Facultad de Ingeniería Informática
Universidad Tecnológica de La Habana
José Antonio Echeverría
cujae

Informe de prácticas profesionales

Modificación de componente KNIME de AutoML en tareas de clasificación.

Autores:
Jennifer Yanez Jiménez
Rainer Pellerano Alvarez

Tutora:
Dra. Raisa Socorro Llanes

La Habana, Abril 2023

Resumen

Vivimos en un mundo en el que se generan grandes cantidades de datos almacenados en diferentes sistemas, y el reto es convertir esos datos en información útil para la toma de decisiones. Una técnica para extraer información valiosa de grandes cantidades de datos es el Aprendizaje Automático, que se enfoca en el desarrollo de modelos y algoritmos que permiten a las computadoras aprender de los datos sin ser programadas explícitamente para hacerlo. Para implementar efectivamente estas técnicas, se requiere de la intervención humana, y la automatización del aprendizaje automático (AutoML) se ha desarrollado como una solución para simplificar y acelerar este proceso. Para ello existen numerosas herramientas, como KNIME, que permite la implementación de AutoML a través de diferentes nodos. En (Carrazana Ruiz, 2022) se desarrolla un componente dedicado a esta tarea, específicamente para el pre-procesado en tareas de clasificación. No obstante, quedaron tareas pendientes, como la optimización de hiperparámetros y la automatización de tareas en la fase de transformación de los datos, las cuales son implementadas en este trabajo.

Palabras clave: Aprendizaje Automático, Minería de datos, AutoML, KNIME, optimización de hiperparámetros, preprocesamiento de datos, clasificación.

Abstract

We live in a world in which large amounts of data stored in different systems are generated, and the challenge is to convert this data into useful information for decision making. One technique for extracting valuable information from large amounts of data is Machine Learning, which focuses on developing models and algorithms that allow computers to learn from data without being cleanly programmed to do so. To effectively implement these techniques, human intervention is required, and Automation of Machine Learning (AutoML) has been developed as a solution to simplify and speed up this process. For this, there are numerous tools, such as KNIME, that allow the implementation of AutoML through different nodes. (Carrazana Ruiz, 2022) develops a component dedicated to this task, specifically for preprocessing in classification tasks. However, there were pending tasks, such as the optimization of hyperparameters and the automation of tasks in the data transformation phase, which are implemented in this work.

Keywords: Machine Learning, Data Mining, AutoML, KNIME, HPO, Data pre-processing, classification

Índice general

Índice de tablas

Índice de figuras

Introducción

En la actualidad, vivimos en un mundo donde cada vez más datos se generan y almacenan en diversos tipos de sistemas, lo que nos presenta un gran desafío: ¿cómo convertir estos datos en información valiosa que pueda ser utilizada para tomar decisiones informadas? Para resolver este reto, se han desarrollado diversas técnicas y herramientas para extraer información útil de grandes cantidades de datos. Uno de estos enfoques es el Aprendizaje Automático.

El Aprendizaje Automático, mayormente conocido como Machine Learning, es un subcampo de la Inteligencia Artificial, que se enfoca en el desarrollo de algoritmos y modelos que permiten a las computadoras aprender de los datos sin ser programadas explícitamente para hacerlo. Se define como un conjunto de métodos que puede detectar patrones en los datos automáticamente, y luego usar los patrones descubiertos para predecir datos en el futuro, o para realizar otro tipo de toma de decisiones bajo incertidumbre (Murphy, 2012). Estos patrones interesantes son los que representan el conocimiento. Sin embargo, el proceso de aprendizaje automático requiere de un gran volumen de datos y, a menudo, estos datos no están estructurados. Es aquí donde la minería de datos entra en juego.

La minería de datos es el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos almacenados en distintos formatos (Hernández Orallo *et al.*, 2004). Como parte del proceso de descubrimiento de conocimiento en los datos (KDD, por sus siglas en inglés), involucra las fases de dicho proceso: integración, recopilación, selección, limpieza, transformación, evaluación e interpretación de los datos; así como la difusión y uso del conocimiento obtenido (Jiawei Han, 2011).

La implementación efectiva de estas técnicas requiere la intervención humana, incluyendo la selección de algoritmos adecuados, el preprocesamiento de datos y la optimización de hiperparámetros. La automatización del aprendizaje automático (AutoML) se ha desarrollado como una solución para simplificar y acelerar este proceso. AutoML tiene como objetivo tomar estas decisiones de una manera automatizada, objetiva y basada en datos: el usuario simplemente proporciona datos y el sistema AutoML determina automáticamente el enfoque que funciona mejor para esta aplicación en particular (Hutter *et al.*, 2019). Para realizar la minería de datos de manera efectiva, se requiere de herramientas especializadas que permitan procesar grandes cantidades de información de forma rápida y eficiente. Una de estas herramientas es KNIME.

KNIME es una herramienta popular que proporciona un entorno de desarrollo visual para la crea-

ción, ejecución y evaluación de flujos de trabajo de análisis de datos. Es una plataforma de software libre y abierto que incluye una amplia variedad de nodos para el preprocesado de datos, la minería de datos y la modelización de datos (kni, 2023). En KNIME, el AutoML se puede implementar a través de diferentes nodos, uno de estos es, por ejemplo, el nodo *AutoML Learner*, que permite seleccionar automáticamente el mejor algoritmo de aprendizaje automático para un conjunto de datos en particular, así como ajustar los hiperparámetros del modelo. No obstante, una de las principales desventajas que presenta es la falta de control y personalización. Aunque el nodo AutoML permite a los usuarios seleccionar automáticamente los algoritmos y ajustar los hiperparámetros, los usuarios no tienen control directo sobre estos procesos y no pueden ajustar los parámetros de manera manual. Esto puede limitar la capacidad de los usuarios para personalizar y optimizar los modelos para satisfacer sus necesidades específicas.
En (Carrazana Ruiz, 2022) se desarrolla un componente de KNIME, que ejecuta y compara el desempeño de múltiples flujos de AutoML en tareas de clasificación, partiendo de la problemática de que no es posible, mediante un único nodo, desarrollar todo el proceso de KDD con la posibilidad de visualizar y analizar las consideraciones realizadas durante el pre-procesado. En este componente se desarrollaron subcomponentes enfocados en tareas concisas de pre-procesado: el procesamiento de datos numéricos, *string*, valores faltantes y el ajuste de tipos columna; y se demostró el correcto funcionamiento del componente propuesto y los subcomponentes que lo integran. Sin embargo, quedaron tareas pendientes, como la optimización de hiperparámetros y la automatización de algunas de las actividades esenciales en el pre-procesado, siendo esta la **situación problemática**.
Se define como **problema a resolver** ¿cómo incorporar a este componente la posibilidad de optimizar los hiperparámetros y algunas de las actividades iniciales en el pre-procesado a partir de un algoritmo de clasificación? Para dar solución a esta problemática, se determina como **objetivo general** desarrollar una nueva versión del componente KNIME que permita la selección de manera automatizada de valores e hiperparámetros predefinidos en la implementación anterior. Este objetivo general se desglosa en los siguientes **objetivos específicos y tareas**:

1. Analizar el estado del arte de Machine Learning, AutoML y KNIME.
    1.1 Investigar las técnicas de Machine Learning.
    1.2 Investigar los enfoques de AutoML.
    1.3 Investigar la implementación e inclusión de nuevos componentes en KNIME.
2. Modificar múltiples flujos de AutoML en tareas de Clasificación.
    2.1 Desarrollar los subcomponentes KNIME que implementen los flujos de AutoML de pre-procesado.
    2.2 Desarrollar los subcomponentes KNIME que implementen los flujos de AutoML en optimización de hiperparámetros.

2.3 Desarrollar componentes KNIME para la visualización de los resultados.

3. Realizar la validación de cada flujo implementado.

3.1 Diseñar y ejecutar las pruebas y experimentos de los flujos implementados.

3.2 Documentar los resultados obtenidos.

El **valor práctico** de este proyecto consiste en las modificaciones desarrolladas al componente KNIME de AutoML para el pre-procesado en tareas de clasificación, capaz de automatizar la optimización de hiperparámetros y ejecutar flujos de pre-procesado para diferentes algoritmos en dicha tarea.
En cuanto a la estructuración, este trabajo está dividido en tres capítulos:

- **Capítulo 1: Aprendizaje automático y AutoML**, se presenta el estudio realizado sobre las temáticas que aborda el trabajo, se muestran los conceptos fundamentales relacionados con el Aprendizaje automático, la Minería de Datos y AutoML; así como la descripción del componente KNIME de AutoML para pre-procesado.
- **Capítulo 2: Propuesta de modificación al componente KNIME de AutoML para pre-procesado**, se presenta y expone el diseño e implementación de la modificación al componente KNIME para tareas de AutoML en pre-procesado y optimización de hiperparámetros.
- **Capítulo 3: Integración y validación de soluciones propuestas al componente de AutoML**, se muestran, comparan y analizan los resultados obtenidos de los algoritmos para diferentes configuraciones.

Capítulo 1

Aprendizaje Automático y *AutoML*

En este primer capítulo, se aborda el marco teórico de la tesis, el cual se enfoca en diferentes temas clave dentro del campo de la inteligencia artificial y la minería de datos. En particular, se discute el aprendizaje automático, el descubrimiento de conocimiento en bases de datos, la minería de datos y la clasificación. Además, se introduce el concepto de AutoML, como herramienta para automatizar el proceso de preprocesado y optimización de hiperparámetros en la implementación de técnicas de aprendizaje automático. Por último, se destaca la importancia de la plataforma KNIME como una herramienta útil para la implementación de técnicas de AutoML y análisis de datos.

1.1. Aprendizaje Automático

Uno de los campos más destacados dentro de la IA es el Machine Learning (aprendizaje automático), que es un enfoque que utiliza algoritmos y modelos matemáticos para permitir que los sistemas aprendan de los datos y realicen tareas específicas sin ser programados explícitamente. Se define el Aprendizaje Automático como un conjunto de métodos que pueden detectar automáticamente patrones en los datos y luego, usar los patrones descubiertos para predecir datos futuros, o para realizar otros tipos de toma de decisiones bajo incertidumbre (Murphy, 2012). Es decir, es el proceso en el que las computadoras descubren cómo hacer cosas sin estar específicamente programadas para hacerlo (R *et al.*, 2021). Por lo tanto, el objetivo principal del aprendizaje automático es estudiar, diseñar y mejorar modelos matemáticos que se pueden entrenar (una vez o continuamente) con datos relacionados con el contexto (proporcionados por un entorno genérico), para inferir el futuro y tomar decisiones sin completo conocimiento de todos los elementos que influyen (factores externos) (Bonaccorso, 2017).
Existen varios tipos de aprendizaje en Machine Learning, cada uno con sus propias técnicas y enfoques. A continuación, se presenta una breve descripción de algunos de los tipos de aprendizaje más comunes:

- Aprendizaje supervisado: se refiere a cualquier proceso de aprendizaje automático que aprende de una función de un tipo de entrada a un tipo de salida, utilizando datos que comprenden ejemplos que tienen valores de entrada y salida. Dos ejemplos típicos de aprendizaje supervisado son el aprendizaje de clasificación y la regresión (Sammut & Webb, 2011).

- Aprendizaje no supervisado: se refiere a cualquier proceso de aprendizaje automático que busca aprender la estructura en ausencia de un resultado identificado o retroalimentación. Tres ejemplos típicos de aprendizaje no supervisado son agrupamiento, reglas de asociación y mapas de autoorganización (Sammut & Webb, 2011).

- Aprendizaje por refuerzo: describe una gran clase de problemas de aprendizaje, característicos de los agentes autónomos que interactúan en un entorno: problemas de toma de decisiones secuenciales con recompensa retrasada. Los algoritmos de aprendizaje por refuerzo buscan aprender una política (mapeo de estados a acciones) que maximice la recompensa recibida a lo largo del tiempo. A diferencia de los problemas de aprendizaje supervisado, en los problemas de aprendizaje por refuerzo no hay etiquetas de ejemplo de comportamiento correcto e incorrecto. Sin embargo, a diferencia de los problemas de aprendizaje no supervisados, se puede percibir una señal de recompensa (Sammut & Webb, 2011).

- Aprendizaje semisupervisado: es una clase de técnicas de aprendizaje automático que hacen uso de ejemplos etiquetados y no etiquetados para aprender un modelo. Los ejemplos etiquetados se usan para aprender modelos de clase y los ejemplos no etiquetados se usan para refinar los límites entre clases (Jiawei Han, 2011).

En resumen, Machine Learning es una técnica que permite a las computadoras aprender patrones a partir de datos, con el objetivo de realizar predicciones o clasificaciones precisas en datos nuevos. Sin embargo, antes de aplicar técnicas de Machine Learning a un conjunto de datos, es necesario realizar una serie de procesos previos, como la selección y preprocesamiento de datos, selección de técnicas de aprendizaje adecuadas y evaluación y ajuste de modelos. De esta manera, el proceso de descubrimiento de conocimiento en bases de datos complementa al Machine Learning, permitiendo una exploración más completa de los datos y una selección más adecuada de las técnicas de aprendizaje.

1.2. Proceso de descubrimiento de conocimiento en bases de datos

La Extracción de Conocimiento en Bases de Datos (*Knwodlege Discovery from Databases*, o *KDD* por sus siglas en inglés) se define como: “el proceso no trivial de identificar patrones válidos, novedosos, potencialmente útiles y, en última instancia, comprensibles a partir de los

datos" (Hernández Orallo *et al.*, 2004). Este proceso está compuesto por una serie de etapas o fases, descritas a continuación:

- Integración y recopilación: Se requiere poseer todos los datos que sean de utilidad a partir de las necesidades de la organización. Determina las fuentes de información que pueden ser útiles y dónde conseguirlas. Como parte del desarrollo de esta fase es necesario diseñar o conocer el modo en que se van a organizar e integrar los datos; con el fin de eliminar redundancias e inconsistencias.

- Selección, limpieza y transformación: Se seleccionan los datos más relevantes y que aporten mejor información, garantizando que el dato tenga la mejor calidad posible logrando obtener las vistas minables, con los datos listos para la aplicación del algoritmo.

- Algoritmos de Minería de datos: A través de la vista minable obtenida en la fase anterior se aplican los algoritmos de extracción del conocimiento.

- Evaluación e Interpretación: El objetivo de esta etapa es medir la calidad de los modelos obtenidos utilizando diferentes métricas de calidad. Las cuales dependen de las técnicas de minería de datos que se utilicen. La interpretación de los resultados se apoya en el uso de técnicas de visualización y de representación con el fin de entender mejor el conocimiento aportado.

- Difusión y uso: En esta etapa, se integra el conocimiento obtenido de la comprensión del negocio, con el conocimiento de los modelos de minería de datos usado en la toma de decisiones de los especialistas. La monitorización de los patrones debe realizarse, pues en ocasiones resulta necesaria la reevaluación del modelo, su reentrenamiento o incluso su reconstrucción total.

Tras el aumento de grandes volúmenes de información en toda institución donde se almacenen datos históricos, ahora informatizados en bases de datos digitales, es necesaria la extracción de información valiosa a través de patrones ocultos en estos datos. Sin embargo, esta cantidad de información sobrepasa las capacidades de los analistas de estas instituciones, provocando la necesidad del empleo de técnicas automatizadas. De aquí, surge como una nueva necesidad la minería de datos, siendo parte del proceso KDD.

1.3. Minería de Datos

Acorde a (Hernández Orallo *et al.*, 2004), la minería de datos es definida como el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos almacenados en distintos formatos.
El conocimiento extraído se puede presentar en forma de relaciones, patrones o reglas inferidos de

los datos, o en forma de descripción un poco más concisa. Estos constituyen un modelo de datos analizados. Estos modelos, o tareas, se categorizan en predictivas y descriptivas (Hernández Orallo *et al.*, 2004).
En las tareas predictivas, los ejemplos están etiquetados y se emplean para estimar valores futuros o desconocidos de variables de interés. En este entorno se encuentra el aprendizaje supervisado. En cambio, las tareas descriptivas son empleadas en el descubrimiento de propiedades de los datos examinados donde los ejemplos no se encuentran etiquetados. Aquí se pone de manifiesto el aprendizaje no supervisado. En (Hernández Orallo *et al.*, 2004) se describen las tareas de minería de datos como sigue:

- Clasificación: La clasificación se encarga de examinar las características de un registro u objeto, y de esta forma asignarle una clase predefinida. Estas clases son valores discretos. Para ello, se tiene que construir un modelo a partir de datos previamente clasificados. Como variantes a la clasificación, existe el aprendizaje de "rankings", aprendizaje de preferencias y el aprendizaje de probabilidad, entre otros.

- Regresión: A diferencia de la clasificación, el valor a predecir es numérico. Consiste en aprender una función real que calcula un valor para un atributo real. Su objetivo es minimizar el error entre el valor predicho y el valor real.

- Correlaciones: Son empleadas para examinar el grado de similitud de los valores de dos variables numéricas. Se basa en el cálculo de correlación de variables numéricas usando la estadística. Este método trata de determinar si el comportamiento de dos variables numéricas está relacionado.

- Reglas de asociación: Las reglas de asociación son situaciones o características que ocurren en un mismo instante de tiempo. Pueden ser relaciones causales o casuales. Representan patrones de comportamiento entre los datos en función de la aparición conjunta de valores de dos o más atributos. Las medidas habituales propuestas en (Agrawal *et al.*, 1993) para establecer la idoneidad y el interés de una regla de asociación son la confianza y el soporte.

- Agrupamiento (Clustering): Para realizar esta tarea se parte de datos sin clasificar, teniendo como objetivo segmentar un grupo de datos diversos en subgrupos. Los miembros de cada grupo (clúster, por su definición en inglés) deben tener mucho en común entre sí y, a su vez, diferenciarse del resto de elementos de otros grupos. Dado que la clasificación de estos grupos no se conoce previamente, es el minero el encargado de darles un significado.

1.3.1. Clasificación

En el uso común, la palabra clasificación significa colocar las cosas en categorías, agruparlas de alguna manera útil. El ser humano generalmente hace esto porque las cosas en un grupo, llama-

do *clase* en aprendizaje automático, comparten características comunes (Sammut & Webb, 2011). En aprendizaje automático, la clasificación se utiliza para identificar a qué clase o categoría pertenece una determinada observación o registro, basándose en un conjunto de características o variables. En esta tarea, se utiliza un algoritmo para construir un modelo predictivo que asigne una etiqueta de clase a cada observación en función de sus características. Este modelo se entrena utilizando un conjunto de datos etiquetados previamente, y luego se aplica a nuevos datos para hacer predicciones (Sammut & Webb, 2011).

Clasificación con Árboles de Decisión

La clasificación con árboles de decisión es un método popular en la minería de datos y en el aprendizaje automático supervisado, utilizado para predecir la clase o categoría de un objeto o registro. Los Árboles de Decisión son unos de los modelos más populares, su representación es de fácil entendimiento, incluso por personas no afines al área, pues su construcción en sencilla: las hojas toman los valores objetivos, mientras los atributos y sus posibles valores conforman los nodos y ramas respectivamente (Sammut & Webb, 2011).
Basados en árboles de decisión existen otros algoritmos de clasificación como: ID3, C4.5 y CART. Cada uno de ellos fue desarrollado como una versión mejorada del anterior, pero todos tienen una alta eficiencia y tiempos de ejecución reducidos, lo que los hace igualmente populares en la actualidad (Javed Mehedi Shamrat *et al.*, 2022). Se comparan y analizan en (Gupta *et al.*, 2017), donde se arrojan sus principales características:

- ID3 (Iterative Dichotomiser 3): Basa su funcionamiento en la entropía y la ganancia de información. El árbol se construye iterativamente de arriba hacia abajo, eligiendo en cada caso el atributo con mayor ganancia de información, hasta que la información ganada sea cero o se haya llegado a todas las hojas (Javed Mehedi Shamrat *et al.*, 2022), (Gupta *et al.*, 2017). Maneja datos nominales y no tolera valores faltantes.

- C4.5 (Classification 4.5): Desarrollado con el objetivo de mejorar los defectos de ID3. Añade la poda, desestimando las ramas sin aportes, que reduce los errores al clasificar como resultado de un gran número de ramas en el modelo (Sammut & Webb, 2011), (Javed Mehedi Shamrat *et al.*, 2022), (Gupta *et al.*, 2017). Adicionalmente maneja valores faltantes y numéricos.

- CART (Classification and Regression Trees): Genera un árbol binario siguiendo el mismo enfoque entrópico que ID3, pero empleando el coeficiente de Gini como criterio de selección (Javed Mehedi Shamrat *et al.*, 2022), (Gupta *et al.*, 2017). Es capaz de manejar datos faltantes, al igual que datos numéricos y nominales.

Clasificación con Redes Neuronales

Las redes neuronales o redes neuronales artificiales, son algoritmos de aprendizaje basados en una vaga analogía de cómo funciona el cerebro humano. El aprendizaje se logra ajustando los pesos en las conexiones entre nodos, que son análogas a las sinapsis y las neuronas (Sammut & Webb, 2011).
Las primeras redes neuronales conocidas como pre-alimentadas, como la de la figura 1.1, se caracterizan por tener una arquitectura en la que cada capa de neuronas está conectada completamente con la capa siguiente, pero no con la capa anterior. Esto significa que la información fluye de forma unidireccional, sin retroalimentación (Abiodun *et al.*, 2018).

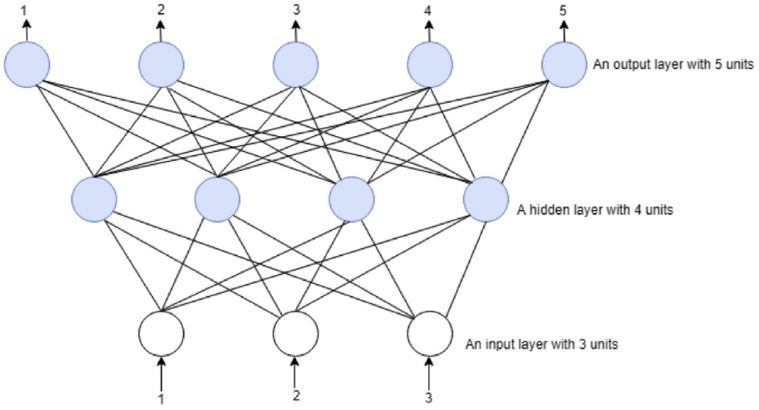

Figura 1.1: Red Neuronal Pre-Alimentada (Abiodun *et al.*, 2018)

Posteriormente, se desarrollaron las redes neuronales por retro-propagación, como la presente en la figura 1.2, que comparan la salida obtenida por la red con la salida deseada, y ajustan los pesos de las conexiones entre las neuronas de la red para reducir el error de predicción. La retro-propagación se utiliza para calcular la contribución de cada peso en el error de la red, y así ajustarlos de manera adecuada (Abiodun *et al.*, 2018).

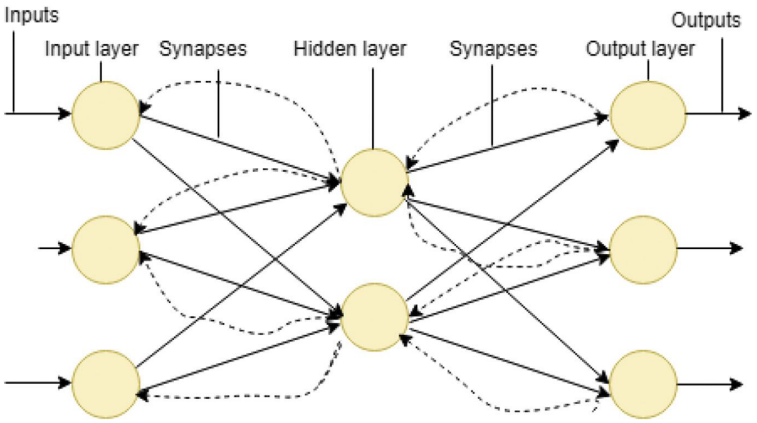

Figura 1.2: Red neuronal por retro-propagación (Abiodun *et al.*, 2018)

Gracias a su gran versatilidad se pueden aplicar en una amplia variedad de campos y disciplinas para resolver problemas complejos de aprendizaje automático, como es el procesamiento del lenguaje natural, reconocimiento de voz e imágenes (Abiodun *et al.*, 2018). Debido a la complejidad de sus modelos puede ser difícil su entendimiento, por lo que preferiblemente se utilizan en el contexto del reconocimiento de patrones. En la figura 1.3 se muestra un ejemplo de red neuronal multicapa para el reconocimiento facial.

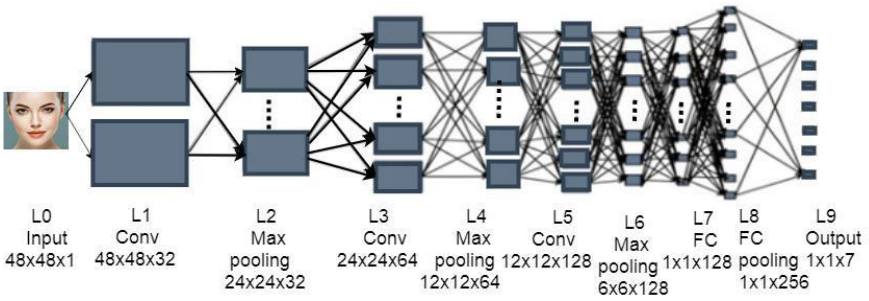

Figura 1.3: Red neuronal multicapa compleja (Abiodun *et al.*, 2018)

Clasificación con Support Vector Machine

Las Máquinas de Soporte Vectorial (Support Vector Machine o SVM, en inglés), son una clase de algoritmos lineales que se pueden emplear para la clasificación (Sammut & Webb, 2011), cuyo objetivo es encontrar el hiperplano que mejor separa dos clases de datos en un espacio de alta dimensionalidad (Guenther & Schonlau, 2016). En la figura 1.4 se representa como el hiperplano H2 divide con mayor margen las clases que el hiperplano H1.

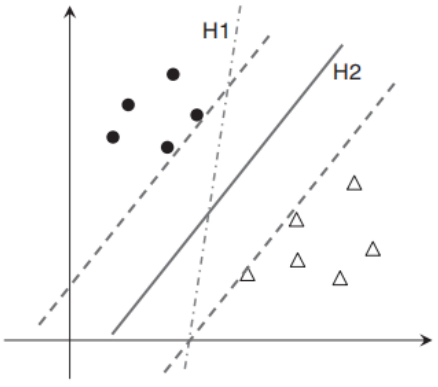

Figura 1.4: Ejemplos de posibles hiperplanos de SVM (Sammut & Webb, 2011)

Uno de los desafíos de SVM en problemas de clasificación de múltiples clases es cómo manejar la predicción de estas. En este contexto, existen dos enfoques principales:

- Uno contra uno (One-vs-One): Durante la fase de predicción, cada clasificador binario vota por la clase que cree que es correcta y la clase con el mayor número de votos se selecciona

como la clase final para el punto de datos dado. Este enfoque presenta la ventaja de que cada clasificador binario solo necesita ser entrenado en un subconjunto de los datos, lo que puede ser útil cuando hay grandes conjuntos de datos. Es más resistente a desequilibrios en la distribución de clases que otros enfoques de SVM (Guenther & Schonlau, 2016).

- Uno contra todos (One-vs-All): Entrena un clasificador binario para cada clase posible, donde el conjunto de datos de una clase se considera positivo y los datos de las otras clases se consideran negativos. Durante la fase de predicción, cada clasificador binario vota por su clase correspondiente y la clase con la mayor puntuación se selecciona como la clase final para el punto de datos dado. Fácil de implementar y puede funcionar bien en conjuntos de datos pequeños, pero puede ser menos preciso en conjuntos de datos grandes y complejos (Guenther & Schonlau, 2016).

Agregar que existen otros dos enfoques: Clasificación jerárquica (Hierarchical classification) y Clasificación por parejas (Pairwise classification). Cada uno de estos tiene sus propias ventajas y desventajas, y la elección de uno de ellos dependerá del tipo de datos y del problema que se esté tratando de resolver.

1.4. *AutoML*

El campo del Aprendizaje Automático Automatizado (AutoML), tiene como objetivo tomar decisiones de una manera automatizada, objetiva y basada en datos: el usuario simplemente proporciona datos y el sistema AutoML determina automáticamente el enfoque que funciona mejor para esta aplicación en particular (Hutter *et al.*, 2019).
AutoML (Automated Machine Learning) es una técnica que tiene como objetivo automatizar todo o parte del proceso de Machine Learning, incluyendo la selección de algoritmos, la optimización de hiperparámetros, la selección de características y la evaluación del rendimiento del modelo (He *et al.*, 2021), (Tuggener *et al.*, 2019). La relación entre AutoML y Machine Learning es que AutoML es una técnica que se utiliza para automatizar el proceso de Machine Learning, lo que significa que se utiliza para automatizar todo o parte del proceso de selección del mejor modelo de Machine Learning, para un conjunto de datos dado. La automatización del proceso de Machine Learning proporciona una solución eficiente y escalable para el análisis de grandes conjuntos de datos, lo que puede resultar en un ahorro significativo de tiempo y recursos para los profesionales de ciencia de datos e investigadores.
Tras un análisis del estado del arte acerca del proceso de AutoML (Tuggener *et al.*, 2019), (Waring *et al.*, 2020), (Hutter *et al.*, 2019), (He *et al.*, 2021), se pueden presentar las principales tareas como:

- Selección de características: Esta tarea consiste en identificar las variables más relevantes para el problema de aprendizaje automático.

- Pre-procesamiento de datos: La calidad de los datos de entrada es un factor crítico en el rendimiento de los modelos de aprendizaje automático. Las técnicas de preprocesamiento de datos se utilizan para limpiar, normalizar y transformar los datos de entrada en un formato que sea adecuado para el modelo. Existen varias técnicas efectivas de preprocesamiento de datos, incluyendo la eliminación de valores atípicos, la imputación de valores faltantes y la normalización y discretización de datos.

- Selección de modelo: Se basa en identificar el modelo de aprendizaje automático que mejor se ajusta al problema dado.

- Ajuste de hiperparámetros: Los modelos de aprendizaje automático tienen varios parámetros que afectan su rendimiento, y encontrar los valores óptimos de estos parámetros es una tarea importante para mejorar el rendimiento del modelo. El estado del arte ha demostrado que existen varias técnicas para el ajuste de hiperparámetros, incluyendo la búsqueda aleatoria (Zöller & Huber, 2021) y la optimización bayesiana (He *et al.*, 2021), Hutter *et al.* (2019).

- Evaluación del modelo: La evaluación del modelo es una tarea crítica para medir el rendimiento del modelo en datos de prueba para determinar su capacidad para generalizar. Existen varias técnicas para la evaluación del modelo, incluyendo la validación cruzada y la evaluación de curvas de aprendizaje.

- Interpretación del modelo: Consiste en analizar el modelo de aprendizaje automático para comprender cómo se toman las decisiones y qué variables son importantes para la predicción.

1.4.1. Pre-procesado

Una tarea importante en el proceso de aprendizaje automático es el pre-procesamiento de datos, que es el conjunto de técnicas utilizadas para preparar los datos de entrada antes de alimentarlos a un modelo de aprendizaje automático. Esta etapa es la equivalente a la fase de selección, limpieza y transformación del proceso de KDD, descrita brevemente en la sección 1.2.
El preprocesamiento de datos ayuda a mejorar la calidad de los datos de entrada y puede mejorar el rendimiento del modelo. La automatización del preprocesamiento de datos a través de herramientas de AutoML, puede mejorar la eficiencia del proceso y ayudar al personal especializado, o sin mucha experiencia en el campo, a trabajar de manera más efectiva. Unas de las técnicas de preprocesado de datos son la discretización y la normalización.

Discretización

Este procedimiento transforma datos cuantitativos en datos cualitativos, es decir, atributos numéricos en atributos discretos o nominales con un número finito de intervalos, obteniendo una partición no superpuesta de un dominio continuo. Luego se establece una asociación entre cada

intervalo con un valor numérico discreto. Una vez realizada la discretización, los datos pueden ser tratados como datos nominales durante cualquier proceso de minería de datos (García *et al.*, 2015), (Garcia *et al.*, 2012). Es necesario realizar la discretización de variables porque, entre varios factores, muchos de los algoritmos de Aprendizaje Automático requieren el uso de valores nominales solamente, como es el caso de ID3, Redes Bayesianas y Apriori. Otras ventajas derivadas de la discretización son la reducción y simplificación de datos, agilizando el aprendizaje y arrojando resultados más precisos, compactos y breves; y se reduce el posible ruido presente en los datos (Garcia *et al.*, 2012). No obstante, cualquier proceso de discretización conlleva generalmente una pérdida de información, siendo la minimización de esta pérdida el objetivo principal de un discretizador.
En general, los métodos de discretización pueden clasificarse como: supervisados o no supervisados, directos o incrementales, globales o locales, estáticos o dinámicos, de arriba hacia abajo o de abajo hacia arriba (top-down / bottom-up, respectivamente) (Garcia *et al.*, 2012), (Kotsiantis & Kanellopoulos, 2006).

- *Supervisados vs. no supervisados:* Los discretizadores no supervisados no consideran la etiqueta de clase, mientras que los supervisados sí. La forma en que estos últimos consideran el atributo de clase depende de la interacción entre los atributos de entrada y las etiquetas de clase, y de las medidas heurísticas utilizadas para determinar los mejores puntos de corte (entropía, interdependencia, etc.).

- *Directos vs. incrementales:* Los métodos directos dividen el rango en k intervalos simultáneamente (de igual ancho), y requieren una entrada adicional del usuario para determinar el número de intervalos. Los métodos incrementales comienzan con una discretización simple, y mejoran gradualmente, necesitando un criterio adicional para detener la discretización.

- *Globales vs. locales:* La distinción entre los métodos de discretización globales y locales depende del momento en que se realiza la discretización, siendo los métodos globales superiores, ya que utilizan todo el dominio de valores de un atributo numérico para la discretización. Mientras, los métodos locales producen intervalos que se aplican a sub-particiones del espacio de instancias.

- *Estáticos vs. dinámicos:* La diferencia entre los métodos de discretización estática y dinámica depende de si el método tiene en cuenta las interacciones entre características. Los métodos estáticos, como la división en categorías, la partición basada en entropía y el algoritmo 1R, determinan el número de particiones para cada atributo de forma independiente de las demás características; mientras que los métodos dinámicos realizan una búsqueda a través del espacio de posibles k particiones para todas las características simultáneamente, capturando interdependencias en la discretización de características.

- *Top-down vs. bottom-up:* Los métodos de arriba hacia abajo, consideran un intervalo grande que contiene todos los valores conocidos de una característica, y luego dividen este intervalo en subintervalos cada vez más pequeños, hasta que se alcanza un cierto criterio de detención, como el criterio de longitud mínima de descripción (MDL) o el número óptimo de intervalos. Por el contrario, los métodos de abajo hacia arriba, consideran inicialmente un número de intervalos determinado por los puntos de límite y combinan estos intervalos durante la ejecución, hasta un cierto punto de detención.

La identificación del mejor discretizador para cada situación es una tarea muy difícil de llevar a cabo. A pesar de la riqueza de la literatura, y aparte de la ausencia de una categorización completa de los discretizadores usando una notación unificada, hay pocos intentos de compararlos empíricamente. Esto se debe a que la evaluación de resultados es un tema complejo y depende de la necesidad del usuario en una aplicación en particular; además, la evaluación se puede hacer de muchas maneras. Existen tres dimensiones importantes según (Liu *et al.*, 2002):

1. El número total de intervalos: intuitivamente, cuantos menos puntos de corte, mejor será el resultado de la discretización.

2. El número de inconsistencias causadas por la discretización: no debe ser mucho mayor que el número de inconsistencias de los datos originales antes de la discretización.

3. Precisión predictiva: cómo la discretización ayuda a mejorar la precisión.

En resumen, se necesitan al menos tres dimensiones: simplicidad, consistencia y precisión. Idealmente, el mejor resultado de discretización puede obtener la puntuación más alta en los tres departamentos. En realidad, puede no ser alcanzable o necesario.
El agrupamiento (**binning**), es el método más simple para discretizar un atributo de valor continuo mediante la creación de un número específico de grupos (bins). Los bins se pueden crear con el mismo ancho (*equal-width*) y la misma frecuencia (*equal-frequency*). Cada bin está asociado con un valor discreto distinto. En *equal-width*, el rango continuo de una característica se divide uniformemente en intervalos que tienen un ancho igual, y cada intervalo representa un bin. En *equal-frequency*, se coloca un número igual de valores continuos en cada bin (Liu *et al.*, 2002), (Yang & Webb, 2009). Un método simple y similar a estos es el basado en cuantiles (*quantiles-based*), donde se producen bins correspondientes a una lista de probabilidades dada. El elemento más pequeño corresponde a una probabilidad de 0 y el más grande a una probabilidad de 1.
Otro algoritmo de discretización es CAIM. El objetivo del algoritmo CAIM es encontrar el número mínimo de intervalos discretos mientras se minimiza la pérdida de interdependencia de atributo de clase. El algoritmo utiliza información de interdependencia de atributo de clase como criterio para la discretización óptima (Kurgan & Cios, 2004).

Normalización

Este subepígrafe será desarrollado más adelante, durante la redacción de la tesis.

1.4.2. Optimización de hiperparámetros

Cada sistema de aprendizaje automático tiene hiperparámetros, y la tarea básica de AutoML es automatizar el ajuste de estos, para maximizar el rendimiento del modelo en un conjunto de datos de prueba o validación (Hastie *et al.*, 2009). La optimización de hiperparámetros automatizada (HPO) tiene varios casos de uso importantes (Hutter *et al.*, 2019); puede

- reducir el esfuerzo humano necesario para aplicar el aprendizaje automático. Particularmente importante en el contexto de AutoML,
- mejorar el rendimiento de los algoritmos de aprendizaje automático (adaptándolos al problema en cuestión), y
- mejorar la reproducibilidad y equidad de los estudios científicos.

Para ahorrar tiempo y mejorar la precisión de los modelos de aprendizaje automático, se combina la selección de algoritmos y la optimización de hiperparámetros en un solo proceso, denominado Selección de Algoritmos y Optimización de Hiperparámetros Combinados (*CASH*, por sus siglas en inglés) (Tuggener *et al.*, 2019).
CASH, en conjunción con la automatización del pre-procesado de los datos, integran el problema general del AutoML, reflejado en la figura 1.5.

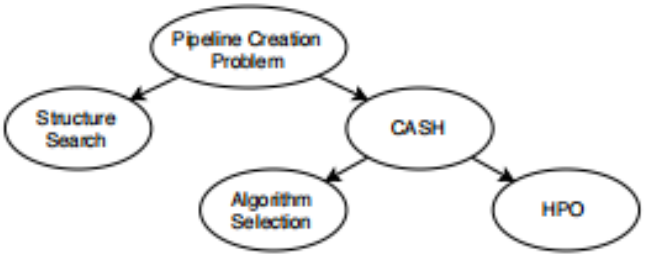

Figura 1.5: Desglose de los subproblemas de AutoML (Zöller & Huber, 2021)

Existen varias estrategias de HPO, algunas de las cuales son:

- Búsqueda aleatoria: Selecciona valores de forma aleatoria dentro de un rango definido. No garantiza encontrar los mejores valores posibles, y puede requerir numerosas iteraciones para encontrar un conjunto de hiperparámetros que proporcione un rendimiento óptimo (Géron, 2022), (Zöller & Huber, 2021).

- Búsqueda voraz: Prueba todas las combinaciones posibles de valores de los hiperparámetros dentro de un rango definido. Presenta una alta demanda computacional, imposible de manejar a medida que escalan los sistemas y las bases de datos (Zöller & Huber, 2021).

- Búsqueda en cuadrícula: Prueba cada combinación de valores de hiperparámetros y selecciona la combinación que haya dado el mejor rendimiento. Puede ser computacionalmente costoso si el espacio de búsqueda y el número de hiperparámetros es grande (He *et al.*, 2021).

- Optimización Bayesiana: Construye modelos probabilísticos para representar la función objetivo, que se actualiza después de cada iteración. Maneja espacios de búsqueda de alta dimensionalidad, y no requiere tantas iteraciones, como la búsqueda en cuadrícula o la búsqueda aleatoria, para encontrar combinaciones de hiperparámetros de alto rendimiento (Hutter *et al.*, 2019), (He *et al.*, 2021).

- Optimización basada en gradiente: Emplea la información del gradiente para optimizar los hiperparámetros de manera iterativa. Genera una gran carga computacional y presenta la limitación de caer en mínimos locales (Zöller & Huber, 2021).

Para seleccionar los mejores valores de hiperparámetros, es necesario evaluar el rendimiento del modelo en datos que no se utilizaron para el entrenamiento. En este contexto, las técnicas de validación son una herramienta esencial para evaluar y comparar diferentes combinaciones de hiperparámetros. La validación cruzada es una de las técnicas más utilizadas, consistente en dividir el conjunto de datos en *k-folds* o particiones y entrenar el modelo *k* veces, utilizando una partición diferente para la validación y las *k-1* particiones restantes para el entrenamiento. El rendimiento del modelo se evalúa en cada iteración utilizando la partición de validación (Hastie *et al.*, 2009).
A pesar del incesante estudio vinculado al HPO, este sigue presentando en la actualidad diversos desafíos (Hutter *et al.*, 2019):

- Costo computacional: la optimización de hiperparámetros puede ser muy costosa en términos de tiempo de cómputo y recursos de hardware, especialmente cuando se utiliza un espacio de hiperparámetros grande o se ejecutan muchas iteraciones de entrenamiento. Esto puede limitar la escalabilidad y la eficiencia de la HPO.

- Generalización del modelo: la optimización de hiperparámetros puede resultar en un modelo altamente ajustado, que no generaliza bien a nuevos datos. Se torna complejo cuando las bases de datos poseen múltiples tipos de datos.

- Complejidad del espacio de hiperparámetros: el espacio de hiperparámetros puede ser muy complejo y estar altamente interconectado, lo que dificulta la exploración y la selección de los hiperparámetros adecuados.

Entre las aplicaciones de HPO, se pueden encontrar (Hernández & Ortiz-Hernández, 2017), donde se aplica HPO en SVM y bosques aleatorios, para la predicción de enfermedades cardiovasculares; y (Waring *et al.*, 2020), que manifiesta el desarrollo del problema de HPO en redes neuronales, en un contexto de análisis de salud.

1.5. Herramienta de minería de datos KNIME

En la minería de datos se utilizan diferentes herramientas que simplifican el trabajo. KNIME es una de las principales herramientas de minería y análisis de datos, siendo muy completa y ofreciendo muchas funcionalidades (Bravo Ilisástigui, 2012).
La herramienta de datos KNIME (*Konstanz Information Miner*, por sus siglas en inglés), es una plataforma de minería de datos de código abierto, disponible para varias plataformas y sistemas operativos, que permite el desarrollo de modelos en un entorno visual. Esta herramienta tiene como objetivo desarrollar procesos de KDD a través de un entorno visual. Se le considera una herramienta gráfica, ya que permite construir flujos de trabajo (kni, 2023).
Los flujos se componen de flechas y nodos que se pueden combinar entre sí. Los nodos contienen funcionalidades tales como: algoritmos de minería de datos, formas de conexión a los datos almacenados, preprocesamiento de datos, reportes, entre otros. Las flechas indican el orden de ejecución y el flujo de la información. En la figura 1.6, se muestra un ejemplo de un flujo en KNIME para cargar y filtrar datos de una tabla, y posteriormente guardar los resultados en un fichero .csv.

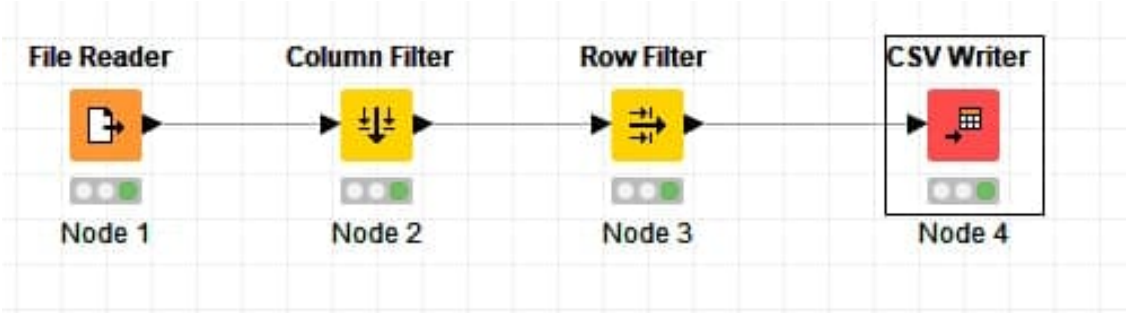

Figura 1.6: Ejemplo de flujo de trabajo en la herramienta KNIME.

La herramienta KNIME puede ser extendida a través de plugins, la mayoría son nuevos nodos, aunque las extensiones pueden ser a cualquier parte de la arquitectura. La extensibilidad de la herramienta es de forma sencilla, ya que está basada en la Plataforma de Cliente Enriquecido de Eclipse (*Eclipse RCP*, por sus siglas en inglés) (Berthold *et al.*, 2009). Gracias a esto, la adición de nuevos plugins a KNIME se torna menos compleja para el desarrollador.
KNIME se diseñó en base a tres principios: modularidad, extensibilidad y ambiente de trabajo interactivo. A continuación, se describen estos principios (Bravo Ilisástigui, 2012):

- Modularidad: Plantea que no deben existir dependencias entre las unidades contenedoras de datos o de procesamiento. Además, se pueden implementar algoritmos de manera independiente. De igual forma, al no tener tipos de datos predefinidos, se pueden definir nuevos tipos de datos, con sus características y especificaciones propias. Estos pueden declararse compatibles con otros existentes.
- Extensibilidad de forma sencilla: Permite adicionar nuevas unidades de procesamiento, visualización y tratamiento de datos, teniendo en cuenta que esto debe ser una tarea fácil de realizar.
- Ambiente de trabajo visual e interactivo: Los flujos de trabajo deben ser fáciles e interactivos para el usuario. Por tal motivo, se harán arrastrando los elementos al área de trabajo.

Dado que se realiza la modificación de un componente en esta herramienta (Carrazana Ruiz, 2022), se continúa el desarrollo en la misma.

1.6. Componente KNIME de AutoML para el pre-procesado en tareas de clasificación

AutoML en KNIME se refiere a la capacidad de la plataforma para automatizar el proceso de modelado de aprendizaje automático. Esto significa que los usuarios pueden cargar datos y permitir que la plataforma seleccione y optimice automáticamente el modelo que mejor se ajuste a los datos. Con tal objetivo, en (Carrazana Ruiz, 2022) se desarrolla un componente KNIME de AutoML para el pre-procesado en tareas de clasificación. Este, a partir de un conjunto de datos y una columna objetivo, ejecuta diferentes flujos de pre-procesado, en aras de cumplir con los requisitos de los diferentes algoritmos de clasificación, siendo capaz de entrenarlos y probarlos, para posteriormente puntuar y graficar los resultados (Carrazana Ruiz, 2022). En la figura 1.7 se muestra el flujo KNIME del Componente AutoML Clasificación (pre-procesado).

Este componente está enfocado en el pre-procesado de datos, donde se desarrollaron subcomponentes enfocados en la realización de las tareas de procesamiento de datos numéricos, *string*, valores faltantes y el ajuste de tipos de columna. Como se observa en la figura 1.7, se aplican los algoritmos de clasificación ID3, C4.5, CART, Redes Neuronales por retro propagación (RProp), Redes Neuronales Probabilísticas (PNN) y Máquina de Soporte Vectorial (SVM). Cada uno de estos requiere un tipo de pre-procesado diferente, acorde a los tipos de datos con los que trabaja.

No obstante hubo procesos que quedaron pendientes: la optimización de hiperparámetros y la automatización de algunas de las actividades esenciales en el pre-procesado, como la discretización y normalización de variables numéricas. Ambas pueden ser automatizadas y, sin embargo, están configuradas de forma estática. Por ejemplo, en la figura 1.8 se muestra el nodo AutoBinner, empleado para la discretización, agrupando datos numéricos en intervalos (bins). De igual forma, la

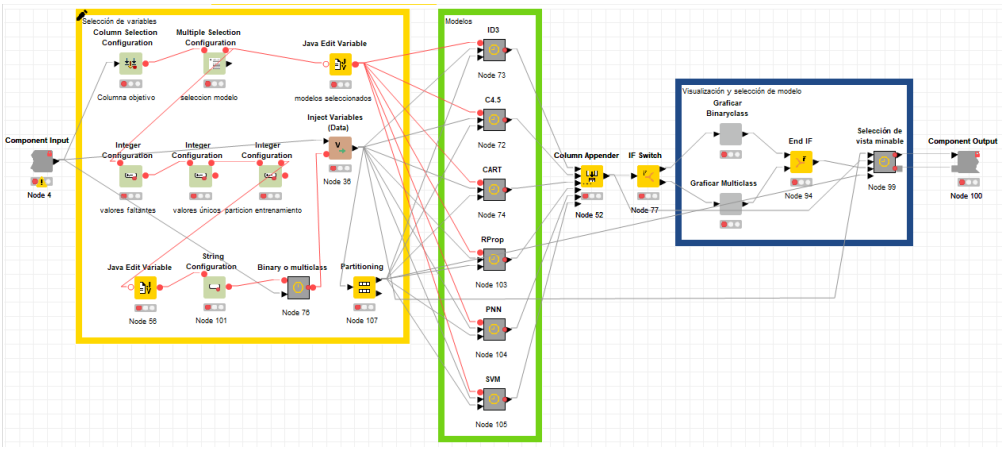

Figura 1.7: Flujo KNIME del Componente AutoML Clasificación (pre-procesado) (Carrazana Ruiz, 2022)

optimización de hiperparámetros no fue implementada en esta solución, siendo una de las tareas más importantes en AutoML para mejorar el rendimiento de los algoritmos empleados.

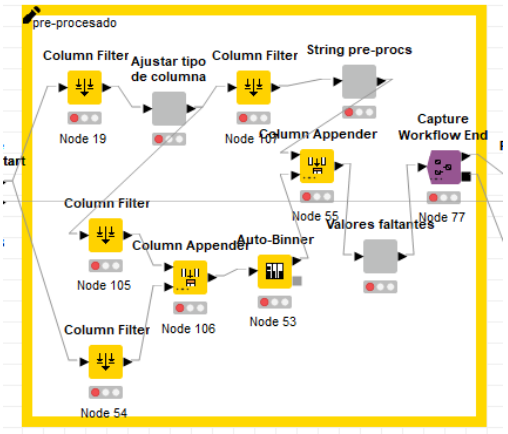

Figura 1.8: Flujo de pre-procesado para el algoritmo ID3 (Carrazana Ruiz, 2022)

1.7. Bases de datos de prueba

Las bases de datos de prueba, son conjuntos de datos creados para ayudar a los desarrolladores a probar y depurar aplicaciones de bases de datos, sin tener que utilizar datos reales y confidenciales. Estas bases de datos de prueba contienen datos ficticios, pero siguen la estructura de una base de datos real, lo que permite a los desarrolladores probar la funcionalidad de la aplicación sin preocuparse por dañar datos importantes o comprometer la privacidad de los usuarios. Kaggle Datasets y UCI Machine Learning Repository son dos de los repositorios en línea más populares para conjuntos de datos de prueba y de aprendizaje automático.

Kaggle Datasets es un sitio web de aprendizaje automático que ofrece una amplia variedad de con-

juntos de datos de muestra, desde datos meteorológicos hasta datos de redes sociales. Los usuarios pueden buscar entre miles de conjuntos de datos y también pueden contribuir con sus propios conjuntos de datos. Kaggle también tiene una comunidad de científicos de datos y aprendizaje automático que pueden proporcionar comentarios y ayudar a los usuarios a mejorar sus modelos de aprendizaje automático.
Por otro lado, el UCI Machine Learning Repository es un repositorio de conjuntos de datos de muestra para aprendizaje automático, minería de datos y otras aplicaciones de análisis de datos. El repositorio fue creado por la Universidad de California, Irvine, y contiene una amplia gama de conjuntos de datos, desde reconocimiento de voz hasta predicción de precios de viviendas. Los usuarios pueden descargar los conjuntos de datos de forma gratuita y utilizarlos para probar y desarrollar sus modelos de aprendizaje automático.
Ambos repositorios ofrecen una amplia variedad de conjuntos de datos, lo que los hace ideales para desarrolladores, estudiantes y profesionales de la ciencia de datos que buscan mejorar sus habilidades en el modelado de bases de datos y en la creación de modelos de aprendizaje automático precisos y efectivos. Por tal motivo, se emplearán ambos repositorios para la obtención de bases de datos para las pruebas que se realizarán en este proyecto.

1.8. Conclusiones parciales

A partir de lo estudiado en este capítulo, se llega a las siguientes conclusiones:

- El Aprendizaje Automático es una técnica que permite a las computadoras aprender a partir de datos, sin necesidad de ser programadas explícitamente.
- El proceso de descubrimiento de conocimiento en bases de datos (KDD) es un proceso iterativo que consiste en varias etapas, incluyendo la selección de datos, la limpieza de datos, la transformación de datos y la minería de datos.
- La Minería de Datos es el proceso de descubrir patrones y relaciones interesantes en grandes conjuntos de datos, utilizando técnicas de aprendizaje automático, estadísticas y visualización de datos. Algunas de las técnicas utilizadas en la Minería de Datos incluyen la clasificación, la agrupación, la regresión y la asociación.
- La Clasificación es una técnica de aprendizaje automático que se utiliza para predecir la etiqueta o clase de un objeto a partir de un conjunto de características.
- El AutoML se puede utilizar para mejorar la eficiencia y la precisión del proceso de modelado, reducir la necesidad de conocimientos especializados y permitir a los usuarios enfocarse en la interpretación de los resultados. Las etapas del AutoML incluyen la selección automática de algoritmos, el preprocesamiento de datos, la optimización de hiperparámetros y la evaluación automática del modelo.

- El preprocesamiento de datos es una etapa crítica en el proceso de modelado, ya que los datos deben limpiarse, integrarse y transformarse antes de ser utilizados por los algoritmos de aprendizaje automático.
- Algunas de las tareas comunes del preprocesado de datos incluyen la eliminación de valores atípicos, el manejo de datos faltantes, la discretización y la normalización de datos numéricos.
- La discretización es una técnica utilizada para transformar datos numéricos en datos categóricos.
- Los hiperparámetros son ajustes que se realizan en los algoritmos de aprendizaje automático para mejorar su rendimiento. La optimización de hiperparámetros implica encontrar la combinación óptima de valores para los hiperparámetros.
- KNIME es una herramienta de minería de datos de código abierto que permite a los usuarios crear y ejecutar flujos de trabajo de análisis de datos.

Capítulo 2

Propuesta de modificación al componente de AutoML para pre-procesado

2.1. Subcomponente para la discretización

La discretización de variables numéricas se utiliza para convertir datos continuos en datos discretos, lo que permite que los algoritmos de aprendizaje automático puedan procesarlos y analizarlos adecuadamente. La discretización también puede mejorar la precisión de los modelos de aprendizaje automático al reducir el ruido en los datos y hacer que los patrones sean más fáciles de detectar. Para el algoritmo ID3, presente en el componente *AutoML para pre-procesado (Clasificación)*, la discretización está implementada de forma estática. En aras de automatizar este proceso, se propone el componente *Discretizer* (Figura 2.1).

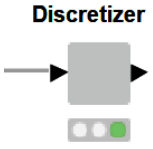

Figura 2.1: Componente *Discretizer*

El componente *Discretizer* toma en su puerto de entrada los datos en formato tabular. Dichos datos, de tipo numéricos, son procesados por varios algoritmos de discretización y, como puerto de salida, se obtienen discretizados acorde al método de discretización con mejor precisión. Este proceso es descrito de forma detallada en el siguiente epígrafe.

2.1.1. Modelación del subcomponente

El diagrama de flujo de la figura 2.2 expone el flujo general del componente *Discretizer*. El flujo KNIME correspondiente está presente en el anexo A.2.

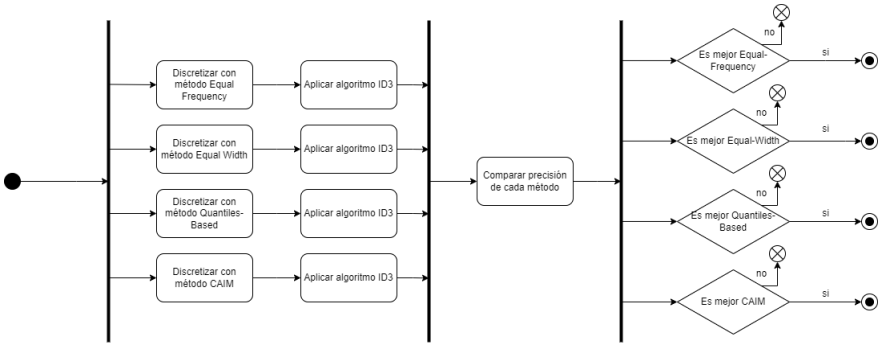

Figura 2.2: Diagrama de flujo general del subcomponente Discretizer

Para la discretización se emplean los nodos *Auto-Binner* (Figura 2.3) y *CAIM Binner* (Figura 2.4). El nodo *Auto-Binner* contiene tres métodos de discretización: *Equal-Width, Equal-Frequency* y *Quantile-Based*; mientras el nodo *CAIM Binner* contiene el método *CAIM*. Para la elección de los k intervalos, pueden emplearse varios métodos, como validación cruzada Torgo & Gama (1997); teniendo en cuenta el número de valores distintos observados en el conjunto de entrenamiento (Dougherty *et al.*, 1995); o simplemente predefiniendo un valor. En este caso no se emplea uno en particular, sino que se predefinió que serían 10 intervalos, dado que el nodo *Auto-Binner* funciona como caja negra y no permite realizar una experimentación como las mencionadas anteriormente. El nodo *CAIM Binner* no requiere una configuración en específico, y para el método Quantile-Based se escogieron los cuantiles por defecto, es decir 0.0, 0.25, 0.5, 0.75 y 1.0.

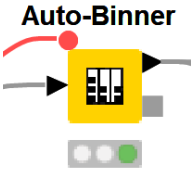

Figura 2.3: Nodo *Auto-Binner*

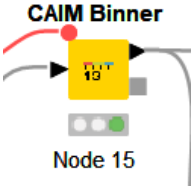

Figura 2.4: Nodo *CAIM Binner*

Tras discretizar el conjunto de datos con cada método correspondiente, se aplica el algoritmo ID3 a cada uno de los datos resultantes, con el objetivo de compararlos en cuanto a precisión y Cohen's Kappa, y escoger el algoritmo con mejor desempeño. Los flujos KNIME correspondientes

a la aplicación del algoritmo ID3 y la elección del mejor método de discretización, se encuentran en los anexos A.3 y A.4 respectivamente.

2.2. Componente para la optimización de hiperparámetros

El componente propuesto, presente en la figura 2.5, para la optimización de hiperparámetros, contiene la siguiente configuración:

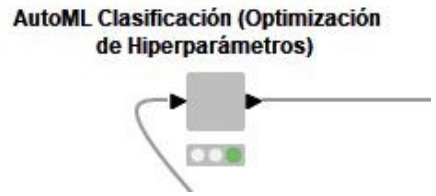

Figura 2.5: Componente *AutoML Clasificación (Optimización de Hiperparámetros)*

1. Puerto de entrada: recibe los datos de entrada en formato tabular.
2. Elementos de la configuración:
   - Columna objetivo: presenta las columnas de tipo *string* que pueden fungir como columna objetivo.
   - Estrategia de optimización de hiperparámetros: se selecciona la estrategia de optimización de hiperparámetros entre las disponibles (Random Search, Bayesian Optimization (TPE), Brute Force y Hillclimbing).
   - Selección del número de subconjuntos de la validación cruzada: el valor introducido determina la cantidad de veces que se divide el conjunto de datos en subconjuntos de entrenamiento y prueba, durante el proceso de validación.
3. Puerto de salida: tabla de hiperparámetros optimizados, siguiendo la estrategia de optimización escogida.

2.2.1. Uso y configuración del componente AutoML Clasificación (Optimización de Hiperparámetros)

A continuación, se define la secuencia de pasos para el correcto funcionamiento del componente *AutoML Clasificación (Optimización de Hiperparámetros)*:

1. Proporcionar el conjunto de datos al puerto de entrada (el componente marcará error en este punto, pues la configuración es obligatoria).

2. Dar click derecho sobre el componente, seleccionar “Configure. . . ”.
3. Seleccionar la configuración deseada (como se observa en el ejemplo de la figura 2.6).
4. Ejecutar el componente (inicialmente el puerto de salida se encuentra vacío).
5. Dar click derecho sobre el componente y seleccionar “Interactive View: AutoML” (Fig3).

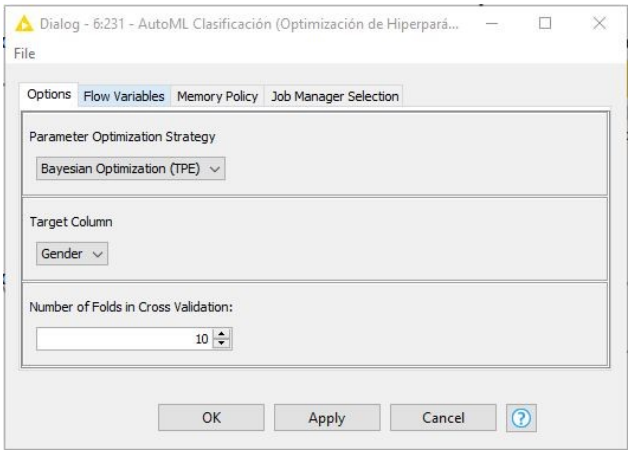

Figura 2.6: Ejemplo de configuración del componente *AutoML (Optimización de Hiperparámetros)*

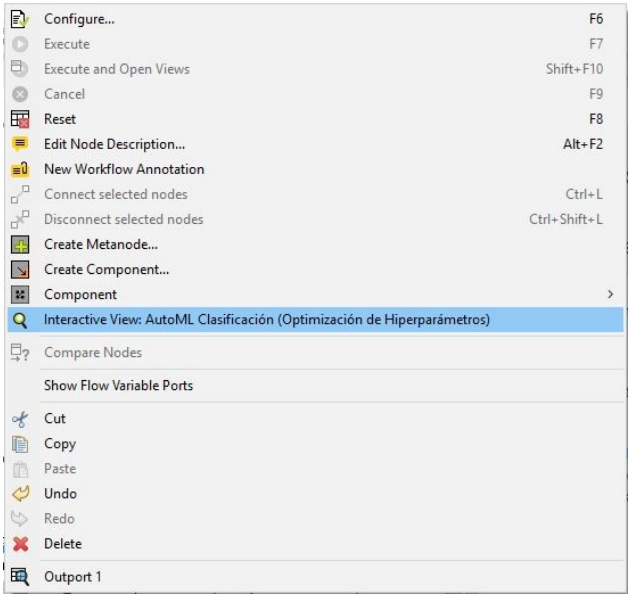

Figura 2.7: Vista de la salida.

2.2.2. Requisitos y restricciones del componente AutoML Clasificación (Optimización de Hiperparámetros)

El componente propuesto debe cumplir los siguientes requisitos funcionales:

- RF1: El componente debe permitir seleccionar columna objetivo
- RF2: El componente debe permitir seleccionar estrategia de optimización de hiperparámetros.
- RF3: El componente debe permitir seleccionar cantidad de subconjuntos en la validación cruzada.
- RF4: El componente debe optimizar hiperparámetros y entrenar datos para Redes Neuronales de Retro propagación.
- RF5: El componente debe retornar tabla de hiperparámetros optimizados.

El componente propuesto presenta las siguientes restricciones para su funcionamiento:

- Los datos de entrada deben encontrarse en formato tabular.
- La columna objetivo debe ser de tipo *string*.
- Los datos de entrada deben ser de tipo numérico, a excepción de la columna objetivo.
- Los datos numéricos deben estar previamente normalizados.

2.2.3. Modelación del componente AutoML Clasificación (Optimización de Hiperparámetros)

El diagrama de flujo de la figura 2.8 expone el flujo general del componente *AutoML Clasificación (Optimización de Hiperparámetros)*.

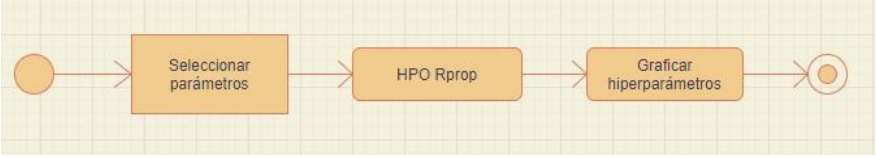

Figura 2.8: Diagrama de flujo general del componente *AutoML Clasificación (Optimización de Hiperparámetros)*

La configuración de la selección de parámetros es el primer paso, en el cual se crearán las variables que dictarán el comportamiento del flujo. Posteriormente, se ejecuta la optimización de hiperparámetros para RProp, donde recoge los resultados en una tabla y luego los grafica. El flujo KNIME correspondiente se evidencia en el Anexo A.1.

Selección de parámetros

Los parámetros que rigen el funcionamiento del componente propuesto, brindan al usuario una mayor personalización de la optimización de hiperparámetros, pues le ofrece la libertad de configurar múltiples factores claves de esta etapa. El diagrama de actividades de la figura 2.9 expone el flujo para la selección de parámetros.

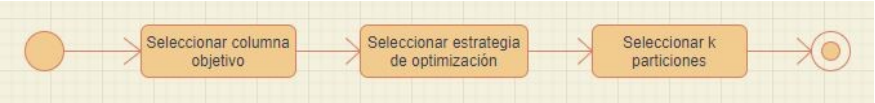

Figura 2.9: Diagrama de actividades de selección de parámetros

La selección de parámetros se realiza con los siguientes nodos de configuración, presentes en el repositorio base:

- Seleccionar columna objetivo: se emplea el nodo *Column Selection Configuration* (Figura 2.10), el cual recibe una tabla y devuelve el nombre de la columna seleccionada como variable de flujo. En este caso, presenta la configuración adicional para solo mostrar las columnas de tipo *string*.

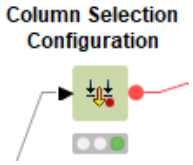

Figura 2.10: Nodo *Column Selection Configuration*

- Seleccionar estrategia de optimización de hiperparámetros: la selección de la estrategia se lleva a cabo empleando el nodo *Single Selection Configuration* (Figura 2.11). Devuelve la variable `strategy` con el valor seleccionado previamente.

Figura 2.11: Nodo *Single Selection Configuration*

- Seleccionar número de subconjuntos en la validación cruzada: la selección de la cantidad de subconjuntos de partición de entrenamiento, se lleva a cabo con el nodo *Integer Configuration* (Figura 2.12). Este devuelve la variable de flujo resultante de la selección, en este caso presenta la configuración para limitar el rango entre 5 y 10.

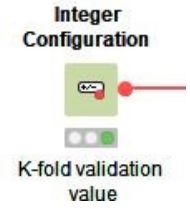

Figura 2.12: Nodo *Integer Configuration*

2.2.4. Optimización de hiperparámetros para Rprop

Las Redes Neuronales por Retro-propagación necesitan que todos los valores sean de tipo numéricos y estos se encuentren normalizados. Para el entrenamiento y prueba de las Redes Neuronales por Retro-propagación, se emplean los nodos *RProp MLP Learner* y *MultiLayerPerceptron Predictor* (Figura 2.13) respectivamente.

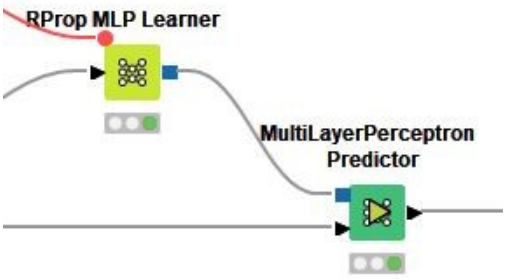

Figura 2.13: Nodos para entrenar y probar Redes Neuronales por retro-propagación

El diagrama de actividades de la figura 2.14, expone el flujo para el procesamiento necesario para la ejecución del algoritmo Redes Neuronales por Retro-ropagación, con optimización de hiperparámetros.

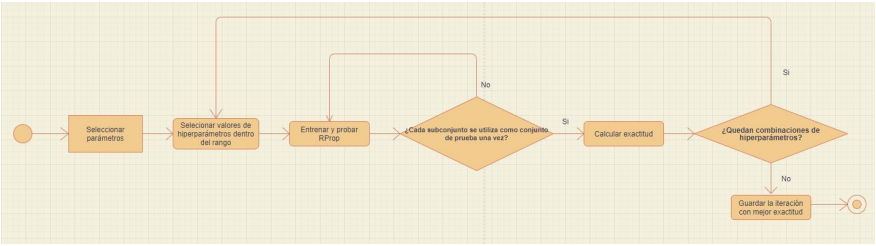

Figura 2.14: Diagrama de actividades para el procesamiento de RProp con HPO

Para llevar a cabo el procesamiento RProp se emplean los siguientes nodos:

- Ciclo para recorrer el rango de hiperparámetros: se emplean los nodos *Parameter Optimization Loop Start* y *Parameter Optimization Loop End* (Figura 2.15). Ambos permiten guardar

todas las iteraciones realizadas por el algoritmo con las diferentes combinaciones de hiperparámetros. Para su configuración, se eligieron como hiperparámetros el número de capas, la cantidad de neuronas y el número máximo de iteraciones (Figura 2.16).

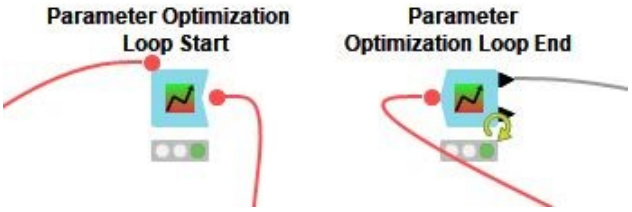

Figura 2.15: Nodos *Parameter Optimization Loop Start* y *Parameter Optimization Loop End*

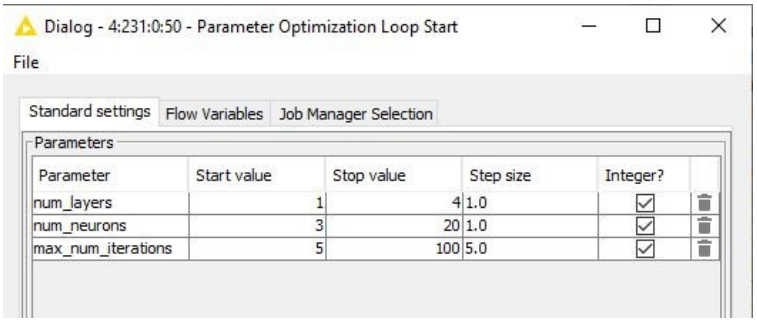

Figura 2.16: Configuración del nodo *Parameter Optimization Loop Start*

- Ciclo para dividir el conjunto de datos: se emplean los nodos *X-Partitioner* y *X-Aggregator* (Figura 2.17), para dividir el conjunto de datos en *k* particiones y realizar una validación cruzada, donde cada partición se utiliza como conjunto de prueba una vez y las otras *k-1* particiones se utilizan como conjunto de entrenamiento.

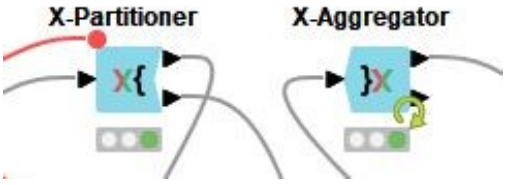

Figura 2.17: Nodos *X-Partitioner* y *X-Aggregator*

- Calcular la exactitud: se emplea el nodo *Scorer* (Figura 2.18), el cual recibe la predicción y la columna objetivo en una tabla para la evaluación.

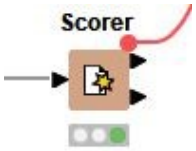

Figura 2.18: Nodo *Scorer*

- Graficar: se emplea el nodo *Table View* (Fig13), capaz de visualizar la tabla de los hiper-parámetros con mejor resultado.

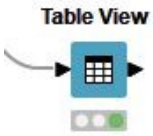

Figura 2.19: Nodo *Table View*

2.3. Conclusiones parciales

Capítulo 3

Integración y validación de soluciones propuestas al componente de AutoML

3.1. Análisis de las bases de datos de prueba

3.2. Pruebas a los nuevos componentes

3.3. Casos de prueba al componente

3.4. Comparación de resultados

3.5. Conclusiones parciales

Conclusiones

Recomendaciones

Referencias bibliográficas

2023 (Mar.). *KNIME Official Site*.

Abiodun, Oludare Isaac, Jantan, Aman, Omolara, Abiodun Esther, Dada, Kemi Victoria, Mohamed, Nachaat AbdElatif, & Arshad, Humaira. 2018. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, **4**(11), e00938.

Agrawal, Rakesh, Imielinski, Tomasz, & Swami, Arun. 1993. Mining Association Rules between Sets of Items in Large Databases.

Berthold, Michael R, Cebron, Nicolas, Dill, Fabian, Gabriel, Thomas R, Kötter, Tobias, Meinl, Thorsten, Ohl, Peter, Thiel, Kilian, & Wiswedel, Bernd. 2009. KNIME-the Konstanz information miner: version 2.0 and beyond. *AcM SIGKDD explorations Newsletter*, **11**(1), 26–31.

Bonaccorso, Giuseppe. 2017. *Machine learning algorithms*. Packt Publishing Ltd.

Bravo Ilisástigui, Lisandra. 2012. PROPUESTA DE HERRAMIENTA PARA APLICAR MINERÍA DE DATOS EN ENTORNOS COMPLEJOS.

Carrazana Ruiz, Ernesto. 2022. Componente KNIME de AutoML para pre-procesado en tareas de Clasificación.

Dougherty, James, Kohavi, Ron, & Sahami, Mehran. 1995. Supervised and unsupervised discretization of continuous features. *Pages 194–202 of: Machine learning proceedings 1995*. Elsevier.

Garcia, Salvador, Luengo, Julian, Sáez, José Antonio, Lopez, Victoria, & Herrera, Francisco. 2012. A survey of discretization techniques: Taxonomy and empirical analysis in supervised learning. *IEEE transactions on Knowledge and Data Engineering*, **25**(4), 734–750.

García, Salvador, Luengo, Julián, & Herrera, Francisco. 2015. *Data preprocessing in data mining*. Springer.

Géron, Aurélien. 2022. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. .ºReilly Media, Inc.".

Guenther, Nick, & Schonlau, Matthias. 2016. Support vector machines. *The Stata Journal*, **16**(4), 917–937.

Gupta, Bhumika, Rawat, Aditya, Jain, Akshay, Arora, Arpit, & Dhami, Naresh. 2017. Analysis of various decision tree algorithms for classification in data mining. *International Journal of Computer Applications*, **163**(8), 15–19.

Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome H, & Friedman, Jerome H. 2009. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.

He, Xin, Zhao, Kaiyong, & Chu, Xiaowen. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, **212**, 106622.

Hernández, Eduardo Sánchez-Jiménez Yasmín, & Ortiz-Hernández, Javier. 2017. Técnicas de Optimización de Hiperparámetros en Modelos de Aprendizaje Automático para Predicción de Enfermedades Cardiovasculares.

Hernández Orallo, José, *et al.* 2004. *Introducción a la Minería de Datos*. Biblioteca Hernán Malo González.

Hutter, Frank, Kotthoff, Lars, & Vanschoren, Joaquin. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.

Javed Mehedi Shamrat, FM, Ranjan, Rumesh, Hasib, Khan Md, Yadav, Amit, & Siddique, Abdul Hasib. 2022. Performance evaluation among ID3, C4.5, and CART Decision Tree Algorithm. *Pages 127–142 of: Pervasive Computing and Social Networking: Proceedings of ICPCSN 2021*. Springer.

Jiawei Han, Micheline Kamber, Jian Pei. 2011. *Data Mining. Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. 3rd edn.

Kotsiantis, Sotiris, & Kanellopoulos, Dimitris. 2006. Discretization techniques: A recent survey. *GESTS International Transactions on Computer Science and Engineering*, **32**(1), 47–58.

Kurgan, Lukasz A, & Cios, Krzysztof J. 2004. CAIM discretization algorithm. *IEEE transactions on Knowledge and Data Engineering*, **16**(2), 145–153.

Liu, Huan, Hussain, Farhad, Tan, Chew Lim, & Dash, Manoranjan. 2002. Discretization: An enabling technique. *Data mining and knowledge discovery*, **6**, 393–423.

Murphy, Kevin P. 2012. *Machine learning: a probabilistic perspective*. MIT press.

R, Praba, G, Darshan, T, Roshanraj, & B, Surya. 2021. Study On Machine Learning Algorithms. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 07, 67–72.

Sammut, Claude, & Webb, Geoffrey I. 2011. *Encyclopedia of machine learning*. Springer Science & Business Media.

Torgo, Luís, & Gama, João. 1997. Search-based class discretization. *Pages 266–273 of: Machine Learning: ECML-97: 9th European Conference on Machine Learning Prague, Czech Republic, April 23–25, 1997 Proceedings 9*. Springer.

Tuggener, Lukas, Amirian, Mohammadreza, Rombach, Katharina, Lörwald, Stefan, Varlet, Anastasia, Westermann, Christian, & Stadelmann, Thilo. 2019. Automated machine learning in practice: state of the art and recent results. *Pages 31–36 of: 2019 6th Swiss Conference on Data Science (SDS)*. IEEE.

Waring, Jonathan, Lindvall, Charlotta, & Umeton, Renato. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial intelligence in medicine*, **104**, 101822.

Yang, Ying, & Webb, Geoffrey I. 2009. Discretization for naive-Bayes learning: managing discretization bias and variance. *Machine learning*, **74**, 39–74.

Zöller, Marc-André, & Huber, Marco F. 2021. Benchmark and survey of automated machine learning frameworks. *Journal of artificial intelligence research*, **70**, 409–472.

Anexo A

Anexos

A.1. Flujo KNIME del procesamiento de RProp con HPO

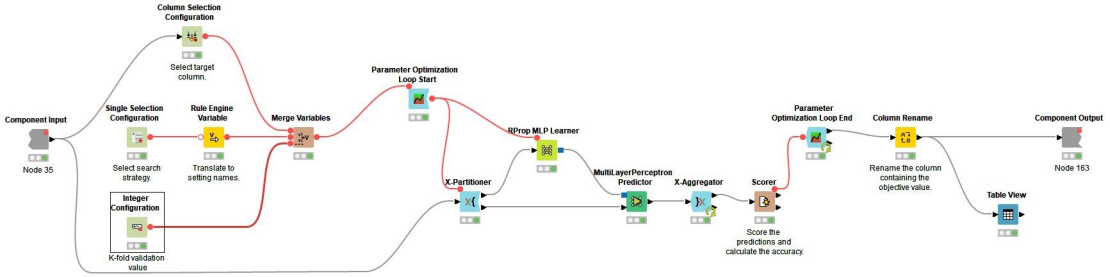

Figura A.1: Flujo KNIME del procesamiento de RProp con HPO

A.2. Flujo KNIME de subcomponente para la discretización para el algoritmo ID3

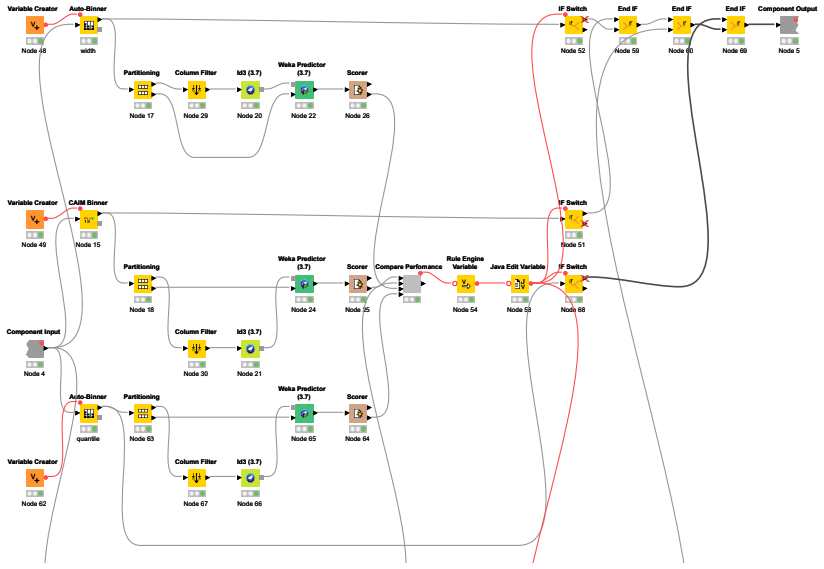

Figura A.2: Flujo KNIME de subcomponente para la discretización para el algoritmo ID3

A.3. Flujo KNIME de la ejecución del método CAIM para la discretización orientada al algoritmo ID3

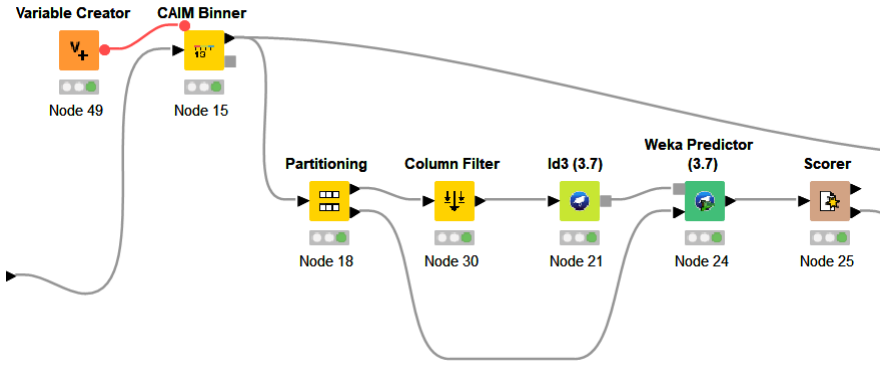

Figura A.3: Flujo KNIME de la ejecución del método CAIM para la discretización orientada al algoritmo ID3

A.4. Flujo KNIME para la comparación de métodos de discretización acorde al algoritmo ID3

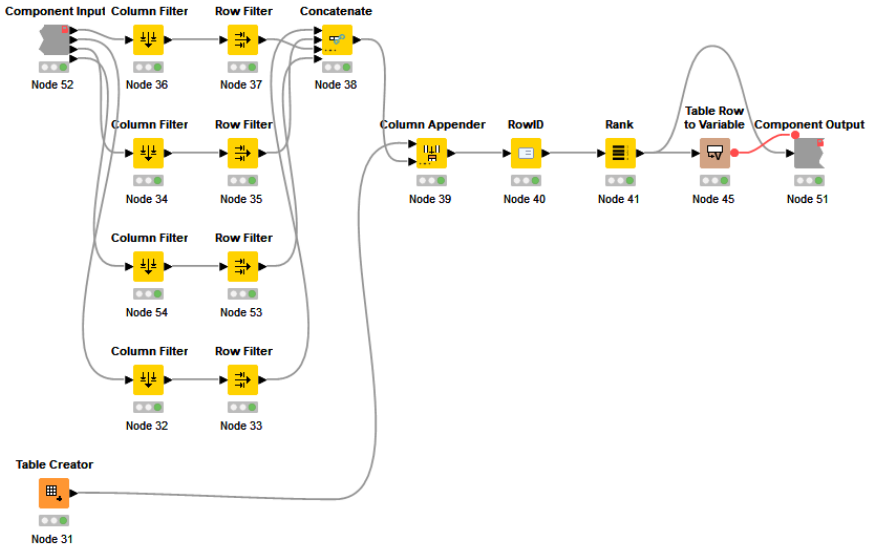

Figura A.4: Flujo KNIME para la comparación de métodos de discretización acorde al algoritmo ID3