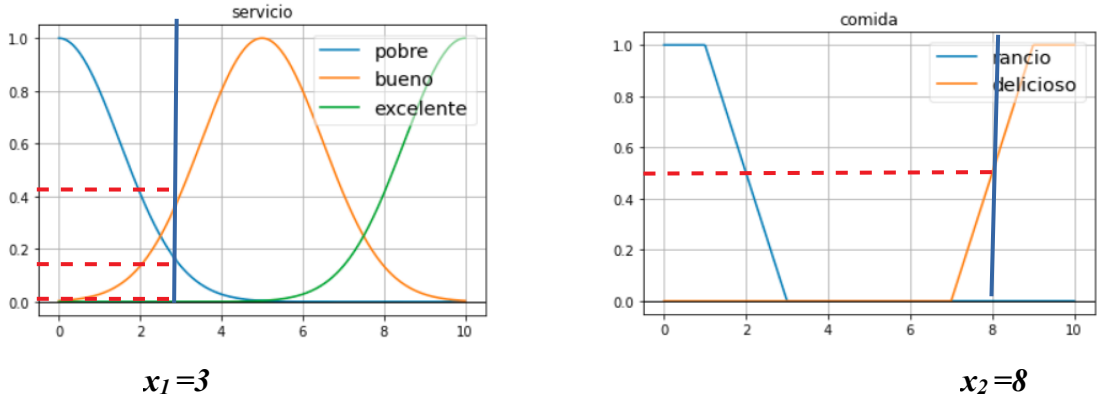


Fig. 29. Fuerza de disparo sobre variables difusas "servicio" y "comida".

(4.1.4) Se realiza la evaluación de las reglas, utilizando las entradas sobre los antecedentes y si se manejan múltiples, entonces se utilizará los conectores AND u OR, para obtener una salida única que represente el resultado de la evaluación.

> ***R¹: IF*** *servicio is pobre* ***(0.135)*** ***or*** *comida is rancio* ***(0)*** ***THEN*** *propina is poco* ***(0.135)***
>
> ***R²: IF*** *servicio is bueno* ***(0.411)*** ***THEN*** *propina is promedio* ***(0.411)***
>
> ***R³: IF*** *servicio is excelente* ***(1.86e-05)*** ***or*** *comida is delicioso* ***(0.5)*** ***THEN*** *propina is generoso* ***(0.5)***

(4.1.5) Los cortes sobre las funciones de membresía consecuentes fueron generados en el proceso de implicación, los cuales se muestran en las Fig. 30.

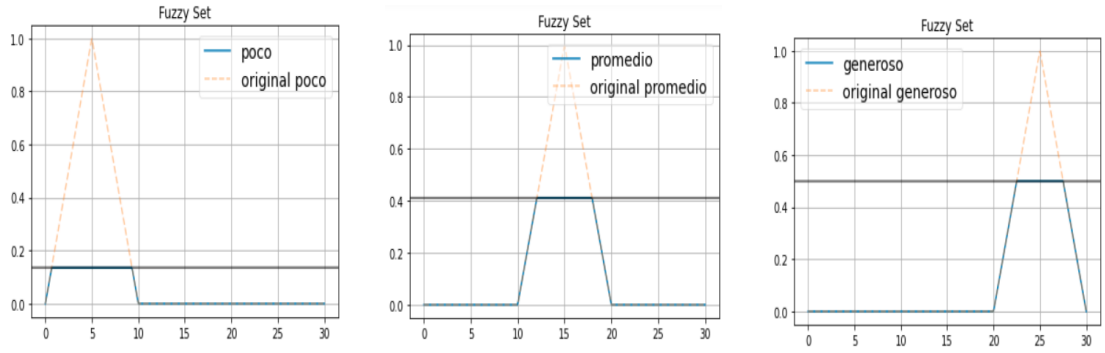


Fig. 30. Cortes sobre las funciones de membresía consecuentes.

(4.1.6) A continuación se realiza el proceso de agregación, el cual se encarga de unificar las salidas obtenidas en el proceso anterior, afín de obtener un único conjunto difuso, como se observa en la Fig. 31.

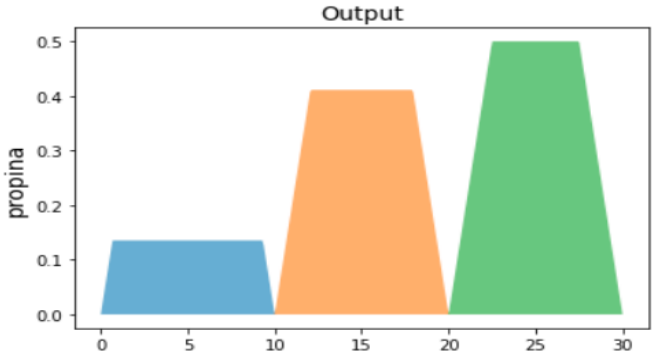


Fig. 31. Conjunto difuso obtenido del proceso de agregación.

(4.2) Modelo de Takagi-Sugeno-Kang (TSK): en contraste con el modelo de inferencia Mamdani, este modelo produce directamente una salida CRISP, y en la parte consecuente de sus reglas utiliza polinomios. La ecuación (10) un sistema de defuzzificación TSK normalizado:

$$y_{TSK}^{N}(\acute{x}) = \frac{\sum_{l=1}^{M} f^{l}(\acute{x}) g^{l}(\acute{x})}{\sum_{l=1}^{M} f^{l}(\acute{x})} \qquad (10)$$

Se puede observar que esta ecuación tiene la misma estructura que la del defuzzificador de centro de conjuntos $y_{cos}(\acute{x})$, lo que lo diferencia es la función consecuente $g^{l}(\acute{x})$. Cuando $g^{l}(\acute{x})$ es solo una constante, entonces $y_{TSK}^{N}(\acute{x})$ es estructuralmente igual a $y_{cos}(\acute{x})$ debido a que $g^{l}(\acute{x})$ reemplazará al centroide y donde $f^{l}(\acute{x})$ es la fuerza de disparo, en caso de que $g^{l}(\acute{x})$ no sea una constante, entonces será una función lineal.

Basándonos en el ejemplo del cálculo del porcentaje de propina que fue creado sobre un sistema de inferencia de Mamdani, lo que cambiaría aquí, es que en lugar de hacer cortes para obtener los nuevos conjuntos difusos en el proceso de implicación y luego agregarlos para generar un conjunto de salida, lo que se hace es obtener los valores constantes y las fuerzas de disparo de cada una de las reglas para poder hacer el cálculo sobre la ecuación (10). Supongamos que los valores constantes para los siguientes términos lingüísticos consecuentes son los siguientes; poco (5%), promedio (15%), generoso (25%), entonces al reemplazar los valores constantes de la función $g^{l}(\acute{x})$y las fuerzas de disparo $f^{l}(\acute{x})$ (que son los mismos calculados en el modelo de inferencia Mamdani (4.1.4)), entonces obtenemos lo siguiente:

$$y_{TSK}^{N}(\acute{x}) = \frac{(5*0.135)+(15*0.411)+(25*0.5)}{0.135+0.411+0.5} = 18.485$$

(5) Defuzzificación: es el proceso de mapear una salida CRISP a partir de los conjuntos difusos combinados (o no) resultado de la agregación en el bloque de inferencia. Dentro de los defuzzicadores más utilizados se encuentran los siguientes:

(5.1) Centroide: combina los conjuntos difusos de salida de tipo-1 de las reglas activadas por las fuerzas de disparo mediante la unión, es decir, se utiliza la t-conorma, generalmente el máximo. Devuelve el centro de gravedad del conjunto difuso a lo largo del eje x. A continuación, se muestra dicho comportamiento en la siguiente ecuación (11).

$$y_{c}(\acute{x}) = \frac{\sum_{i=1}^{N} y_{i} mf_{B}(y_{i}|\acute{x})}{\sum_{i=1}^{N} mf_{B}(y_{i}|\acute{x})} \qquad (11)$$

Donde $mf_i(x_i)$es la función de membresía para el conjunto B, el cual ha sido discretizado en puntos, siendo estos y_1, y_i, y_N y donde $y_c(\acute{x})$ es mostrado como una función de $\acute{x}$ debido a que $mf_b(y_i|\acute{x})$ es una función de entrada difusa del sistema $\acute{x}$, por lo que, para cada $\acute{x}$ un diferente valor es obtenido para y_c, este tipo de defuzzificador representa una alta carga computacional. A continuación, se muestra un ejemplo del defuzzificador por centroide en la Fig. 32.

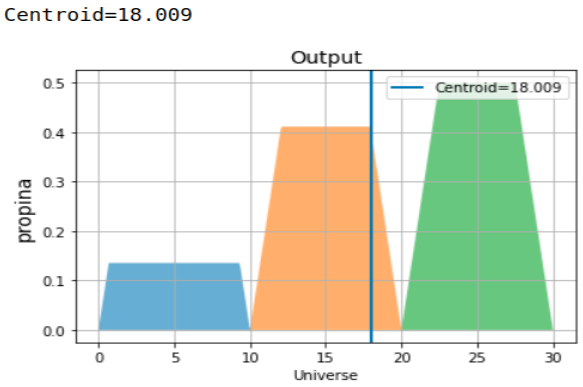


Fig. 32. Defuzzificación por centroide.

(5.2) Alturas: también llamado como **media de centros**, el cual reemplaza el conjunto de salidas de las reglas activadas, de cada regla activada por un valor singleton en el punto que tenga la membresía máxima en el conjunto difuso consecuente de la regla, con la amplitud igual a la función de membresía de la salida de la regla activada en ese punto, y luego calcula el centroide del conjunto de tipo-1 compuesto por estos singletons. A continuación, se muestra dicho comportamiento en la siguiente ecuación (12).

$$y_h(\acute{x}) = \frac{\sum_{l=1}^{M} y^{-l} mf_{B^l}(y^{-l}|\acute{x})}{\sum_{l=1}^{M} mf_{B^l}(y^{-l}|\acute{x})} \qquad (12)$$

Donde y^{-l} es el punto que tiene máxima membresía en el *i-ésimo* conjunto difuso consecuente, si hay más de un punto su promedio se puede tomar como y^{-l} y su grado de membresía en el *i-ésimo* conjunto de reglas activadas es $mf_{B^l}(y^{-l}|\acute{x})$, facilita mucho su computo, pero presenta un problema cuando su máximo valor en el conjunto consecuente es $y^{-l} = 0$. A continuación, se muestra un ejemplo del defuzzificador por alturas en la Fig. 33.

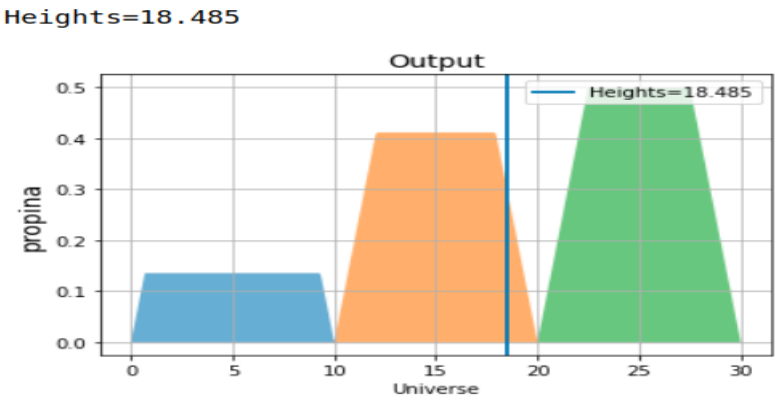


Fig. 33. Defuzzificación por alturas.

(5.2) Centro de conjuntos: cada conjunto consecuente de la regla es reemplazado por un singleton localizado en sus centroides, con amplitud igual a la fuerza de disparo, después del cual el centroide de estos singletons es encontrado. A continuación, se muestra dicho comportamiento en la siguiente ecuación (13).

$$y_{cos}(\acute{x}) = \frac{\sum_{l=1}^{M} COG(G^l) f^l(\acute{x})}{\sum_{l=1}^{M} f^l(\acute{x})} = \frac{\sum_{l=1}^{M} c^l f^l(\acute{x})}{\sum_{i=1}^{M} f^l(\acute{x})} \qquad (13)$$

Donde c^l es el centroide del *i-ésimo* conjunto consecuente y $f^l(\acute{x})$ es la fuerza de disparo, si cada consecuente es simétrico, normal y convexo entonces $c^l = y^{-l}$, pero para no simétricos consecuentes $c^l \neq y^{-l}$. A continuación, se muestra un ejemplo del defuzzificador por centro de conjuntos en la Fig. 34.

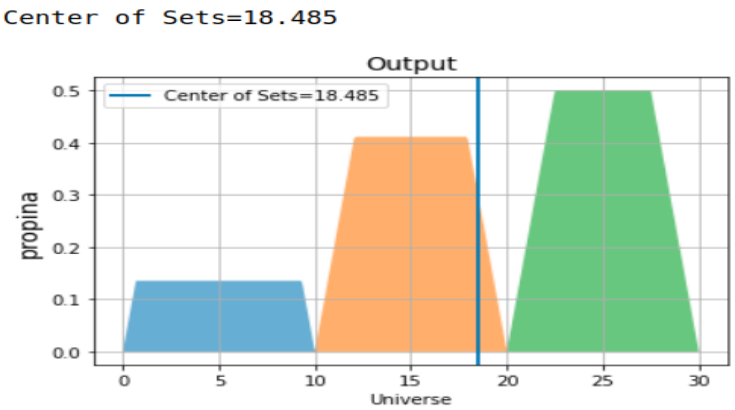


Fig. 34. Defuzzificación por centro de conjuntos.

(5.3) Media de la máxima: regresa el promedio de los valores de la variable base en los que sus valores de membresía alcanzan el máximo. A continuación, se muestra dicho comportamiento en la siguiente ecuación (14).

$$\frac{\sum_{j=1}^{k} x_j}{k} \tag{14}$$

Donde k es el número de elementos discretos del conjunto de salida difuso que alcanzan la membresía máxima. A continuación, se muestra un ejemplo del defuzzificador por media de la máxima en la Fig. 35.

$$x^* = \frac{(a+b)}{2}$$

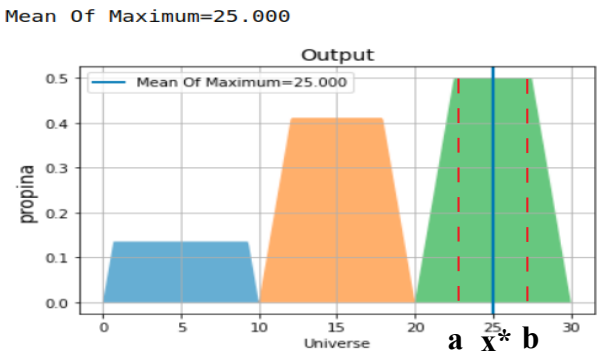


Fig. 35. Defuzzificación por media de máxima.

(5.4) Primera de la máxima: regresa el valor más pequeño de x que pertenece a $[a, b]$ en el cual sus valores de membresía alcanzan el máximo. A continuación, se muestra un ejemplo del defuzzificador por primera de la máxima en la Fig. 36.

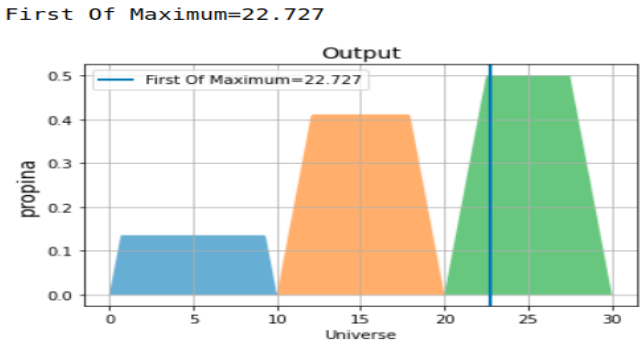


Fig. 36. Defuzzificación por primera de la máxima.

(5.5) Último de la máxima: regresa el valor más grande de xque pertenece a [a, b]en el cual sus valores de membresía alcanzan el máximo. A continuación, se muestra un ejemplo del defuzzificador por último de la máxima en la Fig. 37.

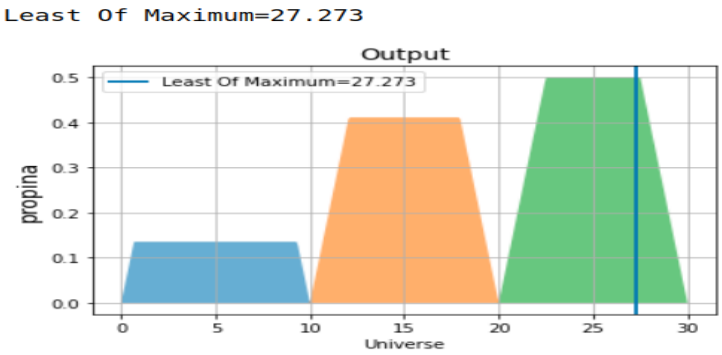


Fig. 37. Defuzzificación por último de la máxima.

En el contexto de Big Data, se tiene la capacidad de manejar varios tipos de incertidumbre, mismos que están presentes en cada fase de su procesamiento. También las técnicas de lógica difusa junto con técnicas de cómputo granular pueden ser empleadas reconstruir problemas con cierto nivel de granularidad. Pueden ser mucho más eficientes aún si se asocian con otras técnicas para la toma de decisiones, como lo son; probabilidad, conjuntos aproximados, redes neuronales, entre otras [77]. Algunas de las aplicaciones de los sistemas difusos han sido dadas en; sistemas de control, sistema de frenado del vehículo, control de elevador, electrodomésticos, control de señal de tráfico, y así sucesivamente [22].

Su relevancia en el entorno de Big Data radica en su capacidad para proporcionar una mejor representación del problema mediante el uso de variables lingüísticas, las cuales facilitan el manejo de volumen y variedad cuando los conjuntos de los datos crecen de forma dinámica y exponencial [24].

Otras aplicaciones se encuentran en los sistemas híbridos inteligentes, donde se combinan los beneficios de los sistemas difusos y las redes neuronales, lo que mejora la capacidad de estos últimos para descubrir a través del aprendizaje de los parámetros necesarios para procesar los datos. También se ha propuesto integrar a estos sistemas inteligentes híbridos, algoritmos genéticos para optimizar los parámetros, ajustar los puntos de control de las funciones de membresía y ajustar sus pesos difusos [10].

CAPÍTULO 3. Sistema Neuro-Difuso basado en Mamdani con Defusificación en Centro de Conjuntos.

3.1 Descripción del sistema propuesto

El sistema propuesto es un sistema hibrido basado en redes neuronales que da soporte al ajuste de los parámetros de diseño utilizando el método de gradiente descendente con momentum y tasa de aprendizaje adaptiva para agilizar dicho ajuste. Se combinó con un sistema difuso basado en Mamdani con defusificación en conjunto de centros, con la finalidad de brindar interpretabilidad al experto, una vez que el modelo pseudo-óptimo ha sido encontrado.

La arquitectura general del sistema propuesto está compuesta por cuatro capas, como se muestra en la Fig. 38, a continuación, detallamos sus características:

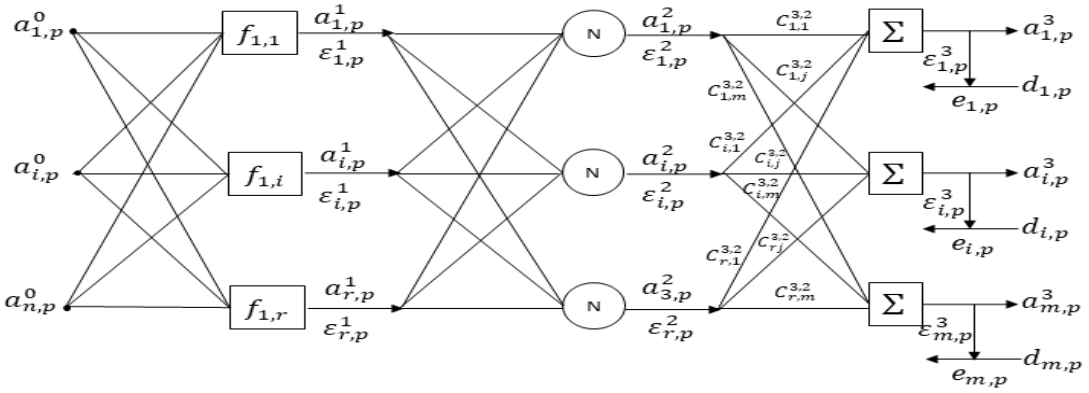


Fig. 38. Arquitectura general del sistema neuro-difuso propuesto basado en Mamdani con defusificación en centro de conjuntos.

- **Capa 0:** en esta capa, se toman las entradas y a partir de ella se genera el ***Sistema de Inferencia Difusa*** (Fuzzy Inference System) ***inicial***, siendo los componentes más importantes de su metadato*; las variables difusas de entrada y salida, número de reglas, universo de discurso (dominio), funciones de membresía y método de defusificación*. La regla genérica basada en Mamdani conocida como base de conocimiento, fue definida de la siguiente manera (15):

$$\boldsymbol{R^k}: \boldsymbol{IF\ x_1\ is\ F_1^k\ \wedge\ x_i\ is\ F_i^k\ \wedge\ x_n\ is\ F_n^k\ THEN\ y_1 is\ C_{G_1^k}, y_i is\ C_{G_i^k}, y_m is\ C_{G_m^k}} \qquad (15)$$

Donde R^k corresponde a la $k-ésima$ regla, la parte antecedente está definida por sus entradas y es representada por $\boldsymbol{x_i^n} \forall\, \boldsymbol{i = 1, \dots, n}$, la definición de sus fuerzas de disparo representadas por $\boldsymbol{F_i^k} \forall\, \boldsymbol{i = 1, \dots, k}$, la parte consecuente definida por sus salidas $\boldsymbol{C_{G_m^k}} \forall\, \boldsymbol{m = 1, \dots, k}$.

Las entradas por su parte han sido definidas como una matriz $\boldsymbol{n\ x\ p}$, donde $\boldsymbol{n}$ corresponde al número de atributos y $\boldsymbol{p}$ al número de instancias por cada atributo en la matriz, como se muestra en la siguiente expresión (16):

$$\boldsymbol{a^0 = \left[a_{i_0,p_q}^n\right] \forall\, i_0 = 1, \dots, n\ ;\ p = 1, \dots, q} \tag{16}$$

- **Capa 1:** en la *retroalimentación hacia adelante (feedforward)* de la red neuronal, se calculan las fuerzas de disparo a partir de las entradas $\boldsymbol{a_{i_0,p}^0}$ sobre una *función de membresía gaussiana (17)* definida en la parte antecedente de las reglas. Esta capa contará con nodos adaptivos (uno por cada regla definida), en el cual durante el proceso iterativo de *retro-propagación (backpropagation)* irá ajustando los parámetros de diseño, los cuales son: $\boldsymbol{m_{i_1,i_0}}$ (media) y $\boldsymbol{\sigma_{i_1,i_0}}$ (desviación estándar).

$$\boldsymbol{a_{i_1,p}^1 = \exp\left[\frac{-1}{2}\left\{\sum_{i_0=1}^{n}\left(\frac{a_{i_0,p}^0 - m_{i_1,i_0}}{\sigma_{i_1,i_0}}\right)^2\right\}\right]} \tag{17}$$

- **Capa 2:** toma las fuerzas de disparo calculadas en la capa anterior y las normaliza a una escala estándar con la finalidad de ayudar a agilizar el aprendizaje de la red neuronal, como se muestra en la siguiente expresión (18).

$$\boldsymbol{a_{i_2,p}^2 = \frac{a_{i_1,p}^1}{\sum_{i_1=1}^{r} a_{i_1,p}^1} \forall\, i_1 = 1, \dots, r\ ; p = 1, \dots, q} \tag{18}$$

Donde $\boldsymbol{a_{i_1,p}^1}$ es la señal de salida o fuerza de disparo obtenida en la $\boldsymbol{capa\ 1}$.

- **Capa 3:** finalmente, en esta capa, el producto de los centroides $\boldsymbol{C_{i_3,i_2}}$ (parámetro de diseño) y las fuerzas de disparo calculadas en la capa anterior $\boldsymbol{a_{i_2,p}^2}$ generará las fuerzas de disparo de esta última capa $\boldsymbol{a_{i_3,p}^3}$, dicho comportamiento se representa bajo la siguiente expresión (19):

$$\boldsymbol{a_{i_3,p}^3 = \sum_{i_3,p}^{m} C_{i_3,i_2}\, a_{i_2,p}^2 \ \forall\, i_2 = 1, \dots, r\ ;\ i_3 = 1, \dots, m} \tag{19}$$

A la completa trayectoria que va de la $\boldsymbol{capa\ 0}$ a la $\boldsymbol{capa\ N}$ se le conoce como ***retroalimentación hacia adelante (feedforward)*** y al proceso que calcula las derivadas parciales del error con respecto a todos los parámetros de diseño y su posterior actualización, se le conoce como ***retro-propagación (backpropagation)***, el cual tiene una trayectoria inversa a la *retroalimentación hacia adelante*, dicha combinación de métodos se realiza de forma iterativa, hasta que la función de error ha llegado a un mínimo pseudo-óptimo o se han activado algunos otros criterios de validación para detener el entrenamiento. A continuación, se muestra el procedimiento para el cálculo del error:

- **Cálculo de Error:** se genera el cálculo de error $\boldsymbol{e}$, el cual es obtenido a partir de la diferencia de la salida deseada $\boldsymbol{T}$ y la señal de salida estimada $\boldsymbol{a^3_{i_3,p}}$, como se muestra en la siguiente expresión (20):

$$\boldsymbol{e = T - \ a^3_{i_3,p}} \tag{20}$$

Posteriormente, el *error calculado se eleva al cuadrado* $\boldsymbol{e^2}$ por cada dato (21).

$$\boldsymbol{E_p = \ \sum_{i_3=1}^{m} \left(e_{i_3,p}\right)^2 \ \forall \ i_3 = 1, \dots, m} \tag{21}$$

Con la finalidad de obtener un error total $\boldsymbol{E}$, se realiza la suma de los errores al cuadrado $\boldsymbol{E_p}$ (22).

$$\boldsymbol{E = SSE = \ \sum_{p=1}^{q} E_p \ \ \forall \ \ p = 1 \dots, q} \tag{22}$$

Una vez calculado el error total $\boldsymbol{E}$ la rutina de la *retroalimentación hacia adelante (feedforward)* ha concluido, por lo que iniciamos con la *retro-propagación (backpropagation)* este es el procedimiento para construir un vector gradiente, como se muestra a continuación:

- **Capa 3**: se calcula el error $\boldsymbol{\varepsilon^3_{i_3,p}}$ (23), a través de la derivada de la medida de error $\boldsymbol{E_p}$**.**

$$\boldsymbol{\varepsilon^3_{i_3,p} = \ -2 e_{i_3,p} \ \forall \ i_3 = 1, \dots, m \ ; p = 1 \dots, q} \tag{23}$$

En esta misma capa, se calcula la derivada parcial del error $\boldsymbol{\varepsilon_{i_3,p}^3}$ con respecto al centroide $\boldsymbol{C_{i_3,p}}$ y a la salida de la $\boldsymbol{capa\ 2}\ \boldsymbol{a_{i_2,p}^2}$ (24).

$$\frac{\partial^{+E}}{\partial C_{i_3,i_2}} = \sum_{p=1}^{q} \frac{\partial^{+E}}{\partial C_{i_3,i_2}} = \boldsymbol{\varepsilon_{i_3,p}^3} \cdot \boldsymbol{a_{i_2,p}^2} \tag{24}$$

$$\forall\ i_2 = 1, \dots, r\ ;\ i_3 = 1, \dots, m\ ; p = 1 \dots, q$$

- **Capa 2:** a partir del producto de la derivada parcial del error calculado en la capa anterior $\boldsymbol{\varepsilon_{i_3,p}^3}$ y los centroides $\boldsymbol{C_{i_2,i_3}}$, obtenemos el error para esta capa (25).

$$\boldsymbol{\varepsilon_{i_2,p}^2} = \sum_{i_3=1}^{m} \boldsymbol{C_{i_2,i_3}}\ \boldsymbol{\varepsilon_{i_3,p}^3} \tag{25}$$

$$\forall\ i_2 = 1, \dots, r\ ;\ i_3 = 1, \dots, m\ ; p = 1 \dots, q$$

- **Capa 1:** el error en esta capa $\boldsymbol{\varepsilon_{i_1,p}^1}$ se calcula a partir de la diferencia del error de la capa anterior $\boldsymbol{\varepsilon_{i_1,p}^2}$ y el producto del error de la $capa\ 2$ y su señal de salida $\boldsymbol{a_{i_2,p}^2}$ entre la sumatoria de la señal de salida de la capa actual $\boldsymbol{a_{i_1,p}^1}$ (26).

$$\boldsymbol{\varepsilon_{i_1,p}^1} = \frac{\boldsymbol{\varepsilon_{i_1,p}^2} - \sum_{i_2=2}^{r} \boldsymbol{\varepsilon_{i_2,p}^2}\ \boldsymbol{a_{i_2,p}^2}}{\sum_{i_1=1}^{r} \boldsymbol{a_{i_1,p}^1}} \tag{26}$$

$$\forall\ i_1 = 1, \dots, r\ ;\ i_2 = 1, \dots, r\ ; p = 1 \dots, q$$

También en esta capa se calculan las derivadas de los parámetros; $media\ \boldsymbol{\partial m_{i_1,i_0}}$ (27) y $\boldsymbol{desviación\ estándar}\ \boldsymbol{\partial \sigma_{i_1,i_0}}$ (28), mismas que servirán para construir el vector gradiente, para finalmente actualizar los parámetros de diseño con el nuevo vector direccional.

$$\frac{\partial^{+E}}{\partial m_{i_1,i_0}} = \sum_{p=1}^{q} \frac{\partial^{+E_p}}{\partial m_{i_1,i_0}} = \boldsymbol{\varepsilon_{i_1,p}^1} \cdot \boldsymbol{a_{i_1,p}^1}\ \frac{\boldsymbol{a_{i_0,p}^0} - \boldsymbol{m_{i_1,i_0}}}{\boldsymbol{\sigma_{i_1,i_0}^2}} \tag{27}$$

$$\frac{\partial^{+E}}{\partial \sigma_{i_1,i_0}} = \sum_{p=1}^{q} \frac{\partial^{+E_p}}{\partial \sigma_{i_1,i_0}} = \boldsymbol{\varepsilon_{i_1,p}^1} \cdot \boldsymbol{a_{i_1,p}^1}\ \frac{\left(\boldsymbol{a_{i_0,p}^0} - \boldsymbol{m_{i_1,i_0}}\right)^2}{\boldsymbol{\sigma_{i_1,i_0}^3}} \tag{28}$$

$$\forall\ i_0 = 1, \dots, n\ ;\ i_1 = 1, \dots, r\ ; p = 1 \dots, q$$

Se construye el vector gradiente y se aplica la tasa de aprendizaje esto permitirá generar el cambio direccional con el cual actualizaremos o ajustaremos todos los parámetros de diseño (29).

$$\nabla\xi = -\eta\,\nabla E(\xi) = -\eta\,\frac{\partial E}{\partial \xi} \tag{29}$$

Donde $\xi \in \{\sigma, m, C\}$ es el vector de todos los parámetros de diseño que corresponden a la *función gaussiana* definida en la parte antecedente de las reglas difusas y al *centroide* en la parte consecuente, por otro lado, $-\eta$ corresponde al *cambio en la tasa de aprendizaje*, la cual forma parte de los parámetros de entrenamiento.

3.2 Hiperparámetros y funcionalidad.

Los hiperparámetros son aquellos que deben ser configurados a priori a la ejecución del entrenamiento, aunque existen algunas recomendaciones para elegir los valores iniciales apropiados, no hay nada concluyente, como es el caso de la tasa de aprendizaje, ya que como se sabe, si se establece con un valor muy bajo y si además el dataset a analizar es complejo, el entrenamiento se conducirá de forma lenta y tendrá por ello una gran carga computacional, ya que requerirá de muchísimas iteraciones para poder aproximarse a una solución cuasi-óptima o pudiera nunca llegar a ella. Ahora, si se establece un valor muy grande, es muy probable que tienda a divergir del área de solución.

Una solución que hemos utilizado para hacer frente a esta problemática es realizar un entrenamiento previo, donde buscamos en un rango de valores para el hiperparámetro de tasa de aprendizaje principalmente, aquel valor que brindan mejores resultados con respecto a la minimización de la función de error establecida como es la suma de los errores al cuadrado.

Los hiperparámetros de entrenamiento requeridos para el sistema neuro-difuso propuesto, son definidos para controlar la duración del entrenamiento, a continuación, se describen cada uno de ellos:

- **Número de épocas totales:** es el límite máximo de iteraciones que serán llevadas a cabo durante el entrenamiento.
- **Error objetivo:** el valor mínimo del error ideal que pudiera ser alcanzado.
- **Tasa de aprendizaje (η):** parámetro que permite controlar la velocidad del descenso del gradiente, avanzando hacia la obtención de los parámetros pseudo-óptimos.
- **Momentum (mc):** fracción del cambio en el parámetro que permite suavizar la oscilación en la trayectoria, tanto incrementando o decrementando los cambios de los parámetros en cada una de las iteraciones.
- **Número máximo de fallas en la validación:** es el límite máximo de fallas permitidas en el proceso de validación.

- **Incremento máximo de error:** se refiere al límite máximo permitido del error calculado.
- **Límite mínimo del gradiente:** es el valor mínimo de la norma del vector gradiente calculado.
- **Tasa de decremento:** este valor permite disminuir la tasa de aprendizaje según la proporción definida, en caso de que el error calculado sea mayor al incremento máximo establecido.
- **Tasa de incremento:** este valor permite incrementar la tasa de aprendizaje según la proporción definida, en caso de que el error calculado sea menor al calculado en la iteración anterior.
- **Número de reglas:** indica el número de reglas con la que contará la parte difusa y también representa el número de neuronas por cada capa en la red neuronal.

El ***Sistema de Inferencia Difusa*** en la definición de sus *antecedentes* y *consecuentes*, tienen ***funciones de membresía*** (*media y desviación* estándar como parámetros dentro de la *función gaussiana*) y ***centroides*** respectivamente, dichos parámetros deben ser inicializados previos al entrenamiento y ajustados durante el mismo, a estos parámetros se les conoce como ***parámetros de diseño***.

Se identifican los procesos funcionales principales, los cuales llevan a cabo el proceso de entrenamiento y aprendizaje en el *sistema neuro-difuso propuesto*, estos son:

- **Inicialización:** en este primer proceso, los parámetros de diseño son calculados como se representa en la ecuación (30), ya que este será requerido durante el entrenamiento en la *función de optimización Gradiente Descendente con Momentum y Tasa de Aprendizaje Adaptiva*.

$$\nabla \xi_{prev} = -\eta \frac{\partial E}{\partial \xi} \tag{30}$$

- **Entrenamiento:** se inicia con el cálculo de la *función de optimización* (31).

$$\nabla \xi_{now} = mc\, \nabla \xi_{prev} - (1 - mc)\, \eta g(\xi) \tag{31}$$

Es en este proceso donde un ***método de aprendizaje (lotes, en línea y mini-lotes)*** es establecido, ya que este definirá la forma en la cual se enviarán los datos a ser procesados para generar el vector gradiente y realizar el cálculo del error. Con la finalidad de controlar la velocidad y dirección del descenso del gradiente, la siguiente heurística fue implementada, como se muestra en el siguiente pseudo-código:

1. ***Mientras*** *el número de épocas no alcance su máximo definido*
2. *Se calcula el Gradiente Descendente con Momentum y Tasa de Aprendizaje Adaptiva*
3. $$\nabla\xi_{now} = mc\,\nabla\xi_{prev} - (1 - mc)\,\eta g(\xi)$$
4. *Se realiza la actualización de los parámetros de diseño y se almacena de manera temporal.*
5. $$\xi^{temporal} = \xi^{old} + \nabla\xi_{now}$$
6. *El error actual se calcula con el cambio aplicado a los parámetros.*
7. ***Si*** *(error actual / error previo) es* ***mayor*** *que el parámetro de* ***incremento máximo de error Entonces***
8. ***La tasa de aprendizaje es decrementada*** $\eta = \eta * tasa\ de\ decremento$
9. *El vector gradiente es actualizado aplicando la nueva tasa de aprendizaje*
10. $$g(\xi) = \eta * g(\xi)$$
11. ***Sino***
12. ***Si*** *error actual es* ***menor*** *que el error previo* ***entonces***
13. ***La tasa de aprendizaje es incrementada***
14. $$\eta = \eta * tasa\ de\ incremento$$
15. ***Fin***
16. *Se actualizan los parámetros con el cambio calculado en la línea 4, pero ahora de forma permanente*
17. $$\xi^{new} = \xi^{old} + \nabla\xi_{now}$$
18. *El error previo es reemplazado por el error actual*
19. *Se recalcula el* ***vector gradiente*** *con los nuevos parámetros*
20. ***Fin***
21. ***Fin***

Los siguientes criterios fueron considerados como soporte para la decisión del momento en el cual el entrenamiento deberá ser interrumpido, estos son:

- Cuando el *número de épocas procesadas* ***es igual*** al parámetro de entrenamiento definido como *número de épocas totales*.
- Cuando el *error calculado* ***es menor o igual*** al parámetro de entrenamiento definido como *error objetivo*.
- Cuando la *norma del vector gradiente* ***es menor que*** el parámetro de entrenamiento definido como *límite mínimo del gradiente.*
- Cuando el *número de validaciones fallidas acumuladas* ***es mayor*** al parámetro de entrenamiento definido como *número máximo de fallas en validación.*
- Finalmente, la métrica de validación que permite verificar el avance del entrenamiento en la dirección de la convergencia óptima y la cual se busca minimizar es la ***suma de los errores al cuadrado***.

3.3 Plataforma para el procesamiento paralelizable y distribuido

Cuando se trabaja con volúmenes de datos masivos o los bien conocidos Big Data, se hace necesario contar con una infraestructura que pueda dar soporte a la ejecución de algoritmos que procesarán y analizarán esta ingente cantidad de datos, mismas que se ven imposibilitas de llevar a cabo sobre tecnologías de corte tradicional como lo son; gestores de bases de datos relacionales con un poder de cómputo limitado.

Es por ello, el gran auge que han tenido tecnologías de código abierto para el procesamiento de Big Data, potenciando las capacidades del cómputo en la nube, la cual permite contar con clúster de cómputo sobre los cuales se pueden paralelizar las tareas que más consumen recursos y distribuir la carga de trabajo sobre todas las unidades de procesamiento en el clúster.

Actualmente uno de los motores de análisis unificado para el procesamiento de datos a gran escala es Apache Spark, considerado como el proyecto de código abierto más grande en procesamiento de datos. Es de alto desempeño, ejecutando cargas de trabajo 100 veces más rápido que otras tecnologías, tanto para lotes datos como en flujo de transmisión de datos en tiempo real.

Algunas de sus características más relevantes son; tolerante a fallos a través de su colección de elementos que además son capaces de operarse de forma paralela, cómputo en memoria (lo que permite mayor velocidad de procesamiento, al no tener que estar leyendo y escribiendo en disco), soporta múltiples lenguajes (Java, Scala y Python), reusabilidad, analítica avanzada, integración con Hadoop, evaluaciones perezosas (las transformaciones sobre los conjuntos de datos tolerante a fallos no se resuelve, sino que se almacenan en un grafo acíclico dirigido), entre otras.

El ecosistema de Apache Spark ha sido diseñado en dos capas principales, donde la capa inferior corresponde al ***núcleo (core) de Spark*** el cual contiene interfaces de programación de aplicaciones en múltiples lenguajes como lo son; *Java, Python, Scala y R*, así como el ***motor de procesamiento*** de datos a gran escala distribuidos en un clúster, el cual es el encargado de particionar en pequeñas tareas los datos para que posteriormente puedan trabajarse de manera paralela, monitorear, administrar dichas tareas y proporcionar tolerancia a fallos, es el responsable además de interactuar con el administrador del clúster y del sistema de almacenamiento

Todo lo anterior corresponde al marco de trabajo para procesamiento de datos, por lo que, para llevar a cabo un buen control de los recursos del clúster, se ha agregado a esta capa un administrador del clúster, siendo los más utilizados; *YARN, Kurbenetes, Mesos, Standalone, entre otros*. Apache Spark tampoco cuenta con un sistema de almacenamiento integrado, por el contrario, te permite procesar tus datos los cuales pueden estar almacenados en una gran variedad de sistemas de almacenamiento, siendo los más utilizados; *HDFS (Hadoop Distributed File System), S3 Amazon (Simple Cloud Storage), Azure Blob, GCS (Google Cloud Storage) y CFS (Cassandra File System)*.

Por otro lado, la capa superior corresponde a los lenguajes de dominio específicos, librerías e interfaces de programación de aplicaciones que han sido desarrolladas por la comunidad de Spark, las cuales soportan diferentes requerimientos de procesamientos, dichos requerimientos han sido agrupadas en; Spark SQL (Dataframes), Streaming (procesamiento continuo y flujo de datos ilimitados), MLlib (aprendizaje máquina) y GraphX (cómputo de grafos).

La siguiente figura es la representación del ecosistema que acabamos de explicar Fig. 39.

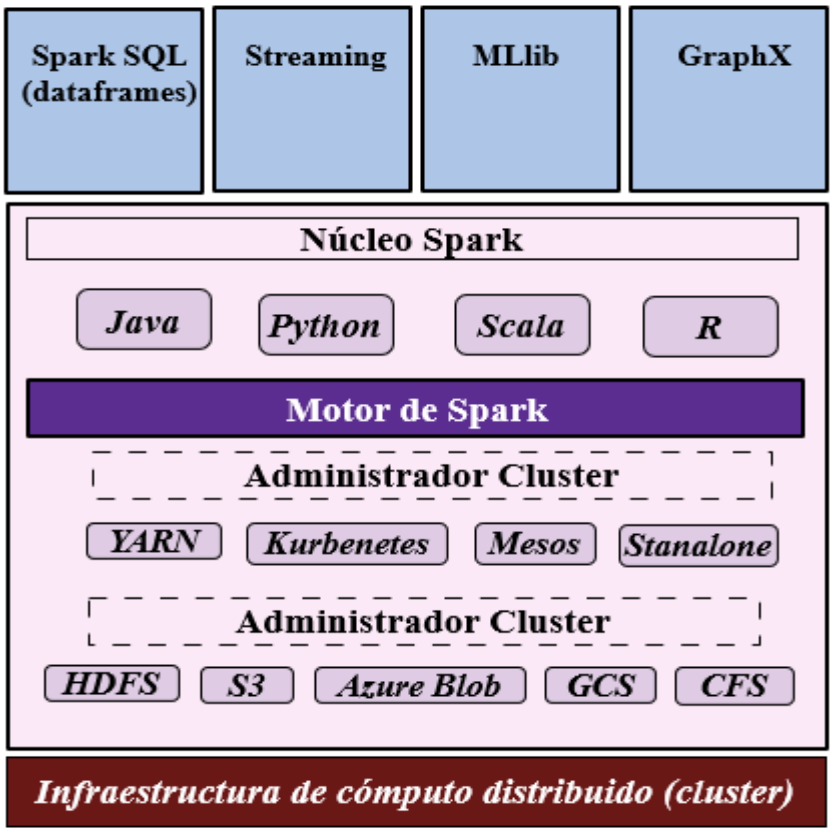


Fig. 39. Esquema de representación de ecosistema de Apache Spark.

El éxito del uso de la plataforma de procesamiento a gran escala y cómputo distribuido de Apache Spark radica en que hace totalmente transparente la forma en la que las tareas y los datos son paralelizados y distribuidos en el clúster, el desarrollador solo deberá preocuparse de codificar su aplicación o algoritmo y de lo demás se encargará el núcleo de Apache Spark. Otras de sus bondades es que es una plataforma unificada que te permite manejar tanto datos estructurados como semi-estructurados, procesamiento en lotes, datos de flujo continuo, procesamiento gráfico, aprendizaje máquina, aprendizaje profundo y consultas SQL, todo en el mismo marco de trabajo, con una variedad de lenguajes de programación para realizar el desarrollo de soluciones que mejor se adapten a nuestras necesidades.

3.3.1 Arquitectura de modelo de procesamiento distribuido en Apache Spark

Apache Spark tiene una arquitectura basada en el modelo ***maestro-esclavo*** sobre la cual se procesa la aplicación y datos enviados del ***cliente al clúster*****.** En la terminología de Spark el ***maestro*** es considerado un ***conductor (driver)*** y los ***esclavos*** son llamados ***ejecutores (executors)****,* estos últimos son asignados a ***nodos trabajadores (workers)*** en el clúster. Debido a que los términos en ingles son los más utilizados dentro del argot de la comunidad de Spark, los estaremos empleando de aquí en adelante.

Cuando un cliente solicita enviar la aplicación al clúster se crea un proceso ***driver***, este solicitará recursos al ***administrador del clúster*** en ***términos de memoria y unidades de procesamiento*** dedicadas a los ***nodos workers***, también será el encargado de convertir el código de la aplicación en un ***gráfico acíclico dirigido lógicamente*** llamado ***DAG*** (*por sus siglas en inglés Directed Acyclic Graph*) el cual es un ***plan de ejecución*** que está compuesto por ***etapas (stages)*** sobre las cuales se crean ***múltiples unidades de ejecución física*** llamadas ***tareas (task)***, que serán agrupados dentro de los ***executors*****.** A continuación, se muestra un esquema donde se representa el comportamiento antes mencionado en la Fig. 40.

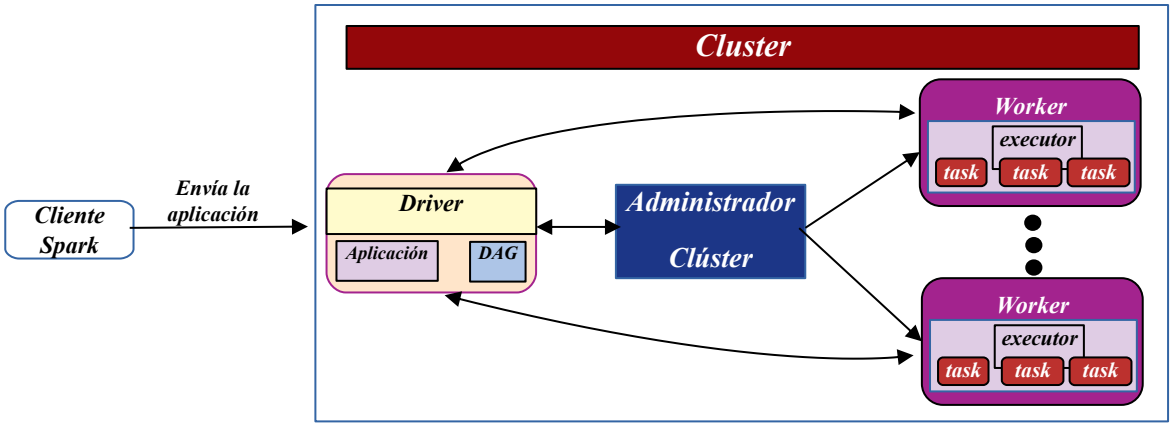


Fig. 40. Representación del Modelo Maestro-Esclavo en el contexto de Apache Spark.

Como se mencionó anteriormente, *Apache Spark* proporciona objetos que permitirán encapsular las tareas y datos particionados que deseamos paralelizar y distribuir sobre el clúster, estos son los conocidos ***conjuntos de datos distribuidos resilientes*** *o también llamados como* ***RDD*** *(por sus siglas en ingles Resilient Distributed Dataset)*.

Estos objetos son ***resilientes o tolerante a fallos***, debido a que por cada ***definición de operación*** que se establece sobre los datos se crea un nuevo RDD, de tal forma que se va creando un ***linaje*** y si en algún punto se presentara un fallo, podemos recuperar el último procesamiento bueno generado. A las *definiciones de operaciones establecidas y almacenadas* en los RDD se les conoce como ***transformaciones***, las cuales son ***evaluaciones perezosas*** lo que significa que la ejecución de la transformación configurada no ocurrirá hasta que una ***acción*** sea invocada.

Las ***acciones*** ejecutarán lo que se haya configurado en cada ***transformación*** de forma paralela y distribuida, y posteriormente ***recolectará todas las particiones procesadas*** *y las enviará de regreso al* ***driver***, todo este puede observarse en la Fig. 41.

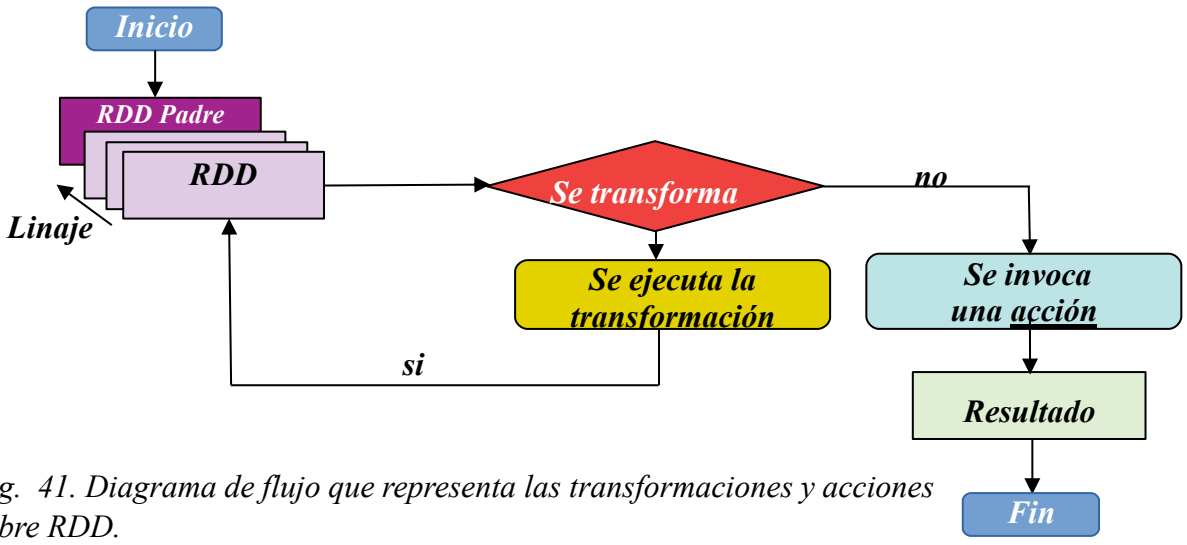


Fig. 41. Diagrama de flujo que representa las transformaciones y acciones sobre RDD.

3.3.2. Adaptación del sistema neuro-difuso para el funcionamiento en la plataforma Spark.

Para poder llevar a cabo la experimentación de Big Data sobre el *sistema neuro-difuso propuesto* mostrado en la Fig. 38, se requirió el ***desarrollo de un módulo de entrenamiento*** que brindara el soporte para ejecutar sus tareas de forma paralela.

Estas tareas internamente trabajan con una red neuronal la cual es la encargada de recibir los mini-batch de entrenamiento y una instancia FIS (Fuzzy Inference System) la cual encapsula entre otras cosas los parámetros de diseño que se desean optimizar dentro de la red neuronal.

El módulo de entrenamiento fue integrado al ***framework Fuzzy System versión 0.242*** [48]*,* ya que proporciona bibliotecas que facilitan la construcción del *FIS inicial*, de forma automática, a partir de; ***datos de entrada, número de reglas difusas, tipo de FIS, método de defusificación y métodos para su posterior evaluación (estabilidad y desempeño).***

Tanto el módulo de entrenamiento como el framework Fuzzy System-0.242 *fueron desarrollado en lenguaje Python versión 3.8.5, y para dar soporte en la paralelización de sus tareas se trabajó con Apache Spark utilizando la biblioteca de Pyspark 3.0.1 el cual ofrece un API de Python para Spark.*

Por cuestión de eficiencia en el cómputo de los datos a procesar durante el entrenamiento, el método de defuzzificación que se utilizó fue el de ***Sugeno de orden cero,*** ya que este es equivalente a un ***Mamdani con defuzzificación con centro de conjuntos***. La adaptación que se realizó a la arquitectura del sistema neuro-difuso propuesto, se explica a continuación:

1. Debido a que el proceso de paralelización requiere que el *drive* distribuya las *tareas* sobre los *ejecutores* del *clúster*, es necesario encapsular tanto las muestras de entrenamiento, validación y prueba, parámetros de entrenamiento, mini-batch y parámetros de diseño en instancias u objetos, ya que si lo queremos trabajar con variables locales, al momento de generar las particiones a través de la biblioteca de pyspark, este nos marcará error al ejecutar las tareas de forma paralela, por la pérdida de referencia en las particiones a esas variables locales. Para ello se crearon distintas clases que podemos observar en el diagrama UML de la Fig. 42, donde se muestra la definición de sus atributos.

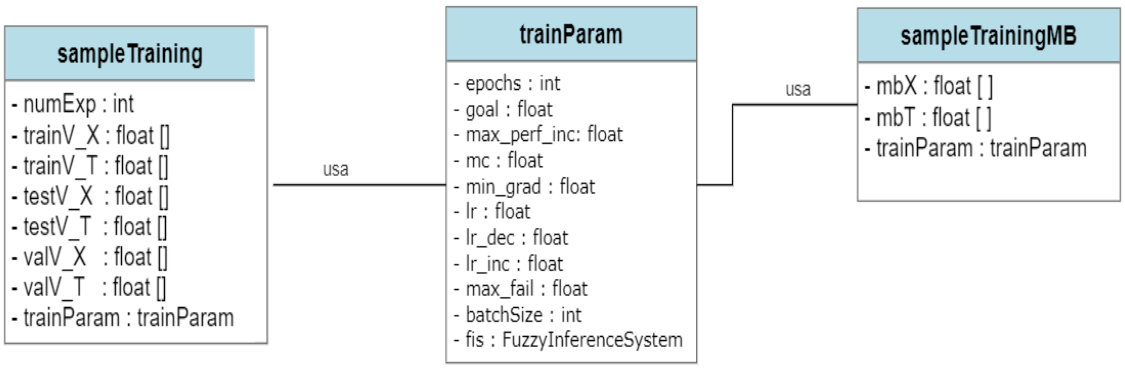


Fig. 42. Clases utilizadas para encapsular datos (muestras), parámetros de diseño y parámetros de entrenamiento.

A continuación, detallaremos cada una de las clases definidas en el diagrama UML de la Fig 42:

- **sampleTraining:** Cada instancia de esta clase contendrá el número de experimento que le corresponde, y sus respectivas muestras de entrenamiento, validación y prueba, las cuales fueron construidas de forma aleatoria donde el 60% de los datos de la muestra completa corresponderán a entrenamiento, 20% será para validación y el 20% restante corresponderá a pruebas, como puede observarse en la Fig. 44, además de una instancia que contiene la definición de los parámetros de entrenamiento. Cabe destacar que este tipo de instancias puede utilizarse también para el procesamiento en modalidad full batch/offline u online.
- **trainParam:** Contiene la configuración de parámetros que forma parte de los criterios de detención del entrenamiento en caso de que alguno de ellos se activara, esto fue explicado en la sección 3.2 Hiperparámetros y Funcionalidad. A demás contiene una instancia tipo *fis* que encapsula los parámetros de diseño, los cuales serán ajustados durante el entrenamiento. Finalmente, también cuenta con el parámetro de batchSize, el cual representa el número de elementos (datos) dentro de cada mini-batch.
- **sampleTrainingMB:** Contiene los mini-batch que se utilizarán para el entrenamiento, tanto el de entradas (inputs) como el del objetivo deseado (targets), también cuenta con una instancia del tipo trainParam, en este caso, porque nos interesa tener acceso al valor establecido en batchSize y *fis* (el cual será contendrá los parámetros de diseño ajustado por cada época) que será requerido para los procesos de *Feedforward* y Backpropagation procedimientos que fueron explicados en la sección 3.1 Descripción del sistema propuesto.

2. El diagrama de flujo de la Fig. 43, da muestra general de los procesos básicos para llevar a cabo el entrenamiento a partir del procesamiento de mini-batch en un entorno paralelizado.

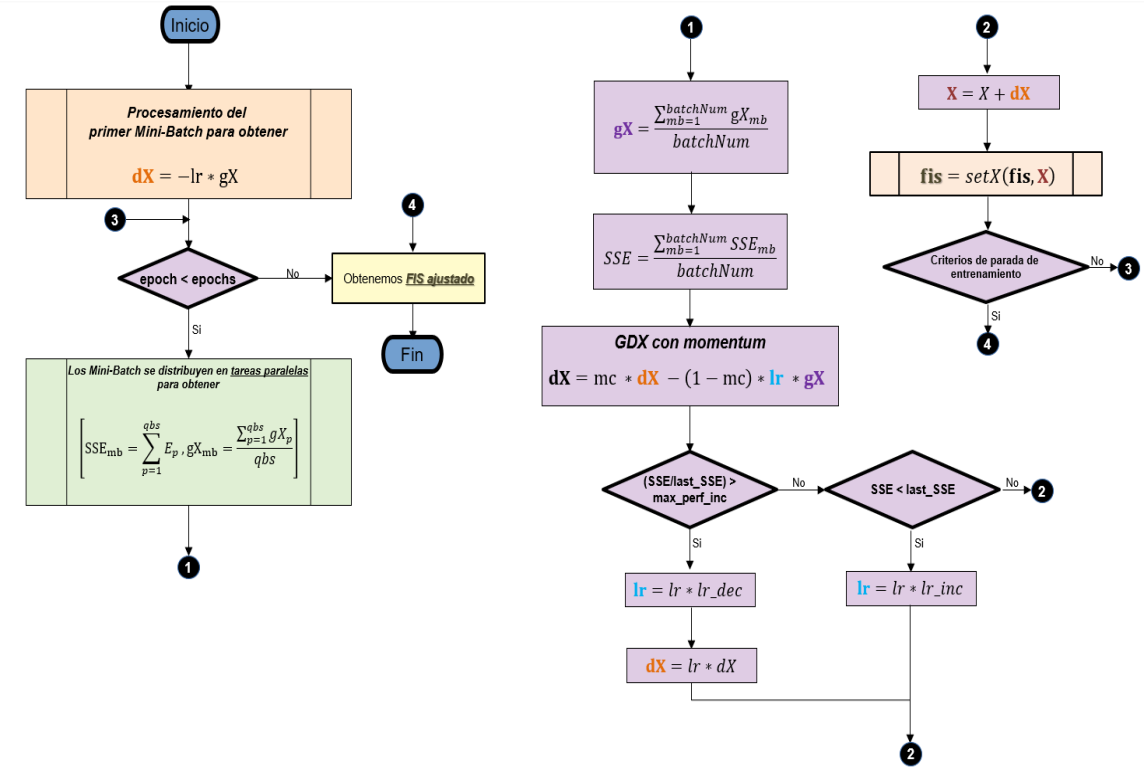


Fig. 43. Diagrama de flujo del entrenamiento de mini-batch sobre un entorno paralelizado.

Como puede observarse en la Fig. 43, el inicio del entrenamiento requiere el procesamiento completo (*offline*) del primer mini-batch para obtener el cambio de los parámetros de diseño (dX) a partir del cálculo del vector gradiente (gX) al cual se le aplica la tasa de aprendizaje (lr), es decir, la regla delta de aprendizaje (32).

$$dX = -lr * gX \tag{32}$$

Por lo tanto, dX será tomado como el cambio previo de los parámetros de diseño necesario para llevar a cabo la ejecución del algoritmo de optimización *GDX con momentum* (Gradiente Descendente con Momentum y Tasa de Aprendizaje Adaptiva), el cual es representado en la siguiente ecuación (33).

$$dX = mc * dX - (1 - mc) * lr * gX \tag{33}$$

Una vez comenzado la iteración por época, se construyen las particiones utilizando la biblioteca de *pyspark* y su método *parallelize,* el cual construirá una *instancia RDD* que contendrá dichas particiones. Las particiones alojarán un *mini-batch* (instancia de la clase *sampleTrainingMB*), como se observa en la Fig. 44.

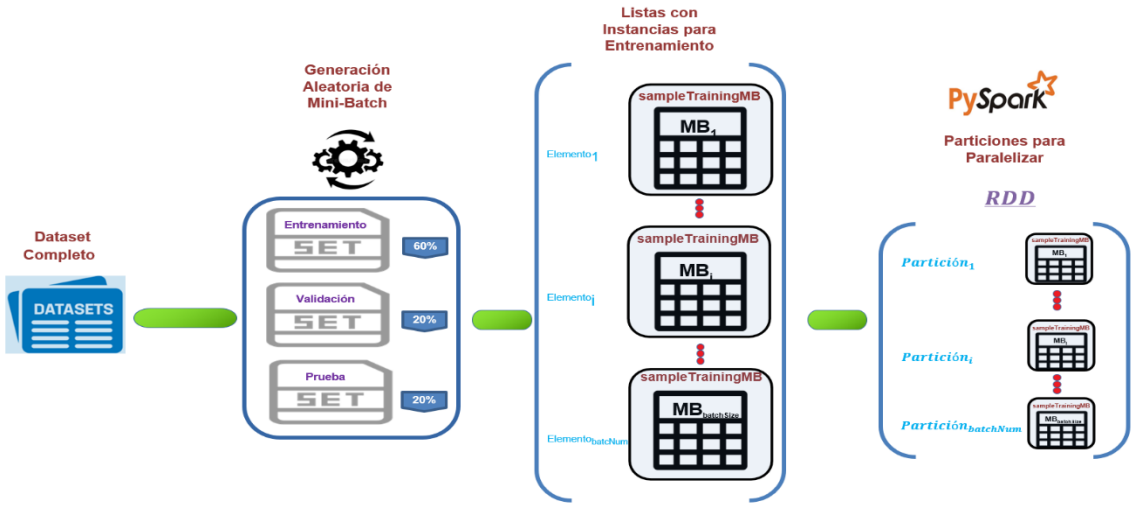


Fig. 44. Esquema de construcción de Mini-Batch y Particiones del RDD.

Hasta el momento a cada partición solamente se le ha adjuntado sus muestras de datos en mini-batch, ahora será necesario añadirles las instrucciones de procesamiento y aprendizaje, en este caso sería la rutina de entrenamiento, a través de su mapeo a cada *partición del RDD* como se muestra en la Fig. 45.

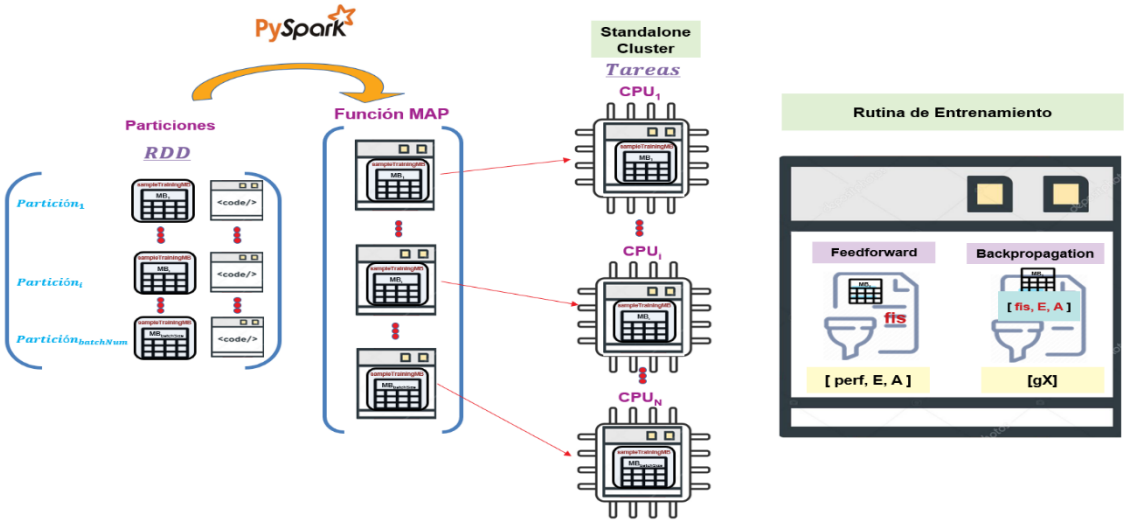


Fig. 45. Esquema del mapeo de las funciones o métodos y las particiones de Mini-Batch para su procesamiento en paralelo.

Esto se logra utilizando la función *map* que forma parte de la biblioteca *pyspark,* la cual es accedida a través de una *instancia RDD* y requiere una función o método que en este caso sería la *rutina de entrenamiento* que internamente contiene las funciones de *feedforward* y *backpropagation* (esta combinación realizada de forma iterativa, conforman el entrenamiento o aprendizaje de la red neuronal)*,* a partir de este momento contamos con la configuración completa, es decir, hemos realizado las transformaciones necesarias (evaluaciones perezosas, explicadas en el apartado 3.3.1) y estamos en condiciones de llamar también a través del *RDD* a la función *collect (acción) para* ejecutar el procesamiento y entrenamiento de los datos en mini-batch de forma paralelizada. Como puede observarse, se utiliza un clúster en modo Standalone para administrar los recursos hardware y distribuir las tareas en los distintos procesadores con que contamos en el servidor donde se realizará el procesamiento.

La *rutina de entrenamiento* internamente toma el mini-batch de la partición correspondiente y lo recorre dato a dato, como se observa en la Fig. 46, primero se calculan las *señales de salida* (A) por cada capa a través de la función *feedforward*, del cual además se obtiene; $E = \hat{y}_p - A_p$ (error calculado, a partir de la diferencia del valor deseado $\hat{y}_p$ y valor estimado A_p) y $perf = {E_p}^2$ (error elevado al cuadrado), seguido a ello se acumula $perf$ en la variable SSE_{mb} .

Posteriormente se ejecuta *backpropagation* para poder determinar el ajuste de los parámetros de diseño (que fungen como pesos) a partir de las derivadas parciales del error (función de coste) con respecto a los parámetros de diseño por cada capa, y obtener así el *vector gradiente* (gX) el cual contiene el cambio direccional que permitirá que en cada nueva época vayamos reduciendo el error y poder así aproximarnos cada vez más al valor deseado.

El vector gradiente (gX) será acumulado y una vez recorrido todos los datos del mini-batch, se calcula el *promedio del vector gradiente dividiendo el vector gradiente acumulado* (sum_gX) entre el *número de datos procesados en el mini-batch (*qbs*)*, finalmente lo que se regresa a la *función principal de entrenamiento* es una estructura tipo *lista* que contendrá SSE_{mb} (Suma de los Errores al Cuadrado) y gX_{mb} (vector gradiente promediado) por cada mini-batch.

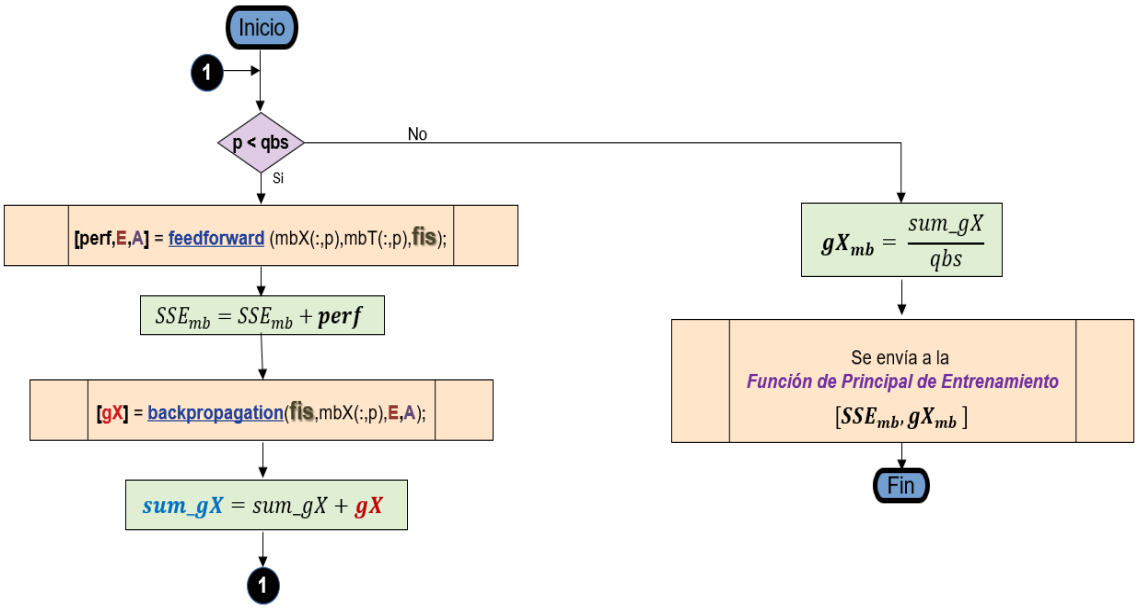


Fig. 46. Diagrama de Flujo de los procesos que contendrán las tareas cuando son paralelizadas.

Siguiendo con la descripción de los procesos del diagrama de flujo de la Fig. 43, se observa que para poder ejecutar el *GDX con momentum*, es necesario antes promediar los SSE_{mb} y gX_{mb} entre el *número de mini-batch procesados* ($batchNum$).

Finalmente, antes de decidir actualizar los parámetros de diseño, se evalúa la siguiente heurística:

- Si la proporción del SSE_{actual} con respecto al $SSE_{anterior}$ es mayor al $\max\ \_perf\_inc$ (valor máximo del error permitido), decrementamos la tasa de aprendizaje (lr) con respecto al valor establecido en el parámetro de entrenamiento lr_dec y posteriormente aplicamos el cambio al vector que contiene los parámetros de diseño (dX) calculado previamente con *GDX con momentum.*
- En caso de que la validación anterior no sea verdadera, entonces se pregunta si SSE_{actual} es menor que $SSE_{anterior}$ y si es verdadero entonces se incrementa la tasa de aprendizaje (lr) a la razón establecida en lr_inc (parámetro de entrenamiento que contiene el valor de incremento de la tasa de aprendizaje).
- Y en cualquiera de los dos casos, actualizaremos los parámetros de diseño vectorizados (X), por último, con este mismo se actualiza también el fis.

- Se evalúan los criterios de detención de entrenamiento y si no se activa al menos uno y aún no se ha llegado al límite de épocas establecido, entonces continuamos con el entrenamiento y ajuste del fis dentro de la red neuronal, en caso contrario, se termina el entrenamiento y se envía como resultado del proceso de aprendizaje un fis ajustado, es decir, que contiene los parámetros pseudo-optimos que componen el modelo difuso sobre el cual podremos evaluar con diferentes muestras de entrada y objetivos, para ejecutar tareas de tipo regresión y clasificación.

3. El escenario de desarrollo que previamente expliqué se llevó a cabo en ***modo Standalone***, el cual permite tener en tu propia computadora el motor de Apache Spark integrado con un simple administrador del clúster, para poder establecer la ***cantidad de workers (y sus ejecutores)*** sobre los cuales se van a paralelizar las tareas, y además poder asignarles la cantidad de memoria y las unidades de procesamiento para llevar a cabo su trabajo.

   Durante la etapa de desarrollo e implementación del Sistema Neuro-Difuso propuesto en el entorno de Spark, otra necesidad importante fue identificar y establecer los recursos adecuados que el clúster en modo Standalone requería para poder realizar una apropiada ejecución del entrenamiento, ya que, al ir incrementando el número de instancias en millones, tendía a provocar el error de *"fuera de memoria: Espacio de Almacenamiento Dinámico de Java (Out of Memory: Java Heap Space)".*

   De acuerdo con la documentación de Spark [1], su motor fue escrito en lenguaje Scala el cual se compila en código de bytes, por lo tanto, puede ejecutarse sobre Java Virtual Machine (JVM), este último es un motor (compilador/interprete) que ejecuta instrucciones compiladas en código de bytes sobre múltiples plataformas, como se muestra en el esquema de la Fig. 47.

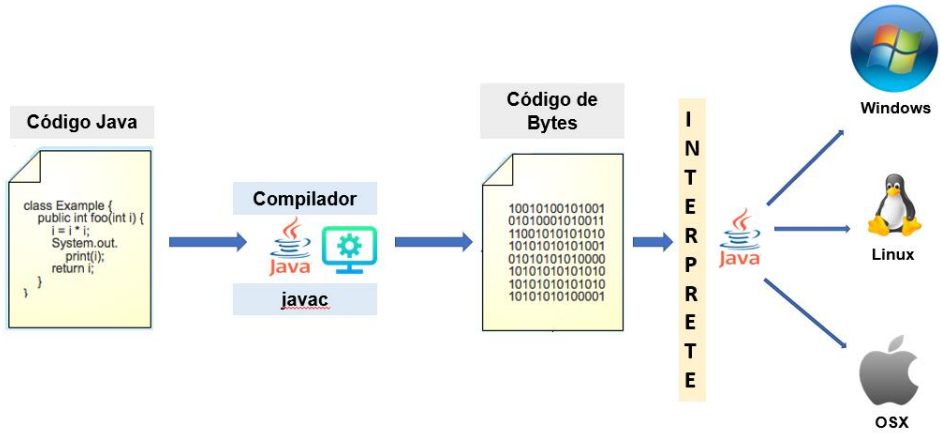


Fig. 47. Esquema que describe el trabajo y arquitectura de JVM.

Tanto el Driver y como el Worker son dos componentes principales dentro del clúster de Spark, y ambos son procesos de JVM (como se observa en la Fig.48), por lo tanto, era importante conocer las mejores prácticas sobre la distribución y asignación de memoria, para que en tiempo de ejecución se evitara la sobrecarga o desbordamiento en memoria y por ende la interrupción del entrenamiento.

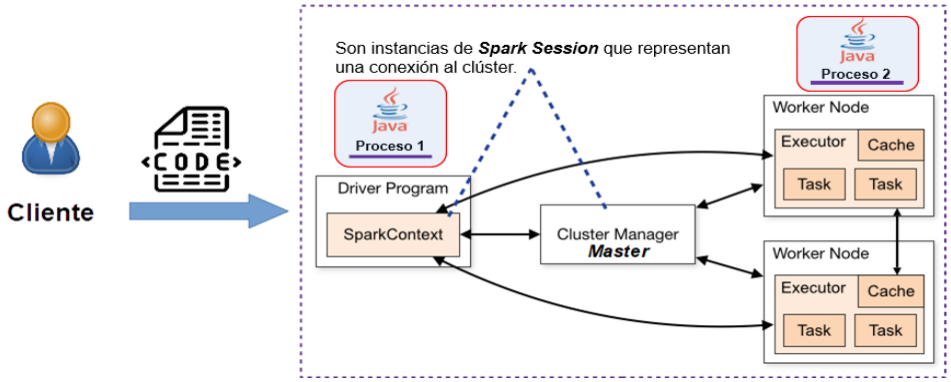


Fig. 48. Esquema que permite observar al Driver y Worker como procesos de JVM.

En un clúster que se ejecuta en modo Standalone, los *Workers* y *Executors* tienen una relación 1:1, es decir, a cada Worker le corresponde un *Executor*, y este último es de gran relevancia, ya que todo el cómputo es realizado sobre el.

Se puede cambiar esta relación por defecto, si se modifica el parámetro de ***spark.executor.cores*** por un valor diferente a 1, ya que cada core fungirá como un ***Executor***. A través del servicio Web que proporciona el clúster de Spark se puede constatar dicha relación entre estos dos componentes y también la modificación que se comenta sobre el parámetro de número de cores por Executor, como se muestra en la Fig. 49.

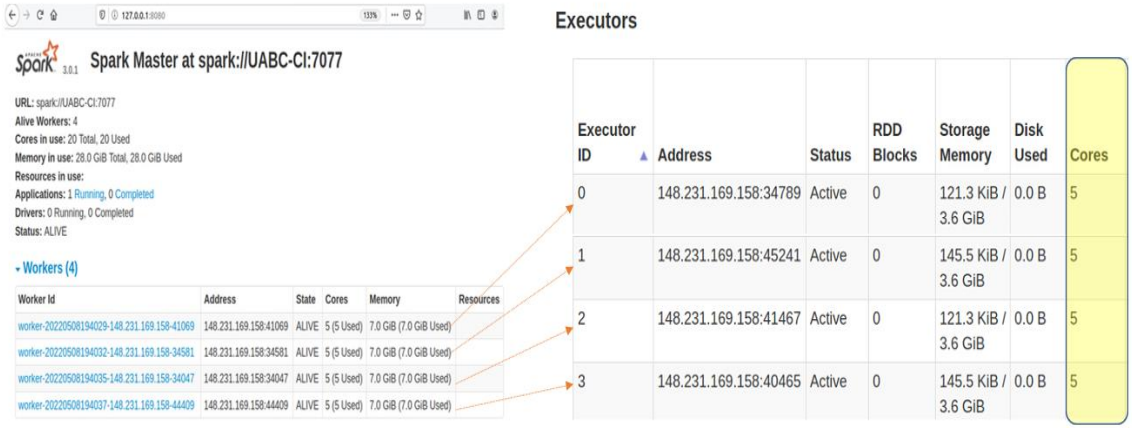


Fig. 49. Relación de Workers y Executors, así como la asignación de más Executor por asignación de cores.

Hay dos escenarios muy recurrentes por la falta de conocimiento de una adecuada distribución y asignación tanto de *Executors* como de la memoria que estos utilizarán, esto durante su ejecución puede generar excepciones de error por inadecuado manejo de memoria, los cuales son:

1. La asignación de *Executors* con una gran cantidad de núcleos virtuales conduce a una baja cantidad de Executors y un paralelismo reducido, así como una asignación de memoria excesiva que resultará en retrasos o interrupción del proceso por sobrecargar el recolector de basura o *Garbage Collector (GC)* cada vez que JVM identifica un objeto que no se está utilizando y lo envía a este basurero, como se observa en la Fig. 50.

Fig. 50. Esquema de función que desempeña GC cuando un JVM desecha un objeto que no está en uso.

De acuerdo con la documentación de Java [59], de forma predeterminada, JVM está configurado para generar el mensaje de *"Error Fuera de Memoria (Out of Memory Error)"* si el proceso de Java pasa más del 98% de su tiempo realizando la función de GC y cuando solo se recupera menos del 2% del almacenamiento dinámico en cada ejecución (Java Heap Space), esto significa que nuestra aplicación ha agotado casi toda la memoria disponible y GC ha dedicado demasiado tiempo a intentar limpiarla y ha fallado repetidamente.

2. La asignación de una cantidad baja de núcleos virtuales conduce a una gran cantidad de Executors, lo que provoca una mayor cantidad de operaciones de entradas y salidas. Correr Executors diminutos (1 solo núcleo con memoria suficiente para ejecutar una sola tarea) descarta los beneficios que se obtienen al ejecutar múltiples tareas en una sola JVM.

Con base a recomendaciones acerca de mejores prácticas para la asignación de recursos sobre un clúster de Spark que se da en [2], explican que:

- Estudios basados en datos históricos sugieren que se tengan *5 núcleos virtuales* (vCPU) por cada *executor* para lograr resultados óptimos sobre clústeres de cualquier tamaño.

  spark.executors.cores = 5 (vCPU)

  Se recomienda que el número de cores para el *Driver* sea igual al asignado a los executors, en caso de no contar con los recursos para hacer esto posible, se ajusta a lo que se tenga, en nuestro caso al *Driver* se le asignaron 4 núcleos, ya que 1 debe dejarse para operaciones del demonio de Hadoop.

  Recomendado:
  spark.driver.cores = spark.executor.cores

  En nuestro caso:
  spark.driver.cores = 4

- Para determinar el número de *Executors*, se recomienda realizar los siguientes cálculos:

$$\textbf{Número de Executors} = \frac{\textbf{Total de vCPU} - \textbf{1 vCPU}}{\textbf{spark. executors. cores}}$$

  En nuestro caso, el servidor que se utilizó cuenta con *24 (vCPU)*, sin embargo, se recomienda no utilizar todos, ya que el *Driver* también requerirá tener sus propios recursos (como se comentó en el punto anterior), es por ello, que se han determinado *21 vCPU* solo para los *Executors*, sin embargo, dentro del cálculo se observa la resta de *1 vCPU*, esto es, porque se debe reservar 1 núcleo para los procesos de tipo demonio de Hadoop.

$$\textbf{Número de Executors} = \frac{\textbf{21} - \textbf{1}}{\textbf{5}} = \textbf{4 executors}$$

- Ahora debemos determinar la memoria total de los *Executors* y el *Driver*, para ello los cálculos a realizar son:

$$\textbf{Total del Memoria por Executors} = \frac{\textbf{Total de Memoria RAM} - \textbf{1 GB (demonio Hadoop)}}{\textbf{Número de Executors}}$$

El servidor que se utilizó cuenta con $47.1\ GB\ RAM\ \approx 47\ GB$, sin embargo, como se mencionó debemos establecer recursos de memoria tanto para el *Driver* como para los procesos del demonio de Hadoop, por ende, no podemos utilizar la memoria total para los *Executors*, en este caso, hemos determinado $30\ GB\ RAM$ para los *Executors*, $1\ GB\ RAM$ para el *demonio de Hadoop* y $16\ GB\ RAM$ para el *Driver*.

$$\boldsymbol{Total\ del\ Memoria\ por\ Executors} = \frac{\mathbf{30\ -1}}{\mathbf{4}} = \mathbf{7.25\ GB\ RAM\ \approx 7\ GB\ RAM}$$

Y se deberá establecer de la siguiente manera; $\boldsymbol{spark.executors.memory = 7}$, se recomienda que la instancia Driver tenga la misma asignación de memoria $\boldsymbol{spark.driver.memory = spark.executors.memory}$, sin embargo, como no contamos con los recursos de memoria para realizar dicha recomendación, generamos el siguiente cálculo, dado que el *Driver* es una sola instancia:

$$\boldsymbol{Total\ del\ Memoria\ para\ el\ Driver} = \frac{\boldsymbol{Total\ de\ Memoria\ RAM\ \ para\ Driver}}{\boldsymbol{Número\ de\ Cores\ del\ Driver}}$$

$$\boldsymbol{Total\ del\ Memoria\ para\ el\ Driver} = \frac{\mathbf{16}}{\mathbf{4}} = \mathbf{4\ GB\ RAM\ por\ vCPUen\ el\ Driver}$$

Una vez determinado el número de *Executors*, la *asignación de núcleos* y *memoria RAM*, cada *Executor* de manera interna administrará y distribuirá la memoria asignada de la siguiente manera como se muestra en el esquema de la Fig.51.

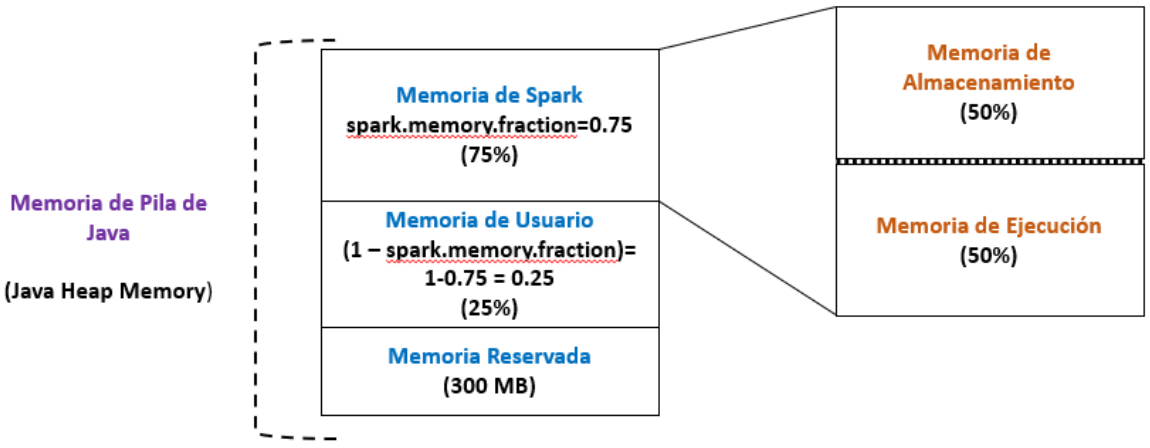


Fig. 51. Esquema de distribución de memoria de un Executor de Spark.

Con base a la Fig. 51, a continuación, se describen cada uno de los elementos que componen dicho esquema:

- **Memoria de Pila de Java (Java Heap Memory)**: es el área de memoria utilizada para almacenar los objetos instanciados por las aplicaciones que se ejecutan sobre el JVM, en el caso del entorno de Spark, serían las tareas que se están ejecutando dentro del Executor de forma paralelizada.
- **Memoria de Spark**: segmento de memoria gestionado por Spark, es responsable de almacenar estados intermediarios (como transformaciones tipo join o variables brodcast), así como los datos que se almacenan en caché o persistentes (estos últimos serán manejados por el segmento memoria de almacenamiento).
    - **Memoria de Almacenamiento:** utilizado para almacenar todos los datos en caché o cualquier operación persistente que incluya memoria en ella. Spark libera espacio para nuevas solicitudes de caché mediante la eliminación de objetos almacenados en caché antiguos en función del mecanismo de uso menos reciente.
    - **Memoria de Ejecución:** este segmento es utilizado para objetos creados durante la ejecución de una tarea, también puede combinarse con almacenamiento en disco si no hay suficiente memoria disponible, pero los bloques generados en este segmento no pueden ser desalojados a la fuerza por otras tareas.
- **Memoria de Usuario:** es el área de memoria que almacena todas las estructuras de datos definidas por el usuario.
- **Memoria Reservada:** almacena objetos internos de Spark, esta designado por default $300\ MB\ RAM$ para esta tarea y solo puede ser cambiado si Spark es recompilado.

4. Se presenta la siguiente Fig. 52 para mostrar un diagrama de secuencia, donde se observan los procesos que se llevan a cabo en el *Sistema Neuro-Difuso propuesto* y su despliegue sobre el *escenario Standalone de Spark.*

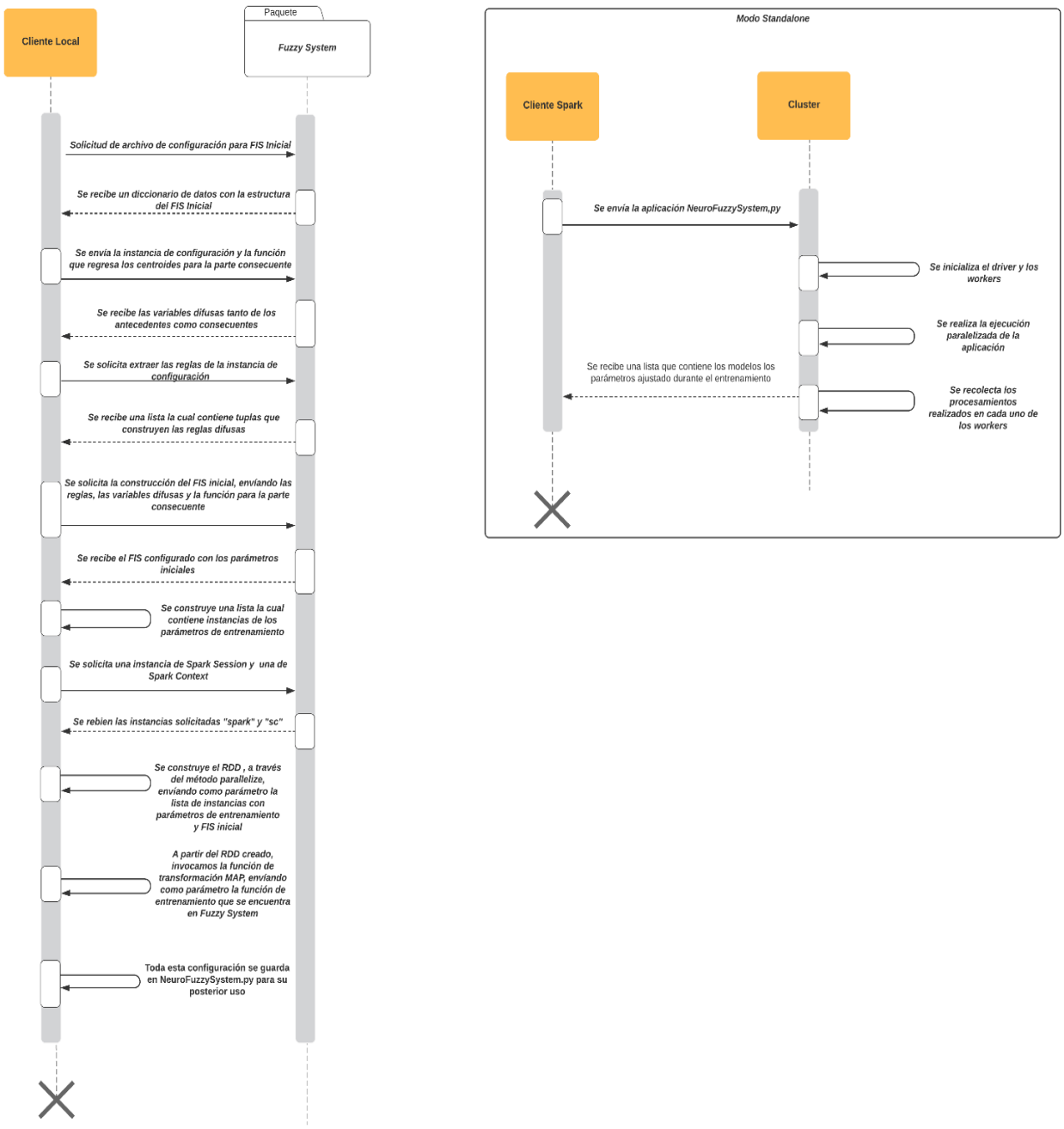


Fig. 52. Diagrama de procesos del Sistema Neuro-Difuso propuesto y su despliegue en modo Standalone.

Las características de Software y Hardware donde se llevó a cabo este entorno de desarrollo en modo Standalone fueron los siguientes:

- Sistema Operativo: Ubuntu 19.10
- Procesador: Intel Xeon CPU X5680
- Núcleos: 24
- Memoria: 47.1 GB
- Disco: 1 TB

CAPÍTULO 4. Estudio experimental.

4.1 Resultados y discusiones.

En esta sección, se presenta la metodología de experimentación y los resultados obtenidos tanto de análisis de regresión como de clasificación. El objetivo de estos análisis es llevar a cabo un estudio comparativo del desempeño y estabilidad de modelos construidos a partir del procesamiento paralelizado con aprendizaje basado en mini-batch, los cuales fueron ejecutados sobre una arquitectura Neuro-Difusa con Defusificación con Centro de Conjuntos basados en Mamdani.

La experimentación se llevó a cabo en dos etapas que permitieron ir escalando la arquitectura neuro-difusa propuesta e ir evaluando sus resultados, a continuación, se describen cada una de ellas:

Primera etapa de desarrollo.

En una primera etapa, cuando se adaptó la arquitectura Neuro-Difusa propuesta al entorno paralelizado con el motor de Spark en un ambiente de desarrollo Standalone, se utilizaron datasets con pocas instancias, ya que la intención era poder asegurar que técnicamente funcionara de forma correcta todas las partes identificadas como proclives a ser paralelizadas, estas partes fueron (Fig. 53):

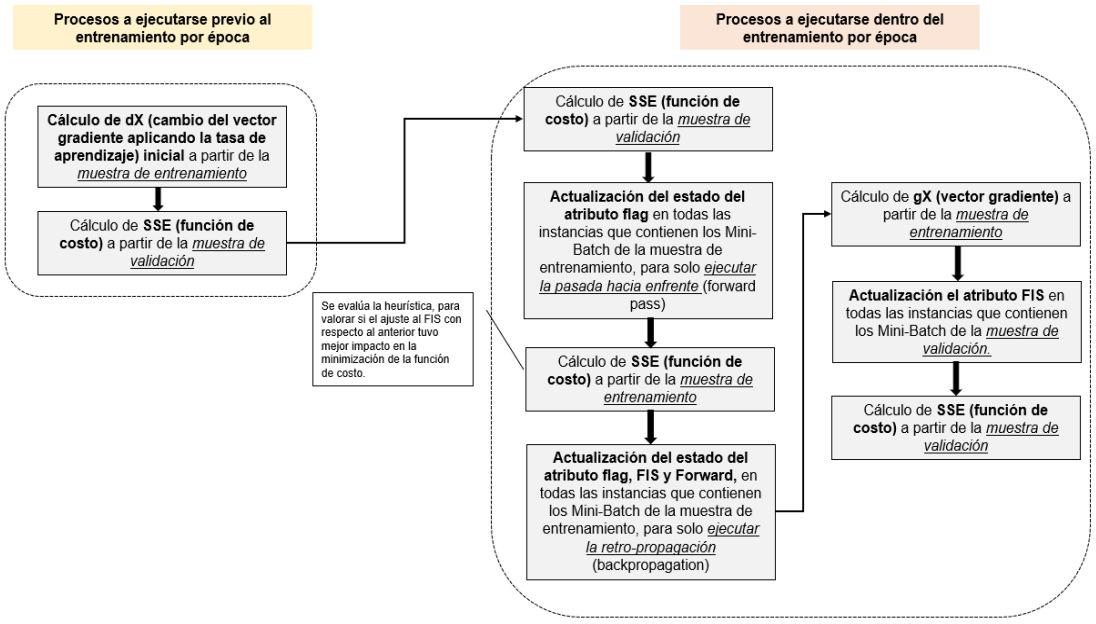


Fig. 53. Diagrama de bloques de procesos proclives a ser paralelizados

Posterior a la realización de las adecuaciones requeridas para que el sistema propuesto funcionara en el entorno Spark, se procedió a probar si funcionalmente tenía un desempeño apropiado. A continuación, se describen los diferentes datasets utilizados en esta etapa, sus características se encuentran definidas en la Tabla 6.

Tabla 6. Características descriptivas de los datasets utilizados en la primera etapa de experimentación

Datasets	Descripción	Total		
		Atributos (Inputs)	Objetivo (Outputs)	Instancias
Synthetic curve	Benchmark de ajuste de datos muy simples	1	1	94
Gauss3	Dos gaussianas combinadas sobre una línea base exponencial decreciente más ruido incluido	1	1	250
Engine behavior	Contiene el comportamiento de dos atributos, estos son Torque y emisiones de óxido de nitroso	2	2	1199
Chemical sensor	Contiene mediciones tomadas de 8 sensores durante un proceso químico	8	1	498
Abalone shell rings	Contiene 8 atributos que describen diferentes conchas	8	1	4177
Bodyfat percentage	Contiene 13 atributos con los cuales podemos estimar el porcentaje de grasa corporal de una	13	1	252

La técnica implementada para la construcción de las muestras fue la de ***validación de submuestreo aleatorio (random sub-sampling)***, las cuales fueron generadas bajo los siguientes porcentajes de distribución; *entrenamiento* con el 60%, *validación* con 20% y *pruebas* con 20%, esto con la finalidad de evitar el sobre ajuste y sobre entrenamiento, así como también garantizar la robustez del sistema neuro-difuso propuesto. El tipo de procesamiento de datos utilizado durante la etapa de aprendizaje fue de Mini-Batch.

Las métricas sobre las cuales se basaron los análisis comparativos fueron:

- ***R coeficiente de correlación:*** muestra la relación de impacto (positiva, negativa, compleja o nula) entre la variable de respuesta esperada y los valores predichos de la variable de respuesta del modelo.
- ***R^2 coeficiente de determinación:*** representa la proporción de la varianza de la variable de respuesta predicha que puede ser explicada por la varianza colectiva de las variables predictoras del modelo.
- ***RMSE raíz del error cuadrático medio:*** nos muestra la desviación estándar de los residuales (error en las predicciones), lo cual permite saber que tan concentrados o dispersos están los datos alrededor de la línea de mejor ajuste.

La siguiente Tabla 7 muestra los resultados obtenidos de la experimentación con base en la primera métrica, como puede observarse se presentan ***mínimo, media y máximo***, así como su ***desviación estándar***, también el ***% de estabilidad en la experimentación***, donde para calcular este indicador se contabilizaron todos los resultados de $\boldsymbol{R}$ que estuvieron entre el $\bar{\boldsymbol{R}}$ y $\max(\boldsymbol{R})$, se dividió entre 30 (total de experimentos realizados) y se multiplicó por 100 para generar el porcentaje.

Tabla 7. Resultados obtenidos a partir del procesamiento en Mini-Batch y bajo el enfoque de evaluación de la métrica R

Dataset	$min(R)$	$\bar{R}$	$max(R)$	σ	% Estabilidad en Experimentación
Synthetic Curve	0.9044	0.9698	0.9975	0.0259	70
Gauss3	0.9725	0.9953	0.9968	0.0043	97
Bodyfat percentage	0.7031	0.8217	0.8560	0.0325	53
Chemical sensor	-0.8182	0.6407	0.8358	0.4760	90
Engine behavior (output 1)	0.9403	0.9702	0.9946	0.0153	80
Engine behavior (output 2)	0.8459	0.8919	0.9652	0.0254	77
Abalone shell rings	0.5862	0.6025	0.6246	0.0069	57

La Fig. 54 muestra que solamente en dos casos el valor de $\bar{\boldsymbol{R}}$ tiende a estar por debajo del 65%, en los demás casos se sostiene por encima del 82% e incluso llegando a 99%, esto nos dice que el 71% de sus experimentos obtuvieron valores que estuvieron en el rango medio calculado y su valor máximo.

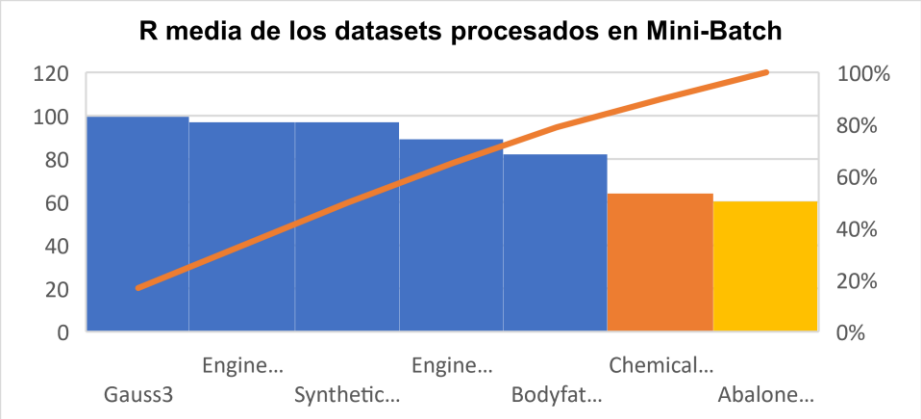


Fig. 54. Comparativa de R media de los datasets procesados en Mini-Batch

También como se comentó en la explicación de la Tabla 7,otro indicador que permite valorar la estabilidad de los modelos generados a partir de la arquitectura neuro-difusa propuesta, es el ***% de estabilidad en la experimentación***, que como bien se muestra en la Fig. 50, solamente en tres casos el valor de este indicador tiende a estar por debajo o igual al 70%, en los demás casos se sostiene por encima del 70% e incluso una máxima del 97%, esto nos dice que el 57% de sus experimentos de los diferentes casos de análisis obtuvieron valores de $\boldsymbol{R}$ que estuvieron en el rango medio calculado y su valor máximo.

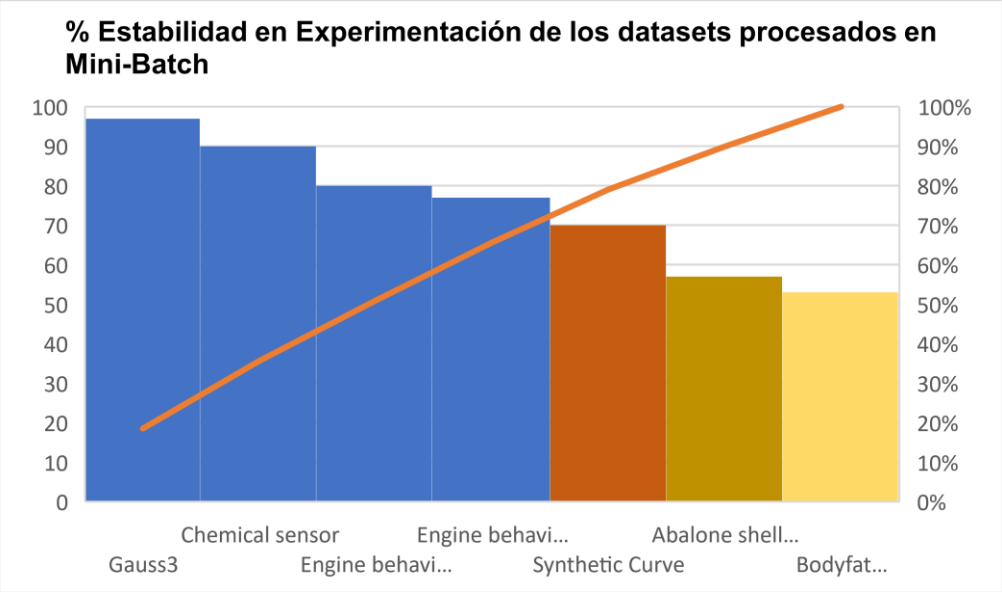


Fig. 55. Comparativa del % Estabilidad en Experimentación entre datasets procesados en Mini-Batch

Otra métrica analizada fue el *coeficiente de determinación* $\boldsymbol{R^2}$ el cual nos permite saber que tan bien se ajustan los datos al modelo generado, la Tabla 8 muestra los resultados por cada caso evaluado, proporcionando información de los resultados obtenidos distribuidos en sus valores; ***mínimo, medio y máximo*** en una ejecución de 30 experimentos, así como también su ***desviación estándar*** $\boldsymbol{\sigma}$ y el ***% estabilidad de la experimentación*** (que fue explicada arriba, bajo la métrica de $\boldsymbol{R}$).

Tabla 8. Resultados obtenidos a partir del procesamiento en Mini-Batch y bajo el enfoque de evaluación de la métrica R^2

Dataset	$min(R^2)$	$\overline{R^2}$	$max(R^2)$	Std	% Estabilidad en Experimentación
Synthetic Curve	0.8160	0.9406	0.9950	0.0503	70
Gauss3	0.9456	0.9906	0.9936	0.0085	97
Bodyfat percentage	0.4923	0.6749	0.7317	0.0519	93
Chemical sensor	0.1403	0.6288	0.6980	0.1152	87
Engine behavior (output 1)	0.8841	0.9415	0.9893	0.0297	80
Engine behavior (output 2)	0.7152	0.7960	0.9316	0.0458	73
Abalone shell rings	0.3435	0.3628	0.3900	0.0084	57

En la Tabla 8 se puede observar que el ***57% (4 de los 7 datasets procesados)*** muestran resultados de su experimentación entre el rango de $\overline{R^2}$ y $\max(R^2)$, de acuerdo con [74] esto indica que si dichos resultados se encuentran en el siguiente rango $0.7 < \overline{R^2} > 1$ entonces se concluye que el poder explicativo del modelo construido a través de aprendizaje es ***fuerte***, el ***29% (2 de los 7 datasets procesados)*** obtuvieron un nivel explicativo ***medio*** ($0.4 < \overline{R^2} > 0.7$) y el ***14% (1 de los 7 datasets procesados)*** con un nivel explicativo ***bajo*** ($0.2 < \overline{R^2} > 0.4$), finalmente como se muestra en la Fig. 56 el % de estabilidad en experimentación tiene un comportamiento similar.

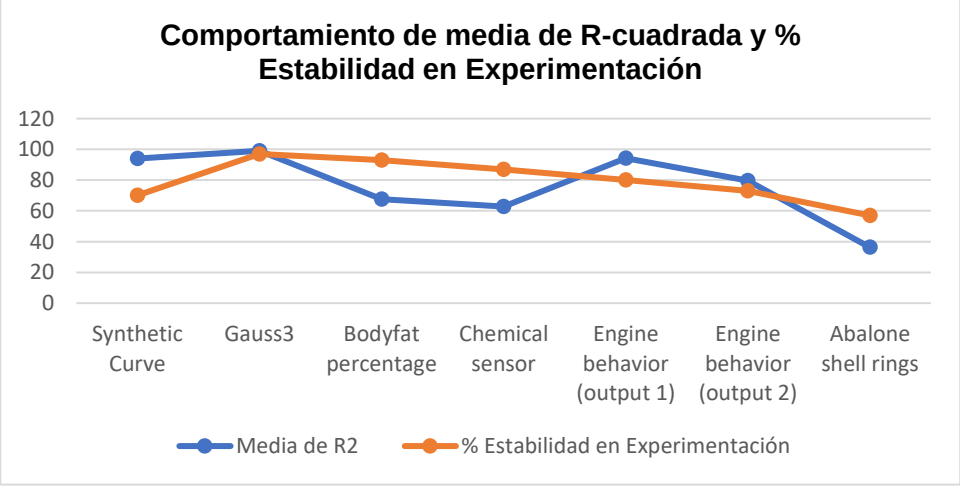


Fig. 56. Comportamiento de media de R-cuadrada y % Estabilidad en Experimentación

La última métrica evaluada fue ***RMSE*** (raíz del error cuadrático medio por sus siglas en inglés) o bien conocida como la ***desviación estándar de los errores*** la cual nos muestra la dispersión de las diferencias obtenidas entre lo estimado y el valor esperado en el modelo.

La Tabla 9 además de presentar los resultados ***mínimos, medios y máximos*** con base al ***RMSE*** calculado en los 30 experimentos por cada dataset, también se agregó la ***desviación estándar*** y el ***Coeficiente de Variación (C.V.)***, este último de relevancia, ya que al ser un indicador porcentual permite visualizar fácilmente la robustez y confiabilidad del modelo generado durante el aprendizaje, como se puede observar tanto en la Tabla 9 como en la Fig. 57 todos los datasets procesados obtuvieron ***menos del 50% coeficiente de variación***, el ***57% (4 de los 7 datasets procesados)*** estuvieron en el rango del $\mathbf{24} < \%\boldsymbol{C.V.} < \mathbf{25}$ y el ***43% (3 de los 7 datasets procesados)*** estuvieron en el rango del $\mathbf{2} < \%\boldsymbol{C.V.} < \mathbf{11}$, esto permite validar que los datos procesados sobre la arquitectura neuro-difusa propuesta genera modelos estables y confiables, que pueden mejorar si se implementa algún otro método de optimización que permita minimizar aún más la función de costo que en este caso de estudio la utilizada fue SSE (suma de los errores al cuadrado por sus siglas en ingles).

Tabla 9. Evaluación de los resultados obtenidos bajo la métrica RMSE.

Dataset	$min(RMSE)$	$\overline{RMSE}$	$max(RMSE)$	Std	% C.V.
Synthetic Curve	0.20	0.60	1.05	0.26	44.28
Gauss3	3.20	3.72	9.06	1.02	27.38
Bodyfat percentage	3.30	3.84	4.58	0.33	8.59
Chemical sensor	0.44	3.19	7.55	1.35	42.43
Engine behavior (output 1)	55.80	116.75	159.06	28.72	24.60
Engine behavior (output 2)	113.88	178.34	203.48	19.51	10.94
Abalone shell rings	1.40	1.48	1.60	0.04	2.85

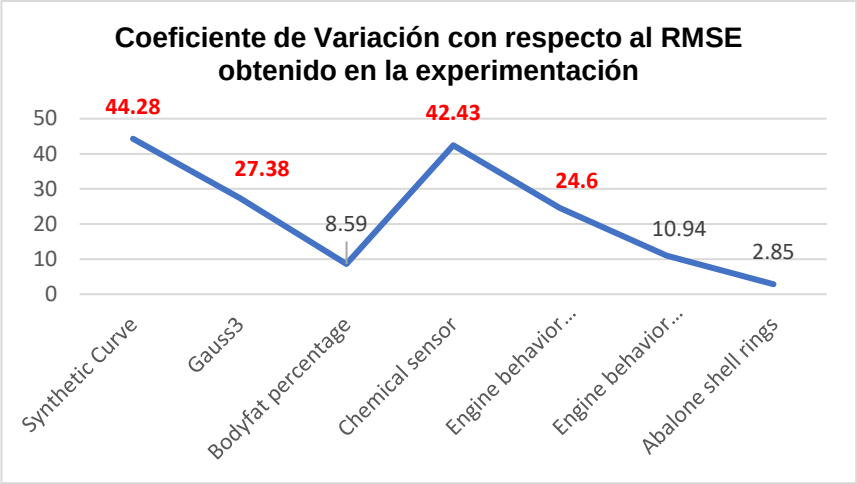


Fig. 57. Comportamiento de los Coeficientes de Variación con respecto al RMSE obtenido en la experimentación de los diferentes datasets.

Segunda etapa de desarrollo.

Después de haber realizado la asignación de recursos de forma apropiada para el clúster de Spark en modo Standalone y sus componentes principales, la experimentación con datasets que tuvieran mayor número de instancias se llevó a cabo, iniciando con *una tarea de clasificación*, a continuación, se describe el dataset utilizado y el proceso realizado para su ejecución de forma paralelizada sobre la arquitectura neuro-difusa propuesta:

- El dataset se obtuvo del sitio Kaggle [3], fue proporcionado por la Oficina de Estadística de Transporte de Estados Unidos, en el cual se realiza un seguimiento de la puntualidad de los vuelos nacionales operados por grandes compañías aéreas.
- Cuenta con registro de datos del 2009 al 2018, y está compuesto por 27 atributos que describen en su conjunto información del tipo; *cantidad de vuelos a tiempo, retrasados, cancelados, desviados, entre otros.*
- Se realizó un pre-procesamiento a partir del análisis de los datos y se realizaron las siguientes acciones; se eliminaron 4 atributos ya que en el 82% de sus registros contaban con datos nulos, se cambiaron datos de textos por alguna equivalencia numérica y se reconstruyo el dataset ya que había un fuerte desbalance de clases (cuando una o más clases se encuentran menos representas en números de muestras, en comparación con muestras de otras clases) el 98% de los registros correspondían a la clase 0 y el 2% a la clase 1.
- El dataset quedo ajustado a un total de 1’666,666 registros, de la cual se divide de forma equitativa 50% para ambas clases.

- Las muestras fueron distribuidas de la siguiente manera; *entrenamiento* 60% (1'000,000), *validación* 20% (333,333) y *prueba* 20% (333,333).
- Para efectos de llevar a cabo la *tarea de clasificación*, *21 atributos* se dejaron como *entradas* y *1 atributo* fue designado como *objetivo* (*target*) este fue el de *cancelación*, el cual paso por un proceso de binarización que producía dos clases: 0 = no cancelado y 1=cancelado.

Las métricas utilizadas para evaluar el desempeño, estabilidad y robustez del modelo generado a partir de la arquitectura neuro-difusa propuesta fueron:

- **Matriz de Confusión:** técnica que permite resumir y visualizar de forma sencilla el desempeño que ha tenido un algoritmo de clasificación, brinda información tanto de predicciones correctas como incorrecta, agrupando y realizando un conteo de dichas predicciones en verdaderos positivos/negativos y falsos positivos/negativos respectivamente, como se muestra en la Fig. 58.

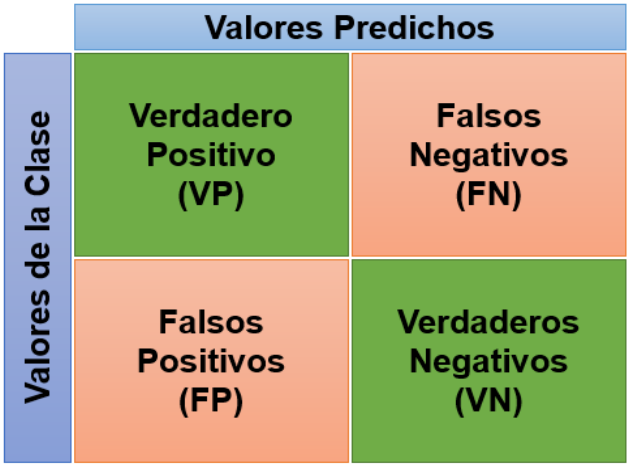


Fig. 58. Ejemplo de una Matriz de Confusión

- **Precisión (accurancy):** resume el desempeño de un modelo de clasificación a través del conteo de número de predicciones correctas y dividido entre el número de predicciones hechas.

$$Precision = \frac{VP + VN}{VP + FP + FN + VN}$$

- **Exactitud (precision):** muestra la proporción de identificaciones positivas que fueron clasificadas de forma correcta por el modelo, es decir, cuando una clase es predicha el porcentaje de exactitud representará la veces que esa clasificación fue correcta.

$$Exactitud = \frac{VP}{VP + FP}$$

- **Sensibilidad (recall):** muestra la tasa de verdaderos positivos, es decir, nos da la proporción de identificación correcta por clase.

$$\boldsymbol{Sensibilidad} = \frac{\boldsymbol{VP}}{\boldsymbol{VP + FN}}$$

- **Especificidad (specificity):** evalúa la capacidad del modelo de para predecir verdaderos negativos de cada clase disponible.

$$\boldsymbol{Especificidad} = \frac{\boldsymbol{VN}}{\boldsymbol{VN + FP}}$$

- **F1 score:** medida armónica ponderada (promedio) de *exactitud* y *sensibilidad,* evalúa el desempeño entre clases.

$$\boldsymbol{F1\ score} = 2\ \frac{\boldsymbol{Exactitud} * \boldsymbol{Sensibilidad}}{\boldsymbol{Exactitud + Sesibilidad}}$$

A partir del modelo producido a través de la arquitectura neuro-difusa propuesta y al haber realizado la evaluación sobre el FIS con una muestra de prueba (desconocida por el modelo), los resultados obtenidos fueron los siguientes:

1. En la matriz de confusión para cada una de las clases, puede observarse que casi el 99.99% tanto para predicciones verdaderos positivos/negativos fueron generadas por el clasificador y solo un 0.006% de error para la clase 0 y un 0.015% de error para la clase 1, como puede observarse en la Fig. 59.

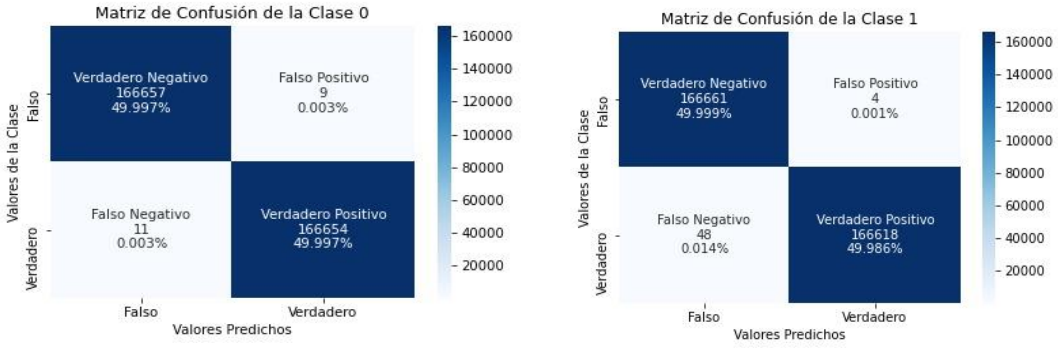


Fig. 59. Matriz de Confusión para la Clase 0 y Clase 1.

Tabla 10. Resultado obtenido del Clasificador (modelo producido por el neuro-difuso propuesto).

Clase	Precisión (accurancy)	Exactitud (precision)	Sensibilidad (recall)	Especificidad (specificity)	F1 score
0	**0.9999**	**0.9999**	**0.9999**	**0.9999**	**0.9999**
1	**0.9998**	**0.9997**	**0.9997**	**0.9999**	**0.9998**

Como puede observarse en los resultados de la Tabla 10 y en la Fig. 59, el modelo generado tiene una buena capacidad de generalización por lo que puede clasificar fácilmente las muestras desconocidas.

Asimismo, durante el entrenamiento para el ajuste de los parámetros de diseño del modelo generado, se estuvieron monitoreando tanto el parámetro de entrenamiento tasa de aprendizaje (lr) como la función de costo (SSE) , a continuación, se describen los resultados:

La Fig. 60 nos muestra el comportamiento adaptivo que tuvo la tasa de aprendizaje durante el entrenamiento, se observa que en las primeras épocas tiene un descenso trepidante muy cercano al 0 y así se mantiene hasta la finalización del entrenamiento a un poco más de 350 épocas, lo cual nos dice que durante la evaluación de la heurística la proporción del error de la época actual con respecto al error de la época anterior era superior al parámetro de entrenamiento que sostenía el límite máximo de incremento de dicha proporción, por tal razón se estuvieron dando estos decrementos en la tasa de aprendizaje hasta que se consiguió estabilizarse.

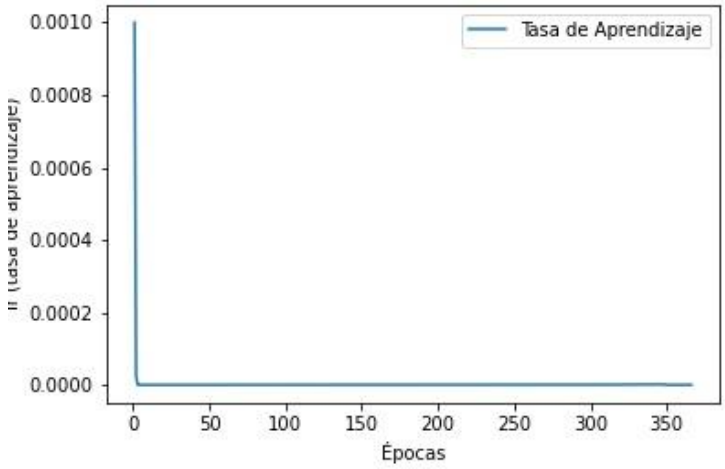


Fig. 60. Comportamiento de la tasa de aprendizaje adaptiva durante el entrenamiento.

La heurística implementada para adaptar la tasa de aprendizaje fue:

- Evaluar si la proporción del $\left(\frac{SSE}{last\_SSE}\right) > \max\_perf\_inc$, donde SSE es el error calculado en la época actual, $last\_SSE$ el error calculado de la época anterior y $\max\_perf\_inc$ parámetro de entrenamiento en el cual se establece el límite máximo permitido para esa proporción, como se muestra en el diagrama de flujo de la Fig.46.
- En caso de que la evaluación sea positiva, se actualiza la tasa de aprendizaje actual, aplicándole el valor de decremento del parámetro de entrenamiento $lr = lr * lr\_dec$.
- En el caso de que la evaluación sea negativa, se valida si el error actual es menor al error anterior $(SSE < last\_SSE)$, y si esta validación es positiva, se aplica un incremento a la tasa de aprendizaje actual $lr = lr * lr\_inc$.
- Esta heurística permitirá encontrar nuevas posiciones del gradiente dentro del área de solución, para evitar caer y estancar el entrenamiento en mínimos locales.

El comportamiento de la función de costo representada por SSE se muestra congruente con el de lr, ya que en las primeras épocas tiene un pronunciado crecimiento (aumento de error), así como se vio en la tasa de aprendizaje, que tendía a aplicar el decremento, dado que la proporción del error con respecto al límite máximo definido era alta.

Así mismo, como se observó en el comportamiento de lr, una vez estabilizado este parámetro, conforme se avanzó en épocas el error continuamente fue disminuyendo hasta que los cambios o decrementos fueron menos notables, como se muestra en la Fig. 61.

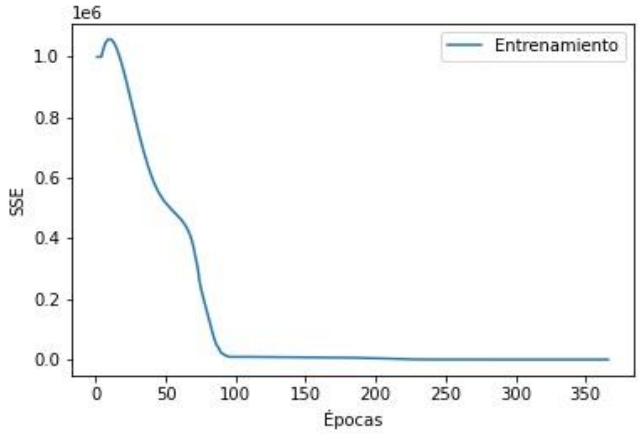


Fig. 61. Comportamiento de la función de costo durante el entrenamiento.

Durante el entrenamiento, también probamos el desempeño del FIS que se iba ajustando, para ello se utilizó la muestra de validación a partir de la cual se calculó las señales de salida en el proceso de pasada hacia adelante (feedforwad), en este mismo proceso se calcula la función de costo $SSE_{validación}$ y si este es mayor que el error calculado SSE a partir de la muestra de entrenamiento, entonces un contador de validación fallida se va incrementando, finalmente, este contador por cada época se evalúa con respecto a un parámetro de entrenamiento $\max\_fail$ y si es mayor al límite definido, entonces el entrenamiento se detiene.

Como se observa en la Fig. 62, en las primeras corridas, el error de la validación va disminuyendo y también se muestra que el error calculado a partir de la muestra de prueba tiende a mantenerse estable por debajo del error calculado a partir de la muestra de validación.

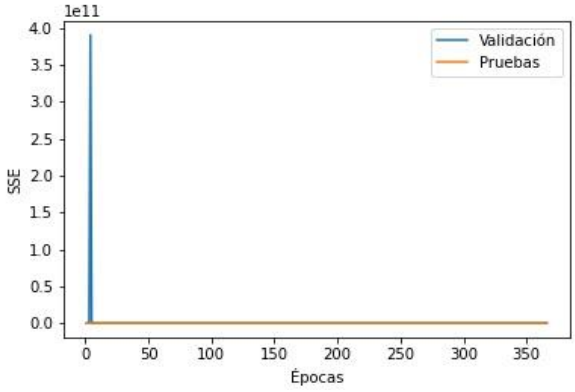


Fig. 62. Comportamiento de la función de costo SSE para la muestra de validación y prueba.

Cabe mencionar que aun cuando SSE no es la métrica que usualmente se utiliza como función de costo o perdida para la optimización y ajuste de los parámetros de diseño del modelo en construcción, en nuestro caso tuvo un buen desempeño, pues las señales de salidas vienen de reglas antecedentes las cuales son caracterizadas por funciones de membresía de tipo Gaussianas, las cuales al evaluar valores singleton, tratarán de hacer un mapeo a valores que van de 0 a 1 (un grado de membresía) lo que asegura una distribución uniforme de los datos, por lo tanto, si asumimos que los datos provienen de una distribución normal, entonces podemos declarar al SSE como una función de costo deseable para la optimización de nuestro modelo, el cual estará captando todo aquel dato predicho discordante respecto al deseable.

Para evaluar el desempeño del modelo generado, se comparan los resultados obtenidos, con respecto a otras técnicas que permiten también llevar a cabo tareas de clasificación, para ello se utilizaron las mismas muestra de datos (entrenamiento, validación y prueba) que en el Sistema Neuro-Difuso propuesto, estos son:

- **Máquina de Vectores de Soporte** (SVM por sus siglas en inglés): actúa como un clasificador discriminativo, a través de un hiperplano que separa las clases identificadas. Los resultados obtenidos fueron:

  La matriz de confusión muestra que la técnica ha podido realizar una buena clasificación, ya que tanto los verdaderos positivos como los verdaderos negativos identificaron correctamente las clases esperadas, solo 81 instancias fueron identificadas de forma errónea como falso negativo, como se observa en la Fig.63.

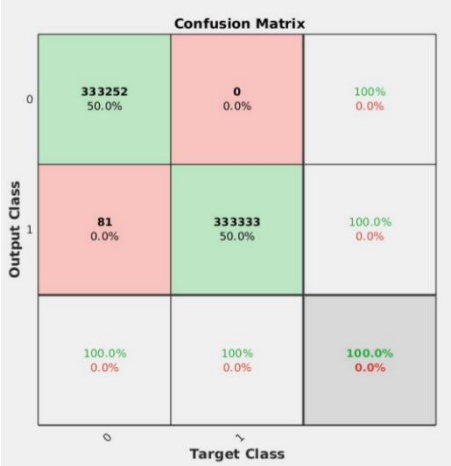


Fig. 63. Matriz de Confusión de la tarea de clasificación procesada en un SVM.

También se calcularon las siguientes métricas:

Tabla 11. Métricas calculadas a partir de la tarea de clasificación procesada en un SVM.

Métrica	Resultado
Regresión	0.9998
Precisión	0.9999
Exactitud	0.9998
Sensibilidad	1
Especificidad	0.9998
F1_Score	0.9999

Como se observa en la Tabla 11, los resultados obtenidos en las distintas métricas dan muestra de que el modelo es preciso, estable, con una alta sensibilidad, especificidad y exactitud con respecto a su tarea de clasificación, además de que es congruente con lo que muestra la matriz de confusión de la Fig.63, ya que solo el 0.01% fueron identificados como falsos negativos y el 99.99% se clasificaron de forma apropiada.

- **Árboles de Decisiones (tree decision):** algoritmo de aprendizaje automático supervisado que utiliza un conjunto de reglas para tomar decisiones. Esta técnica puede ejecutar tareas tanto de clasificación como de regresión. La intuición detrás de esta técnica nos dice que se utilizan todos los atributos del dataset para crear preguntas y respuestas del tipo si/no, y continuamente vamos particionando el dataset hasta que se han aislado todos los puntos de datos pertenecientes a cada clase identificada.

  En este caso, los resultados obtenidos tanto en la matriz de confusión de la Fig.64 como en la Tabla 12, el modelo presenta un excelente desempeño, puede ambas clases fueron identificadas sin error, así como el resto de las métricas.

Fig. 64. Matriz de Confusión de la tarea de clasificación procesada en un Árbol de Decisión.

Tabla 12. Métricas calculadas a partir de la tarea de clasificación procesada en un Árbol de Decisiones.

Métrica	Resultado
Regresión	1
Precisión	1
Exactitud	1
Sensibilidad	1
Especificidad	1
F1_Score	1

Finalmente, las diferencias identificadas entre el modelo generado por SVM y por el Sistema Neuro-Difuso propuesto (se comparan estos dos, pues tuvieron resultados muy cercanos), son los siguientes:

- ***Neuro-Difuso*** *logró identificar 20 observaciones más* colocadas en ***Verdadero Positivo***.
- ***SVM*** *logró identificar 15 observaciones más* como ***Verdadero Negativo***.
- ***SVM*** tuvo *más identificaciones incorrectas siendo un total de 9*, sin embargo, ***todas en Falsos Negativos.***
- ***Neuro-Difuso*** tuvo *menos identificaciones incorrectas en total respecto a SVM*, sin embargo, estuvieron ***divididas entre 13 para Falsos Positivos y 59 Falsos Negativos.***

También se muestra en la Tabla 13, la comparativa de resultados de las diferentes técnicas aplicadas, y podemos observar que el desempeño del Neuro-Difuso propuesto es muy bueno, ya que es sus resultados son cercanos al de las otras técnicas.

Tabla 13. Comparativa de resultados en métricas evaluadas para la tarea de clasificación en diferentes técnicas.

Técnica	Precisión	Exactitud	Sensibilidad	Especificidad	F1_Score
Árboles de Decisiones	1	1	1	1	1
Máquina de Soporte de Vectores	0.9999	0.9998	1	0.9998	0.9999
Sistema Neuro-Difuso propuesto (Clase 0)	0.9999	0.9999	0.9999	0.9999	0.9999
Sistema Neuro-Difuso propuesto (Clase 1)	0.9998	0.9997	0.9997	0.9999	0.9998

Se realizó también la experimentación con una tarea de regresión, el dataset utilizado cuenta con 4,095,000 instancias y 20 atributos, mismo que se obtuvo de UCI Machine Learning Repository [4]. Las fuentes de datos de este dataset corresponden a *14 sensores de gas semiconductores de óxido metálico (MOX) modulados por temperatura*, así como también; *Humedad, Temperatura, Tasa del Flujo, Voltaje del Calentador, Tiempo en el que se registraban dichos datos, y la Captación de Concentración de Monóxido de Carbono (CO)*, este último fue utilizado como atributo objetivo (target) o atributo a predecir.

Debido a que los atributos presentaban una variedad de escalas, se utilizó la técnica de normalización Z-Score, donde por cada atributo se obtiene su media y se divide entre su desviación estándar, de esta manera, la nueva media se convertirá en 0 y su desviación estándar en 1. A continuación se muestra su ecuación (32):

$$Z_{score} = \frac{x - \mu}{\sigma} \tag{32}$$

Donde; $\boldsymbol{x}$ es el valor actual del atributo, $\boldsymbol{\mu}$ es la media del atributo y $\boldsymbol{\sigma}$ la desviación estándar del atributo, buscando normalizar los datos en una escala de 0 a 1, y tratando de eliminar los valores atípicos de este arreglo de valores que representa el atributo.

Como parte del pre-procesamiento, además de la normalización, se eliminó el atributo Tiempo, ya que tendía a sesgar los resultados del modelo, y finalmente se realizó la separación de los datos en muestra de; entrenamiento (60%), prueba (20%) y validación (20%), utilizando la técnica de submuestra aleatoria (random sub-sampling).

Las mismas muestras fueron probadas en diferentes algoritmos para comparar su desempeño con respecto al sistema neuro-difuso propuesto, los algoritmos utilizados y sus resultados fueron:

- **Máquina de Vector de Soporte:** para la tarea de regresión, la línea recta que se requiere para ajustar los datos, se denomina hiperplano, el cual será identificado porque tiene el mayor número de puntos y será utilizado para predecir valores discretos.

  A diferencia de otros modelos de regresión que intentan minimizar el error entre el valor real y el predicho, esta técnica intenta ajustar la mejor línea dentro de un valor umbral, mismo que es representado por la distancia entre el hiperplano y la línea de ajuste.

El resultado de $\boldsymbol{R = 0.9185}$ **,** el cual, al ser coeficiente de correlación entre los valores predichos y los valores observados, nos dice que los atributos de entradas respecto al atributo objetivo están fuertemente correlacionados. A continuación, se muestra la gráfica de regresión en la Fig 65.

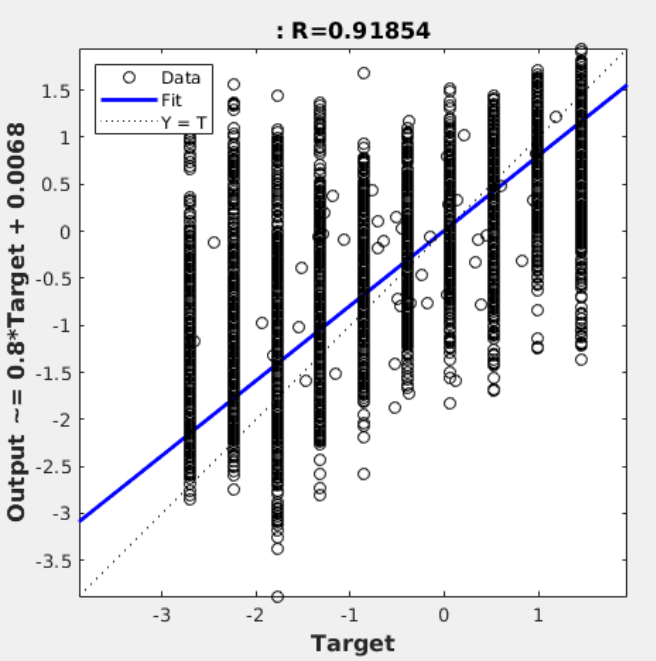


Fig. 65. Gráfica de regresión obtenida del modelo SVM.

- **Redes Neuronales:** algoritmo bio-inspirado, que imita los procesos de aprendizaje del cerebro humano, en tareas de regresión trata de realizar predicciones a partir de datos existentes (conocimiento base). La red neuronal utilizada, se configuró con 10 neuronas, una función de transferencia trainlm (Levenberg-Marquardt), 1000 épocas y el error objetivo se estableció como 0.0001, al término del entrenamiento el resultado de $\boldsymbol{R = 0.9562}$, por lo cual se concluye que la fuerza de correlación entre los atributos y el objetivo a predecir es alta, como también puede observarse en la gráfica Fig 66.

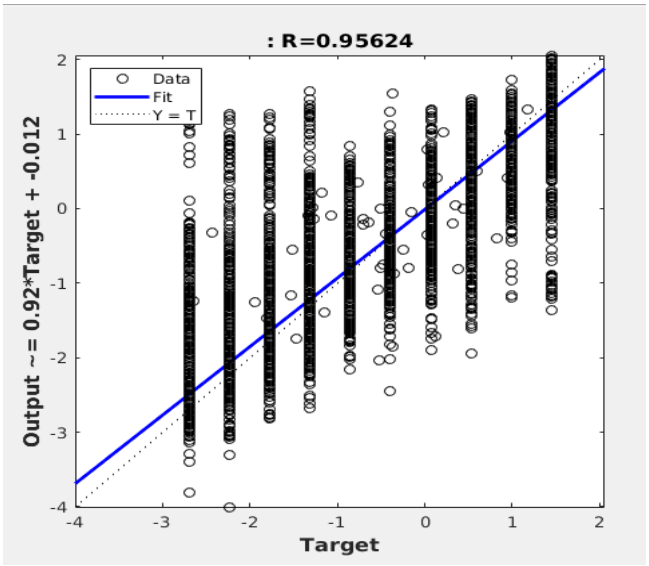


Fig. 66. Gráfico de Regresión para Redes Neuronales.

- **Árbol de Decisiones:** en problemas de regresión, el algoritmo trata de identificar una agrupación de puntos para dibujar un límite de decisión, calculando el error del punto identificado, así lo hace para cada punto, una vez realizado el cálculo se considera el error mínimo como el límite. El resultado obtenido bajo esta técnica fue $\boldsymbol{R = 0.9263}$, sin embargo, esta técnica no es recomendado cuando el volumen de datos es muy grande, ya que tiene una gran carga computacional al momento de ejecutarse la tarea. A continuación, se muestra el gráfico obtenido de estas técnicas, como se observa en la Fig 67.

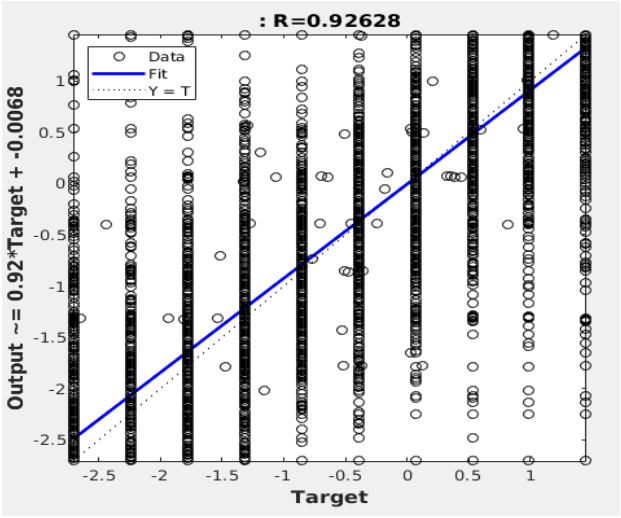


Fig. 67. Gráfico de Regresión para Árboles de Decisiones.

- **Neuro-difuso propuesto:** Los parámetros que utilizamos para la ejecución de este algoritmo fue; 6 reglas, 0.001 como valor inicial de la tasa de aprendizaje y el tamaño del mini-batch fue de 256 (datos por mini-batch), el resultado obtenido fue de $\boldsymbol{R = 0.7963}$ lo cual muestra una fuerza de correlación media, a continuación, se muestra el gráfico de regresión en la Fig. 68.

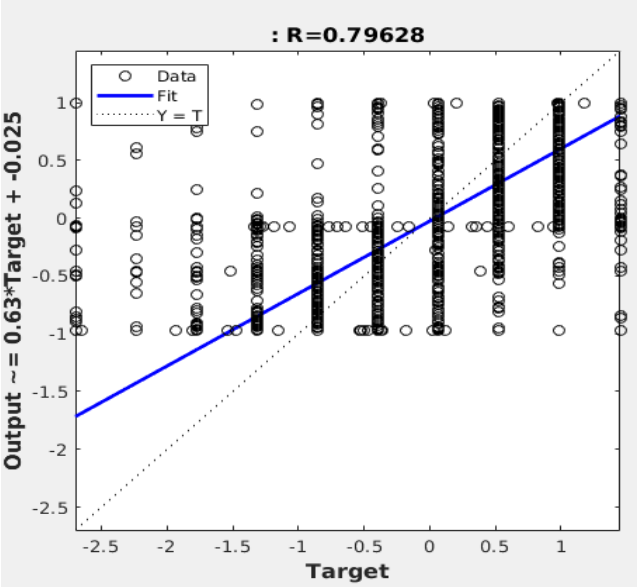


Fig. 698. Gráfico de Regresión para Neuro-Difuso propuesto.

Finalmente, como se observa en la Tabla 14, la técnica que brindó mejor desempeño basado en la fuerza de coeficiente de regresión $\boldsymbol{R}$ fue el de ***Redes Neuronales*** que como bien sabemos es un gran aproximador universal de funciones, y en su proceso de entrenamiento puede aprender cualquier relación entre un conjunto de variables de entrada y salida. Como pudimos observar en los resultados previos, el sistema neuro-difuso propuesto, logro una fuerza de correlación media, quedando por debajo de las técnicas utilizadas para su comparación, sin embargo, el objetivo de esta tesis no es el de mejorar una técnica o crear una nueva, sino, lograr llevar al entorno de datos masivos (Big Data), el procesamiento de estos bajo una arquitectura Neuro-Difusa, aprovechado el poder computacional que ofrece un ecosistema basado en Big Data y las bondades de aprendizaje e interpretabilidad que brinda el modelo.

Tabla 14. Comparativa de resultados de las diferentes técnicas para una tarea de Regresión.

Técnica	Coeficiente de Correlación (R)
Redes Neuronales	0.9562
Árboles de Decisiones	0.9263
Máquina de Soporte de Vectores	0.9185

Sistema Neuro-Difuso propuesto	0.7963

4.2 Conclusiones y trabajos futuros.

A través del desarrollo de esta tesis de investigación, pudimos demostrar que el diseño, construcción e implementación de un Sistema Neuro-Difuso basado en Mamdani con Defusificación en Centro de Conjuntos era posible, ya que después del diseño de su arquitectura y su definición teórica, como primera fase se realizó su implementación sobre escenarios comunes, es decir, procesamientos de datos que bien pudieran ajustarse a los recursos con capacidades limitadas con respecto a memoria y CPUs virtuales principalmente.

En esta primera fase, además de este desarrollo, se buscó experimentar con diferentes formas de procesamiento de datos durante el aprendizaje, evaluando y analizando los resultados que nos brindaba los modos de aprendizaje; *full batch, online y mini-batch*. También este escenario permitió estudiar e identificar cuáles eran los hiper parámetros requeridos para lanzar un entrenamiento adecuado, de los cuales se detectaron que eran; *tasa de aprendizaje, número de reglas y el tamaño del mini-batch*

La segunda fase consistió en la adaptación de esa arquitectura del Sistema Neuro-Difuso propuesto, pero ahora sobre un entorno que tuviera a disposición un motor de procesamiento que permitiera la paralelización de los trabajos, es en esta fase donde el modo de aprendizaje utilizado fue hibrido, pues los datos se dividían en mini-batch los cuales son enviados a los distintos ejecutores para que procesen su entrenamiento de manera independiente, y posteriormente a través de una acción de recolección podamos reunirlos nuevamente y manejar métricas, validaciones y heurísticas como si fuera un full batch.

En esta última fase, además de la tarea de entrenamiento que tuvo que ser paralelizada, se tuvieron que evaluar otros procesos dentro del sistema, potenciales a ser paralelizados, para aprovechar la capacidad que nos ofrece este motor de procesamiento, así que fueron varias tareas de actualización que debieron ser paralelizadas para tener un flujo constante y desempeño adecuado durante el entrenamiento.

Este Sistema Neuro-Difuso propuesto ha sido diseñado para resolver tareas tanto de clasificación como de regresión, y se han comparado sus resultados con respecto a otras técnicas, si bien, nuestro objetivo no era crear un sistema que superara los resultados que técnicas tradicionales ya nos brindan, pudimos demostrar que además de poder realizarse dichas tareas, también podíamos obtener resultados muy adecuados o competitivos.

Finalmente, podemos concluir que el desarrollo de algoritmos inteligentes que se adecuen a las necesidades y retos actuales de los datos que presentan características de*; ambigüedad, alta dimensionalidad, fuentes diversas, ruido,*

volumen, entre otras, al ser tratadas con este tipo de técnicas y en su conjunto con tecnología para su manejo adecuado, representa grandes oportunidades para los tomadores de decisiones que quieran generar estrategias y acciones basados en hechos y de manera informada, ya que los modelos producidos a través de este Sistema Neuro-Difuso propuesto es confiable y altamente interpretable.

Como parte de los trabajos futuros tenemos el poder implementar dicho Sistemas Neuro-Difuso propuesto, en un entorno en la nube, donde además de aprovechar como ya lo hicimos su capacidad de paralelización, también pudiéramos aprovechar su capacidad de distribución sobre un clúster de computadoras virtuales, y entonces poder procesar alto volumen de datos, alta dimensionalidad y realizar integración de datos de distintas fuentes de información.

Apéndice.

A continuación, en la Fig. 48, se muestra un servicio Web local que ofrece Apache Spark, para el monitorear la inicialización del *driver (master) y workers (executors),* y posteriormente, para monitorear la aplicación paralelizada sobre los workers. Se puede observar algunas de las características de hardware antes mencionadas y la asignación de 6 workers, a los cuales de manera individual se les ha otorgado los siguientes recursos; 4 núcleos (cores) y 7 GB de memoria.

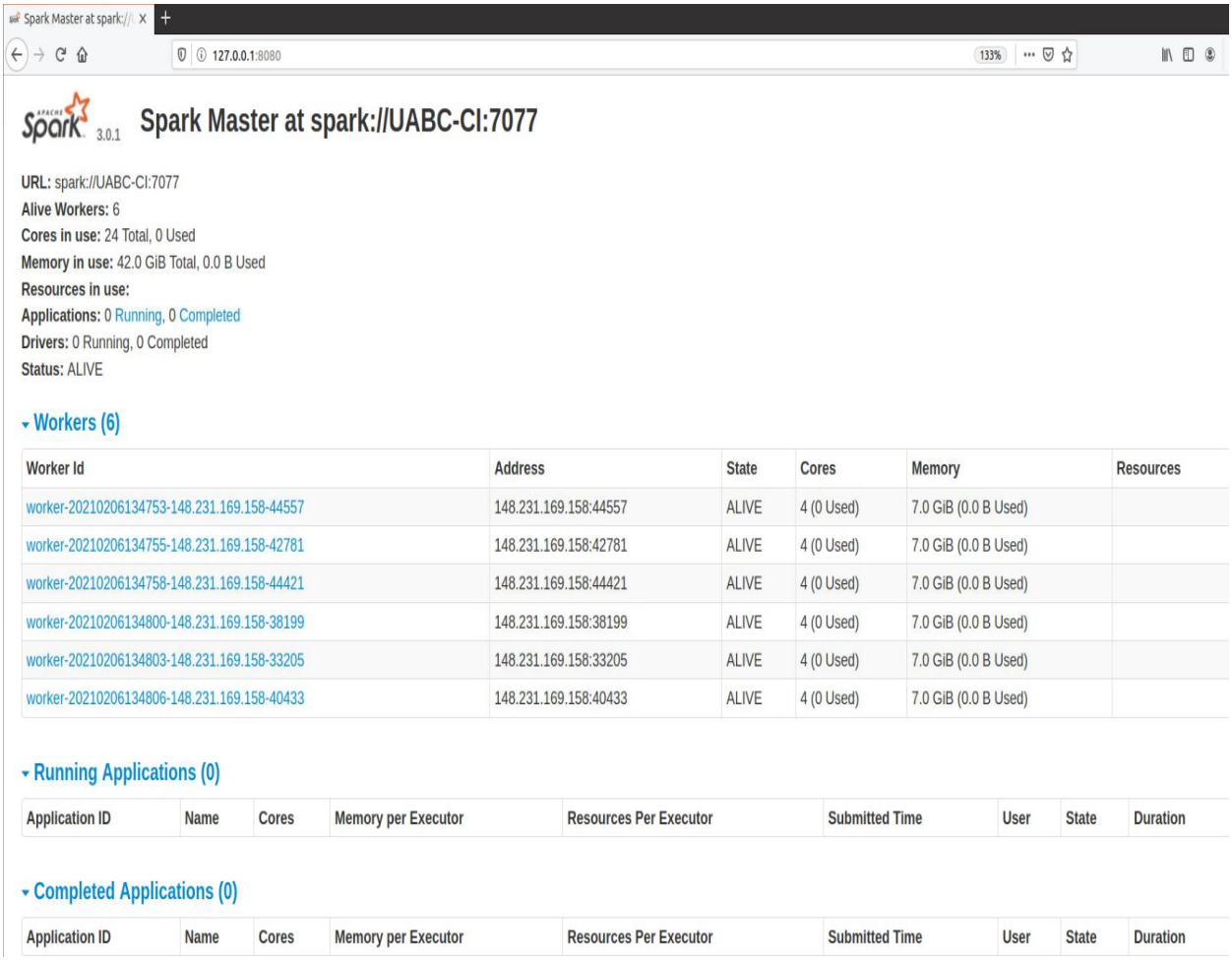


Fig. 70. Servicio web de Spark Standalone para el monitoreo de inicialización de Máster y Workers.

Al hacer el envío de la aplicación al clúster, cambia el estatus del apartado que monitorea la aplicación en ejecución, como se observa en la siguiente Fig. 49.

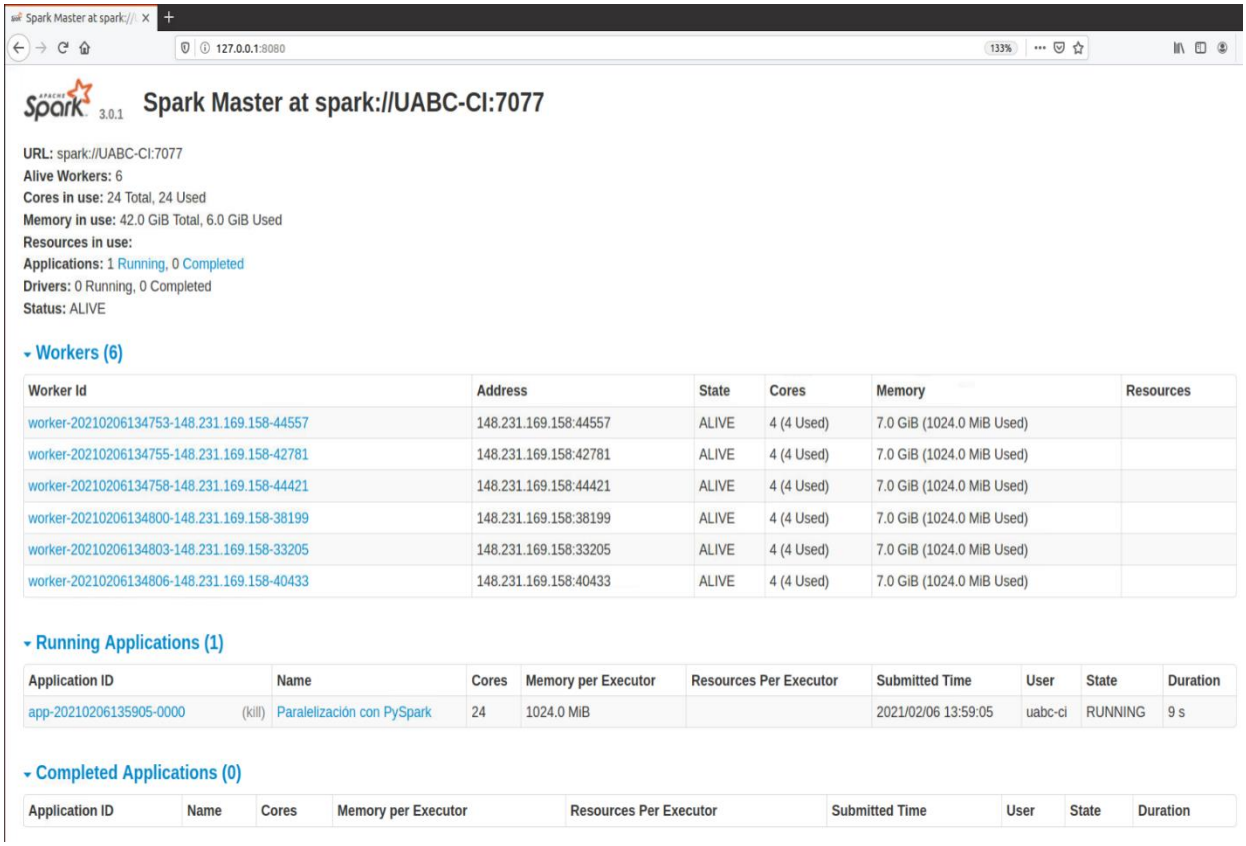


Fig. 71. Servicio web de Spark Standalone para el monitoreo de la ejecución de la Aplicación.

También podemos observar en el monitor del sistema del servidor donde hemos establecido nuestro entorno de desarrollo, como es que los núcleos asignados, están siendo utilizados al 100% en el arranque, como puede verse en la siguiente Fig. 50.

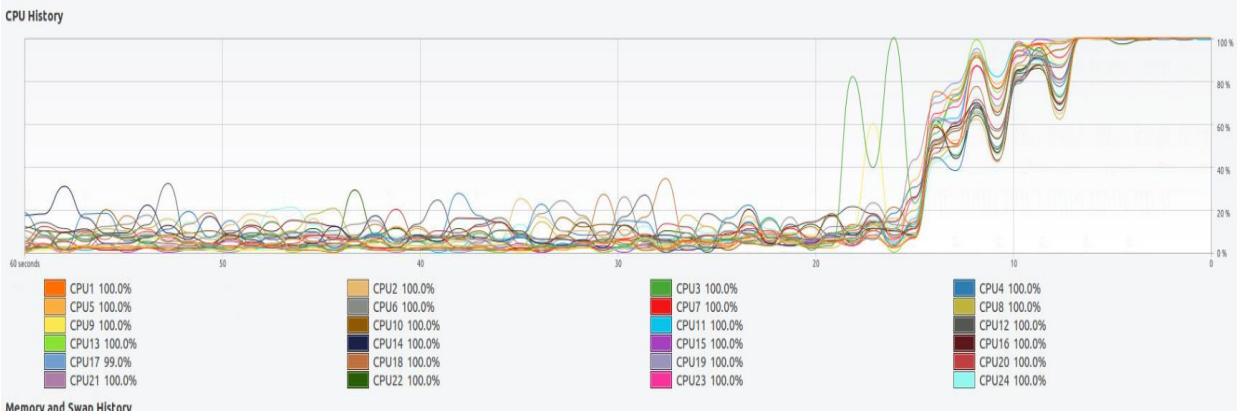


Fig. 72. Sistema de monitoreo del servidor para visualizar la ocupación de los núcleos.

También se puede monitorear el detalle de los *executors* agregados, el avance de las tareas completadas y la acción en ejecución, es decir, la petición collect. Como puede observarse en la Fig. 51.

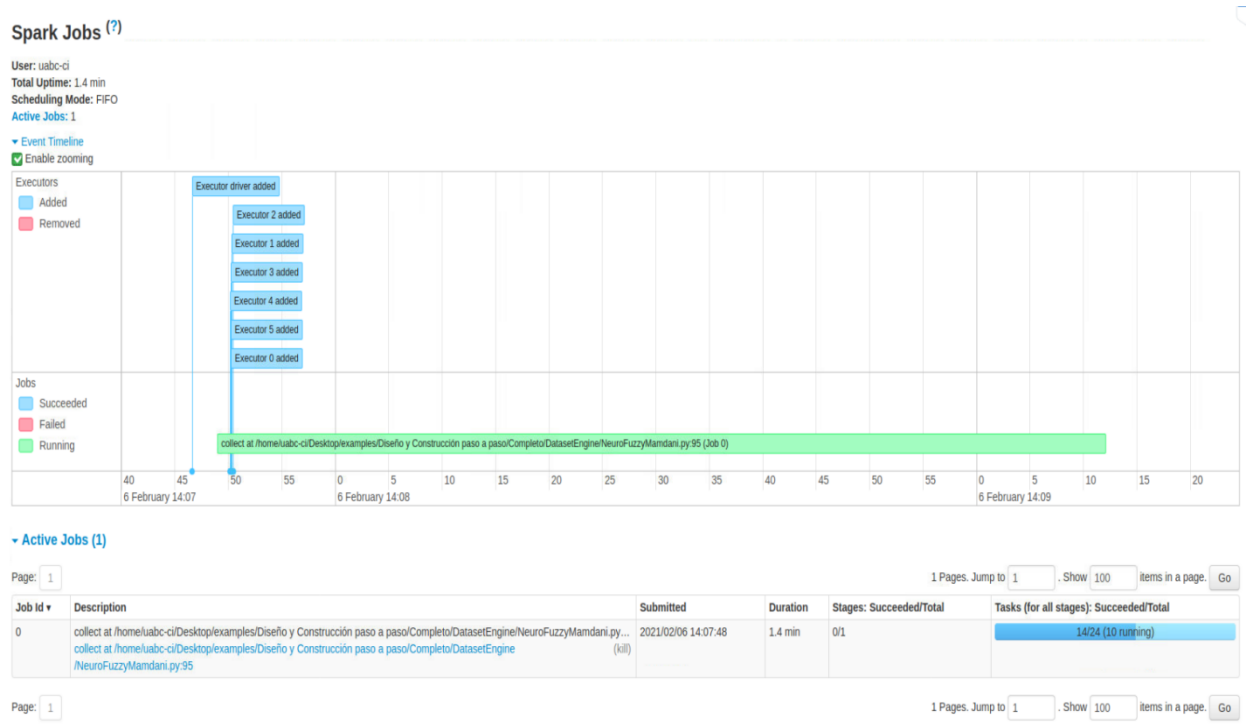


Fig. 73. Servicio web de Spark Standalone para el monitoreo de los ejecutores agregados y las acciones en ejecución.

Y una vez que ha finalizado la ejecución de la aplicación, podemos observar el apartado de aplicación completada en la Fig. 52, donde el estado cambia a finalizado y nos dice el tiempo de duración de toda su ejecución, entre otras cosas.

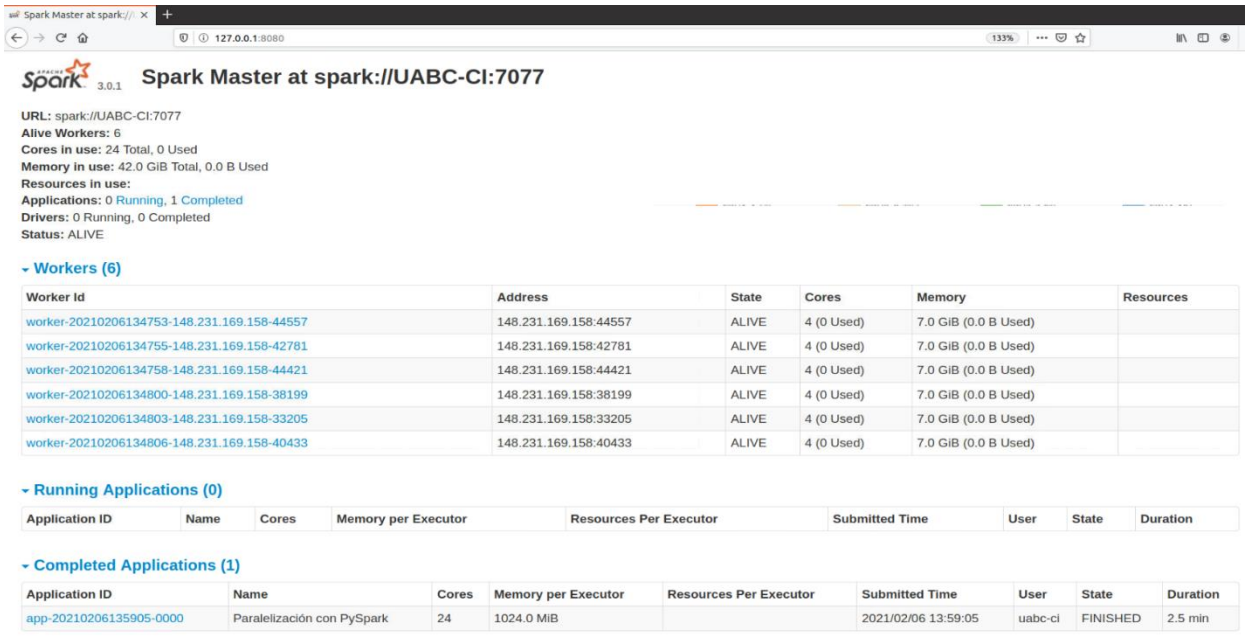


Fig. 74. Servicio web de Spark Standalone para el monitoreo de las aplicaciones completadas.

Para realizar las pruebas en un clúster en la nube, utilizamos Databricks, el cual nos ofrece una plataforma comunitaria con 2 núcleos y 15.3 GB de memoria. Una vez configurado el clúster, agregamos el framework Fuzzy System, el cual fue integrado como una librería de tipo egg. Finalmente abrimos un notebook, y con la librería de Fuzzy System integrada, empezamos a establecer los parámetros requeridos para poder ejecutar las tareas de forma paralelizada y finalmente obtener el resultado de los modelos entrenados.

Como se observa en la Fig. 53, la visualización del DAG nos muestra una etapa (stage) con dos tareas a realizar, la primera es la paralelización y la segunda corresponde a la recolección de las tareas paralelizadas.

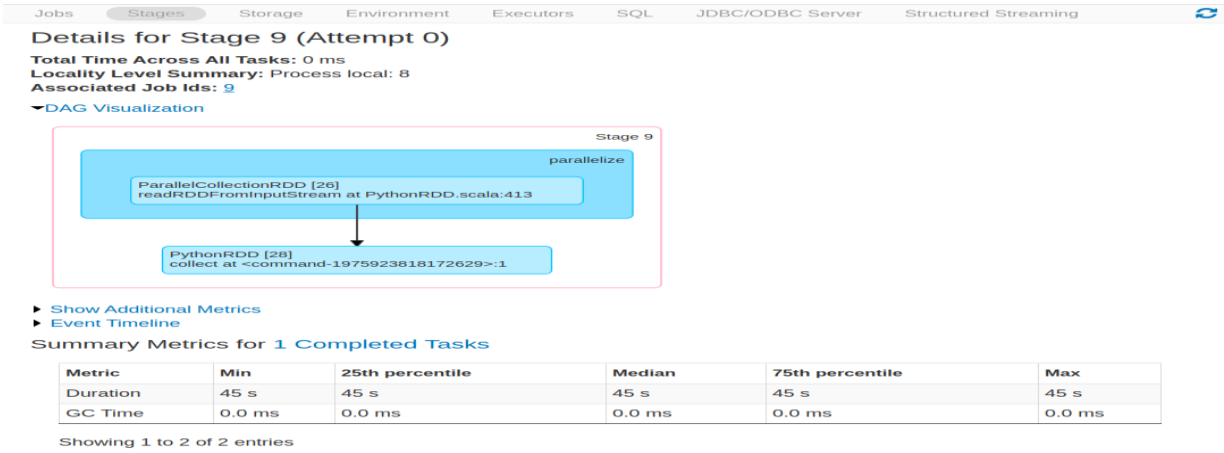


Fig. 75. Visualización del DAG sobre la plataforma Databricks.

También como muestra la Fig 54, se han generado 9 tareas, donde podemos ver cuales están en ejecución o han sido completadas, según va cambiando su estatus.

Tasks (9)

Show 20 entries Search:

Index	Task ID	Attempt	Status	Locality level	Executor ID	Host	Logs	Launch Time	Duration	GC Time	Errors
0	32	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49			
1	33	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49			
2	34	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49			
3	35	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49			
4	36	0	SUCCESS	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49	45 s		
5	37	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49			
6	38	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49			
7	39	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:37:49			
8	40	0	RUNNING	PROCESS_LOCAL	driver	ip-10-172-238-121.us-west-2.compute.internal		2021-02-06 14:38:34			

Fig. 76. Monitoreo de estatus de tareas en ejecución paralelizadas en la plataforma Databricks.

Referencias.

[1] "Quick Start - Spark 3.0.0 Documentation." [Online]. Available: https://spark.apache.org/docs/3.0.0/quick-start.html. [Accessed: 23-Aug-2022].

[2] "Best practices for successfully managing memory for Apache Spark applications on Amazon EMR | AWS Big Data Blog." [Online]. Available: https://aws.amazon.com/es/blogs/big-data/best-practices-for-successfully-managing-memory-for-apache-spark-applications-on-amazon-emr/. [Accessed: 23-Aug-2022].

[3] "Airline Delay and Cancellation Data, 2009 - 2018 | Kaggle." [Online]. Available: https://www.kaggle.com/datasets/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018. [Accessed: 23-Aug-2022].

[4] "UCI Machine Learning Repository: Gas Sensor Array Drift Dataset Data Set," 2022. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Gas+sensor+array+temperature+modulation. [Accessed: 23-Aug-2022].

[5] M. Abramson and H. Wechsler, "Competitive reinforcement learning for combinatorial problems," *Neural Networks, 2001. Proceedings. IJCNN '01. Int. Jt. Conf.*, vol. 4, pp. 2333–2338 vol.4, 2001.

[6] E. Ahmed *et al.*, "The role of big data analytics in Internet of Things," *Comput. Networks*, 2017.

[7] J. Akoka, I. Comyn-Wattaiau, and N. Laoufi, "Research on Big Data – A systematic mapping study," *Comput. Stand. Interfaces*, vol. 54, pp. 105–115, 2017.

[8] O. Alade, A. Selamat, and R. Sallehuddin, "A Review of Advances in Extreme Learning Machine Techniques and Its Applications." pp. 885–895, 2018.

[9] A. Alharthi, V. Krotov, and M. Bowman, "Addressing barriers to big data," *Bus. Horiz.*, 2017.

[10] K. Almejalli, K. Dahal, and A. Hossain, "GA-Based Learning Algorithms to Identify Fuzzy Rules for Fuzzy Neural Networks," 2008, pp. 289–296.

[11] T. G. Babbitt, E. Brynjolfsson, and B. Kahin, "Understanding the Digital Economy: Data Tools, and Research," *Acad. Manag. Rev.*, 2001.

[12] M. I. Baig, L. Shuib, and E. Yadegaridehkordi, "Big data adoption: State of the art and research challenges," *Inf. Process. Manag.*, vol. 56, no. 6, p. 102095, 2019.

[13] L. E. Baum and T. Petrie, "Statistical Inference for Probabilistic Functions of Finite State Markov Chains," *Ann. Math. Stat.*, vol. 37, no. 6, pp. 1554–1563, 1966.

[14] C. Biryulev, Y. Yakymiv, and A. Selemonavichus, "Research of ANN usage in Data Mining and Semantic Integration," *MEMSTECH'2010*, 2010.

[15] Y. Bodyanskiy, “Computational Intelligence Techniques for Data Analysis,” pp. 15–36.

[16] V. Chaorasiya and A. Shrivastava, “A survey on Big Data : Techniques and Technologies,” *Int. J. Res. Dev. Appl. Sci. Eng.*, vol. 8, no. 1, 2015.

[17] M. Chen, S. Mao, and Y. Liu, “Big data: A survey,” in *Mobile Networks and Applications*, 2014.

[18] C. Cortes and V. Vapnik, “Support-Vector Networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.

[19] S. Desai, K. Joshi, and B. Desai, “Survey on Reinforcement Learning Techniques,” *Int. J. Sci. Res. Publ.*, vol. 6, no. 2, pp. 179–2250, 2016.

[20] A. C. Eberendu, “Unstructured Data: an overview of the data of Big Data,” *Int. J. Comput. Trends Technol.*, 2016.

[21] M. M. Elmetwally, F. A. Aal, M. L. Awad, and S. Omran, “A hopfield neural network approach for integrated transmission network expansion planning,” in *Proceedings of the 11th International Middle East Power Systems Conference, MEPCON’2006*, 2006.

[22] A. P. Engelbrecht, *Computational Intelligence*. 2007.

[23] B. Erar, “Mixture model cluster analysis under different covariance structures using information complexity,” 2011.

[24] A. Fernández, C. J. Carmona, M. J. del Jesus, and F. Herrera, “A View on Fuzzy Systems for Big Data: Progress and Opportunities,” *Int. J. Comput. Intell. Syst.*, 2016.

[25] A. Gandomi and M. Haider, “Beyond the hype: Big data concepts, methods, and analytics,” *Int. J. Inf. Manage.*, 2015.

[26] D. Helbing, *Thinking ahead-essays on big data, digital revolution, and participatory market society*. 2015.

[27] R. Hill, “Computational intelligence and emerging data technologies,” in *Proceedings - 2nd International Conference on Intelligent Networking and Collaborative Systems, INCOS 2010*, 2010, pp. 449–454.

[28] S. Horwitz, “Wal-Mart to the rescue private enterprise’s response to Hurricane Katrina,” *Indep. Rev.*, 2009.

[29] L. C. Jain, M. Seera, C. P. Lim, and P. Balasubramaniam, “A review of online learning in supervised neural networks,” *Neural Computing and Applications*, vol. 25, no. 3–4. pp. 491–509, 2014.

[30] S. Jain, “Mining Big Data using Genetic Algorithm,” *Int. Res. J. Eng. Technol.*, vol. 4, no. 7, pp. 743–747, 2017.

[31] J. S. R. Jang, C. T. Sun, and E. Mizutani, “Neuro-Fuzzy and Soft Computing-A Computational Approach to Learning and Machine Intelligence [Book Review],” *IEEE Trans. Automat. Contr.*, 2005.

[32] X. Jin, B. W. Wah, X. Cheng, and Y. Wang, “Significance and Challenges of Big Data Research,” *Big Data Res.*, 2015.

[33] C. Ka Yuk Chan *et al.*, “Learning in Artificial Neural Networks,” in *Encyclopedia of the Sciences of Learning*, Boston, MA: Springer US, 2012, pp. 1893–1898.

[34] C. Kacfah Emani, N. Cullot, and C. Nicolle, “Understandable Big Data: A survey,” *Computer Science Review*. 2015.

[35] A. K. Kar, “Bio inspired computing - A review of algorithms and scope of applications,” *Expert Systems with Applications*. 2016.

[36] A. K. Kar, “Bio inspired computing - A review of algorithms and scope of applications,” *Expert Syst. Appl.*, vol. 59, pp. 20–32, 2016.

[37] R. Kruse, C. Borgelt, C. Braune, F. Klawonn, C. Moewes, and M. Steinbrecher, *Computational Intelligence: Eine methodische Einführung in Künstliche Neuronale Netze, Evolutionäre Algorithmen, Fuzzy-Systeme und Bayes-Netze*. 2015.

[38] E. P. Kumar and E. P. Sharma, “Artificial Neural Networks-A Study,” *Int. J. Emerg. Eng. Res. Technol.*, vol. 2, no. 2, pp. 143–148, 2014.

[39] D. Laney, “3D Data Management: Controlling Data Volume, Velocity, and Variety.,” *Appl. Deliv. Strateg.*, 2001.

[40] Y. A. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” in *Neural networks: Tricks of the trade*, Springer, 2012, pp. 9–48.

[41] I. Lee, “Big data: Dimensions, evolution, impacts, and challenges,” *Bus. Horiz.*, 2017.

[42] A. L’Heureux, K. Grolinger, H. F. Elyamany, and M. A. M. Capretz, “Machine Learning with Big Data: Challenges and Approaches,” *IEEE Access*, 2017.

[43] X.-M. Li and J.-H. Ouyang, “Tuning the Learning Rate for Stochastic Variational Inference,” *J. Comput. Sci. Technol.*, vol. 31, no. 2, pp. 428–436, 2016.

[44] G. Maheswaran, “K Means Clustering Algorithms : A Comparitive Study,” 2012.

[45] J. Manyika, M. Chui Brown, B. B. J., R. Dobbs, C. Roxburgh, and A. Hung Byers, “Big data: The next frontier for innovation, competition and productivity,” *McKinsey Glob. Inst.*, 2011.

[46] P. T. Metaxas, E. Mustafaraj, and D. Gayo-Avello, “The Parable of Google Flu: Traps in Big Data Analysis The Parable of Google Flu: Traps in Big Data Analysis,” in *Proceedings - 2011*

IEEE International Conference on Privacy, Security, Risk and Trust and IEEE International Conference on Social Computing, PASSAT/SocialCom 2011, 2011.

[47] M. Mitchell, "Genetic algorithms: An overview," *Complexity*, vol. 1, no. 1, pp. 31–39, 1995.

[48] R. Navarro, "GitHub - Raul-Navarro/fuzzy-framework: Framework to build fuzzy inference systems." [Online]. Available: https://github.com/Raul-Navarro/fuzzy-framework. [Accessed: 23-Aug-2022].

[49] M. Negnevitsky, "Artificial intelligence: a guide to intelligent systems. Pearson Education.," in *Artificial intelligence: a guide to intelligent systems*, 2005.

[50] M. Niño and A. Illarramendi, "ENTENDIENDO EL BIG DATA: ANTECEDENTES, ORIGEN Y DESARROLLO POSTERIOR," *DYNA NEW Technol.*, 2015.

[51] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," *ArXiv e-prints*, 2015.

[52] A. Oussous, F. Z. Benjelloun, A. Ait Lahcen, and S. Belfkih, "Big Data technologies: A survey," *Journal of King Saud University - Computer and Information Sciences*. 2018.

[53] S. K. Pal, S. K. Meher, and A. Skowron, "Data science, big data and granular mining," *Pattern Recognit. Lett.*, vol. 67, pp. 109–112, 2015.

[54] D. Pandey and P. Pandey, "Approximate Q-Learning: An Introduction," in *2010 Second International Conference on Machine Learning and Computing*, 2010, pp. 317–320.

[55] J. S. Park, H. G. Kim, D. G. Kim, I. J. Yu, and H. K. Lee, "Paired mini-batch training: A new deep network training for image forensics and steganalysis," *Signal Process. Image Commun.*, vol. 67, pp. 132–139, Sep. 2018.

[56] D. Partouche, M. Pasquier, and A. Spalanzani, "Intelligent Speed Adaptation Using a Self-Organizing Neuro-Fuzzy Controller," *2007 IEEE Intell. Veh. Symp.*, pp. 846–851, Jun. 2007.

[57] R. Patgiri and A. Ahmed, "Big Data: The V's of the Game Changer Paradigm," in *Proceedings - 18th IEEE International Conference on High Performance Computing and Communications, 14th IEEE International Conference on Smart City and 2nd IEEE International Conference on Data Science and Systems, HPCC/SmartCity/DSS 2016*, 2017.

[58] D. Pelleg, D. Pelleg, A. W. Moore, and A. W. Moore, "X-means: Extending K-means with efficient estimation of the number of clusters," *Proc. Seventeenth Int. Conf. Mach. Learn. table contents*, pp. 727–734, 2000.

[59] A. Petrasova, "Chapter 1 Introduction," in *Tangible Modeling with Open Source GIS*, 2015, pp. 1–15.

[60] C. L. Philip Chen and C. Y. Zhang, “Data-intensive applications, challenges, techniques and technologies: A survey on Big Data,” *Inf. Sci. (Ny).*, 2014.

[61] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, “A survey of machine learning for big data processing,” *Eurasip Journal on Advances in Signal Processing*. 2016.

[62] J. Rifkin, *The third industrial revolution : how lateral power is transforming energy, the economy, and the world / Jeremy Rifkin*. 2011.

[63] I. Rish, “An Empirical Study of the Naïve Bayes Classifier,” *IJCAI 2001 Work Empir Methods Artif Intell*, vol. 3, 2001.

[64] L. Rodríguez-Mazahua, C. A. Rodríguez-Enríquez, J. L. Sánchez-Cervantes, J. Cervantes, J. L. García-Alcaraz, and G. Alor-Hernández, “A general perspective of Big Data: applications, tools, challenges and trends,” *J. Supercomput.*, 2016.

[65] T. J. Ross *et al.*, “Fuzzy Logic with Engineering Applications,” *IEEE Trans. Inf. Theory*, 2004.

[66] M. A. Sanchez, O. Castillo, and J. R. Castro, “An Overview of Granular Computing Using Fuzzy Logic Systems,” vol. 667, Springer Verlag, 2017, pp. 19–38.

[67] U. Sivarajah, M. M. Kamal, Z. Irani, and V. Weerakkody, “Critical analysis of Big Data challenges and analytical methods,” *J. Bus. Res.*, vol. 70, pp. 263–286, Jan. 2017.

[68] P. Smets and P. Magrez, “Implication in fuzzy logic,” *Int. J. Approx. Reason.*, vol. 1, no. 4, pp. 327–347, Oct. 1987.

[69] P. Sridhar and N. Dharmaji, “A Comparative Study on How Big Data is Scaling Business Intelligence and Analytics,” *Int. J. Enhanc. Res. Sci. Technol. Eng.*, 2013.

[70] J. R. Thomson, *High Integrity Systems and Safety Management in Hazardous Industries*. 2015.

[71] C. Y. Wang and L. Wan, “Type-2 fuzzy implications and fuzzy-valued approximation reasoning,” *Int. J. Approx. Reason.*, vol. 102, pp. 108–122, Nov. 2018.

[72] H. Wang, Z. Xu, and W. Pedrycz, “An overview on the roles of fuzzy set techniques in big data processing: Trends, challenges and opportunities,” *Knowledge-Based Syst.*, 2017.

[73] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical Machine Learning Tools and Techniques*. 2016.

[74] Y. Yao, “Human-Inspired Granular Computing,” 2007.

[75] M. Ying, “Implication operators in fuzzy logic,” *IEEE Trans. Fuzzy Syst.*, vol. 10, no. 1, pp. 88–91, 2002.

[76] P. Zarikas, Vasilios and Papageorgiou, Elpiniki and Regner, “Bayesian network construction using a fuzzy rule based approach for medical decision support,” *Expert Syst.*, vol. 32, pp. 344–369, 2015.

[77] L. Zhou, S. Pan, J. Wang, and A. V Vasilakos, “Machine learning on big data: Opportunities and challenges,” *Neurocomputing*, vol. 237, pp. 350–361, 2017.

[78] B. Nagy, R. Basbous, and T. Tajti, “Lazy evaluations in Lukasiewicz type fuzzy logic,” *Fuzzy Sets Syst.*, Nov. 2018.