



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

---

## Nueva versión de componente KNIME para AutoML en tareas de clasificación

---

*Trabajo de diploma para optar por el título de Ingeniería Informática*

### **Autores:**

Jennifer Yanez Jiménez  
Rainer Pellerano Alvarez

### **Tutora:**

Dr. C. Raisa Socorro Llanes

La Habana, Noviembre 2023

## **Declaración de autoría**

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Facilisi etiam dignissim diam quis. Lacinia at quis risus sed. Eu feugiat pretium nibh ipsum consequat. Viverra mauris in aliquam sem fringilla ut. Aliquam ultrices sagittis orci a scelerisque. Erat imperdiet sed euismod nisi porta lorem. Libero id faucibus nisl tincidunt. Amet volutpat consequat mauris nunc congue nisi vitae. Tellus molestie nunc non blandit massa enim nec dui nunc. Leo vel fringilla est ullamcorper. Velit euismod in pellentesque massa placerat duis ultricies.

## Opinión del tutor

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Cras tincidunt lobortis feugiat vivamus. Aliquam purus sit amet luctus venenatis lectus magna fringilla. Habitasse platea dictumst quisque sagittis purus sit amet. Vitae et leo duis ut diam quam nulla. Commodo quis imperdiet massa tincidunt. Amet tellus cras adipiscing enim eu turpis egestas. Faucibus ornare suspendisse sed nisi lacus sed. Facilisis leo vel fringilla est ullamcorper eget. Sagittis orci a scelerisque purus semper eget. Netus et malesuada fames ac turpis egestas maecenas pharetra convallis. Quis eleifend quam adipiscing vitae proin sagittis nisl. Enim sit amet venenatis urna cursus eget nunc scelerisque viverra. Ipsum dolor sit amet consectetur adipiscing elit. Ac odio tempor orci dapibus ultrices. Tincidunt vitae semper quis lectus nulla at volutpat diam. Maecenas accumsan lacus vel facilisis volutpat. Tristique sollicitudin nibh sit amet commodo nulla facilisi nullam vehicula. Quis risus sed vulputate odio. Pharetra vel turpis nunc eget lorem dolor sed viverra.

  Hac habitasse platea dictumst vestibulum rhoncus est. Suspendisse faucibus interdum posuere lorem. Diam sollicitudin tempor id eu nisl nunc mi. Dignissim diam quis enim lobortis scelerisque. Sed tempus urna et pharetra pharetra massa massa ultricies mi. Urna duis convallis convallis tellus id interdum velit. Viverra ipsum nunc aliquet bibendum enim facilisis. Viverra accumsan in nisl nisi scelerisque eu. Facilisi nullam vehicula ipsum a arcu cursus vitae congue. Aliquet eget sit amet tellus cras adipiscing. Risus ultricies tristique nulla aliquet enim tortor at. Leo urna molestie at elementum eu. Mollis aliquam ut porttitor leo a. Convallis posuere morbi leo urna. Proin fermentum leo vel orci porta non pulvinar. Nulla facilisi morbi tempus iaculis urna id volutpat lacus. Pretium nibh ipsum consequat nisl vel pretium lectus quam id.

## Dedicatoria

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Cras tincidunt lobortis feugiat vivamus. Aliquam purus sit amet luctus venenatis lectus magna fringilla. Habitasse platea dictumst quisque sagittis purus sit amet. Vitae et leo duis ut diam quam nulla. Commodo quis imperdiet massa tincidunt. Amet tellus cras adipiscing enim eu turpis egestas.

  Parturient montes nascetur ridiculus mus mauris vitae ultricies leo integer. Dolor morbi non arcu risus quis varius. Ac placerat vestibulum lectus mauris ultrices eros in. Interdum varius sit amet mattis. Volutpat sed cras ornare arcu dui vivamus. Vulputate dignissim suspendisse in est ante in nibh. Scelerisque eu ultrices vitae auctor eu augue ut. Potenti nullam ac tortor vitae purus faucibus ornare suspendisse sed. Massa eget egestas purus viverra. Viverra adipiscing at in tellus integer feugiat. Tellus molestie nunc non blandit. Augue interdum velit euismod in pellentesque massa placerat duis. Auctor elit sed vulputate mi sit amet mauris commodo quis. Pulvinar neque laoreet suspendisse interdum consectetur libero id faucibus nisl. Est ultricies integer quis auctor elit sed vulputate mi sit. Arcu vitae elementum curabitur vitae nunc sed velit. Nec nam aliquam sem et. Pellentesque elit eget gravida cum sociis natoque penatibus et. Viverra adipiscing at in tellus integer feugiat scelerisque. Porta lorem mollis aliquam ut porttitor leo a diam.

  Aenean sed adipiscing diam donec adipiscing. Consectetur a erat nam at lectus urna. Cursus metus aliquam eleifend mi in nulla. Sed adipiscing diam donec adipiscing tristique risus nec. Molestie nunc non blandit massa enim nec dui nunc. Arcu vitae elementum curabitur vitae. Elementum integer enim neque volutpat ac. Nisi porta lorem mollis aliquam. Mi in nulla posuere sollicitudin aliquam ultrices. Rhoncus mattis rhoncus urna neque viverra justo nec ultrices. Sit amet tellus cras adipiscing enim eu turpis.

# Agradecimientos

  Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Vitae nunc sed velit dignissim sodales ut eu sem integer. Et malesuada fames ac turpis. Sed vulputate odio ut enim blandit volutpat. Lacus suspendisse faucibus interdum posuere lorem ipsum dolor. Placerat dui ultricies lacus sed turpis. At erat pellentesque adipiscing commodo elit. Quam pellentesque nec nam aliquam sem et tortor consequat. Nec tincidunt praesent semper feugiat nibh sed. Feugiat in fermentum posuere urna nec tincidunt praesent semper feugiat. Euismod in pellentesque massa placerat dui ultricies lacus sed turpis. Quis vel eros donec ac odio tempor orci dapibus. Consectetur lorem donec massa sapien faucibus.

  Parturient montes nascetur ridiculus mus mauris vitae ultricies leo integer. Dolor morbi non arcu risus quis varius. Ac placerat vestibulum lectus mauris ultrices eros in. Interdum varius sit amet mattis. Volutpat sed cras ornare arcu dui vivamus. Vulputate dignissim suspendisse in est ante in nibh. Scelerisque eu ultrices vitae auctor eu augue ut. Potenti nullam ac tortor vitae purus faucibus ornare suspendisse sed. Massa eget egestas purus viverra. Viverra adipiscing at in tellus integer feugiat. Tellus molestie nunc non blandit. Augue interdum velit euismod in pellentesque massa placerat dui. Auctor elit sed vulputate mi sit amet mauris commodo quis. Pulvinar neque laoreet suspendisse interdum consectetur libero id faucibus nisl. Est ultricies integer quis auctor elit sed vulputate mi sit. Arcu vitae elementum curabitur vitae nunc sed velit. Nec nam aliquam sem et. Pellentesque elit eget gravida cum sociis natoque penatibus et. Viverra adipiscing at in tellus integer feugiat scelerisque. Porta lorem mollis aliquam ut porttitor leo a diam.

  Aenean sed adipiscing diam donec adipiscing. Consectetur a erat nam at lectus urna. Cursus metus aliquam eleifend mi in nulla. Sed adipiscing diam donec adipiscing tristique risus nec. Molestie nunc non blandit massa enim nec dui nunc. Arcu vitae elementum curabitur vitae. Elementum integer enim neque volutpat ac. Nisi porta lorem mollis aliquam. Mi in nulla posuere sollicitudin aliquam ultrices. Rhoncus mattis rhoncus urna neque viverra justo nec ultrices. Sit amet tellus cras adipiscing enim eu turpis.

  Ipsum dolor sit amet consectetur. Libero volutpat sed cras ornare arcu dui vivamus arcu. Odio facilisis mauris sit amet massa vitae tortor condimentum lacinia. Mattis ullamcorper velit sed ullamcorper morbi tincidunt ornare massa eget. Vitae tempus quam pellentesque nec nam. Pulvinar neque laoreet suspendisse interdum consectetur libero id faucibus. Nibh sed pulvinar proin gravida hendrerit lectus. Vestibulum morbi blandit cursus risus at ultrices mi tempus. Nulla at volutpat diam ut.

## Resumen

Vivimos en un mundo en el que se generan grandes cantidades de datos, los cuales se almacenan 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 la presente investigación.

**Palabras clave:** Aprendizaje Automático, Minería de datos, AutoML, KNIME, optimización de hiperparámetros, preprocessamiento 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

<b>Introducción</b>	<b>1</b>
<b>1. Aprendizaje Automático y AutoML</b>	<b>5</b>
1.1. Proceso de descubrimiento de conocimiento en bases de datos . . . . .	5
1.1.1. Minería de Datos . . . . .	6
1.2. Aprendizaje Automático . . . . .	7
1.3. <i>AutoML</i> . . . . .	9
1.3.1. Pre-procesado . . . . .	10
1.3.2. Optimización de hiperparámetros . . . . .	18
1.4. Herramienta de minería de datos KNIME . . . . .	21
1.5. <i>AutoML</i> en KNIME . . . . .	23
1.5.1. Extensión H2O.ai . . . . .	23
1.5.2. Componente AutoML KNIME . . . . .	24
1.5.3. Componente KNIME de AutoML Clasificación (pre-procesado) . . . . .	25
1.5.4. Análisis comparativo de implementaciones para AutoML en KNIME .	27
1.6. Métodos para la evaluación en clasificación . . . . .	28
1.6.1. Bases de datos de prueba . . . . .	28
1.6.2. Métricas . . . . .	29
1.7. Conclusiones parciales . . . . .	30
<b>2. Propuesta de modificaciones al componente AutoML para pre-procesado</b>	<b>32</b>
2.1. Modificaciones en el pre-procesado . . . . .	32
2.1.1. Discretización de variables numéricas . . . . .	33
2.1.2. Pre-procesado de variables de tipo <i>string</i> . . . . .	34
2.1.3. Manejo de valores faltantes . . . . .	36
2.1.4. Codificación y normalización . . . . .	38
2.2. Componente AutoML Clasificación (Optimización de hiperparámetros)	40
2.2.1. Requisitos y restricciones del componente AutoML Clasificación (Optimización de Hiperparámetros) . . . . .	42

2.2.2.	Modelación del componente AutoML Clasificación (Optimización de Hiperparámetros) . . . . .	43
2.2.3.	Optimización de hiperparámetros para RProp . . . . .	46
2.2.4.	Optimización de hiperparámetros para PNN . . . . .	47
2.2.5.	Optimización de hiperparámetros para SVM . . . . .	48
2.2.6.	Optimización de hiperparámetros para Random Forest . . . . .	49
2.3.	Modelación de nueva versión del componente AutoML Clasificación . . . . .	50
2.3.1.	Procesado de ID3 . . . . .	51
2.3.2.	Procesado para C4.5 . . . . .	52
2.3.3.	Procesado para CART . . . . .	52
2.3.4.	Procesamiento para Random Forest . . . . .	53
2.3.5.	Procesamiento para Redes Neuronales por Retropropagación . . . . .	54
2.3.6.	Procesamiento para Redes Neuronales Probabilísticas . . . . .	54
2.3.7.	Procesamiento para SVM . . . . .	55
2.4.	Conclusiones parciales . . . . .	55
<b>3.</b>	<b>Validación de Soluciones Propuestas al Componente de AutoML</b>	<b>57</b>
3.1.	Pruebas de caja negra a subcomponentes para el pre-procesado . . . . .	57
3.1.1.	Caso de prueba al subcomponente <i>Discretizer</i> . . . . .	57
3.1.2.	Caso de prueba al subcomponente <i>String proprocs</i> . . . . .	59
3.1.3.	Caso de prueba al subcomponente <i>MV Imputation</i> . . . . .	60
3.1.4.	Caso de prueba al subcomponente <i>Codificar y normalizar</i> . . . . .	61
3.2.	Pruebas de caja negra al componente <i>AutoML Clasificación (Optimización de Hiperparámetros)</i> . . . . .	63
3.2.1.	Caso de prueba para el modelo RProp . . . . .	63
3.2.2.	Caso de prueba para el modelo PNN . . . . .	63
3.2.3.	Caso de prueba para el modelo SVM . . . . .	63
3.2.4.	Caso de prueba para el modelo Random Forest . . . . .	63
3.3.	Pruebas de integración al componente <i>AutoML Clasificación (pre-procesado)</i> .	63
3.4.	Conclusiones parciales . . . . .	63
<b>Conclusiones</b>		<b>65</b>
<b>Recomendaciones</b>		<b>66</b>
<b>Referencias bibliográficas</b>		<b>67</b>
<b>A. Anexos</b>		<b>73</b>
<b>Anexos</b>		<b>73</b>

# Índice de tablas

1.1. Comparativa de las implementaciones para AutoML en KNIME . . . . .	27
1.2. Bases de datos empleadas en la experimentación . . . . .	29
3.1. Caso de prueba al componente <i>Discretizer</i> . . . . .	58
3.2. Caso de prueba al componente <i>String proprocs</i> . . . . .	59
3.3. Caso de prueba al componente <i>MV Imputation</i> . . . . .	60
3.4. Caso de prueba al componente <i>Codificar y normalizar</i> . . . . .	62

# Índice de figuras

1.1. Ejemplo de One-Hot Encoding (también conocido como Dummy Encoding) para el atributo ZIP Code . . . . .	18
1.2. Desglose de los subproblemas de AutoML (Zöller & Huber, 2021) . . . . .	19
1.3. Ejemplo de flujo de trabajo en la herramienta KNIME. . . . .	22
2.1. Diagrama de flujo general de Discretizer . . . . .	33
2.2. Nodos empleados para la discretización . . . . .	34
2.3. Diagrama de flujo para el pre-procesado de <i>string</i> . . . . .	35
2.4. Vista previa de flujo KNIME para el filtrado de valores únicos . . . . .	35
2.5. Vista previa de flujo KNIME para el reemplazo por 'other' . . . . .	36
2.6. Diagrama de flujo para el manejo de valores faltantes . . . . .	36
2.7. Diagrama de flujo para la imputación de valores faltantes . . . . .	36
2.8. Vista previa de flujo KNIME para kNNI . . . . .	38
2.9. Vista previa de flujo KNIME para kMI . . . . .	38
2.10. Diagrama de flujo del pre-procesado para la codificación y normalización . .	39
2.11. Vista previa de flujo KNIME para codificacion One-Hot . . . . .	40
2.12. Diagrama de flujo para la normalización . . . . .	40
2.13. Componente AutoML Clasificación (Optimización de Hiperparámetros) . . .	42
2.14. Diagrama de flujo general del componente AutoML Clasificación (Optimización de Hiperparámetros) . . . . .	44
2.15. Diagrama de actividades de selección de parámetros . . . . .	44
2.16. Diagrama de flujo de la optimización de hiperparámetros . . . . .	45
2.17. Diagrama de flujo para la optimización de hiperparámetros de RProp . . . .	47
2.18. Diagrama de flujo para la optimización de hiperparámetros de PNN . . . .	47
2.19. Diagrama de flujo para la optimización de hiperparámetros de SVM . . . .	49
2.20. Diagrama de flujo para la optimización de hiperparámetros de Random Forest	50
2.21. Diagrama de flujo de Componente AutoML (clasificación) . . . . .	51
2.22. Diagrama de flujo del procesamiento del modelo ID3 . . . . .	51
2.23. Diagrama de flujo del procesamiento del modelo C4.5 . . . . .	52
2.24. Diagrama de flujo del procesado de CART . . . . .	53

2.25. Diagrama de flujo del procesamiento de Random Forest . . . . .	53
2.26. Diagrama de flujo del procesamiento de RProp . . . . .	54
2.27. Diagrama de flujo para el procesado de PNN . . . . .	54
2.28. Diagrama de flujo del procesamiento de SVM . . . . .	55
3.1. Resultados del CP1 de <i>Discretizer</i> . . . . .	58
3.2. Vista previa de la salida del componente <i>Discretizer</i> . . . . .	58
3.3. Base de datos empleada para las pruebas del componente <i>String-preprocs</i> .	59
3.4. Resultados de los casos de prueba del componente <i>String preprocs</i> . . . . .	60
3.5. Resultados del CP1 del componente <i>MV Imputation</i> . . . . .	61
3.6. Resultado del CP2 del componente <i>MV Imputation</i> . . . . .	61
3.7. Resultados del CP1 del componente Codificar y Normalizar . . . . .	62
3.8. Vista previa de la salida del componente Normalizer . . . . .	62
3.9. Categorías a codificar . . . . .	63
3.10. Vista previa de la salida del componente <i>Codificar y Normalizar</i> . . . . .	63

# Introducción

En la actualidad, en el mundo se generan cada vez más datos y se 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.

El proceso de descubrimiento de conocimiento en bases de datos (*KDD*, por sus siglas en inglés), es un procedimiento que se utiliza para extraer conocimiento útil y relevante, a partir de grandes cantidades de datos almacenados en diversos sistemas (Hernández Orallo *et al.*, 2004). Este consta de varias fases: integración y recopilación; selección, limpieza y transformación; aplicación de algoritmos de minería de datos; evaluación e interpretación; así como la difusión y uso del conocimiento obtenido (Jiawei Han, 2011). El proceso de descubrimiento de conocimiento en bases de datos ha sentado las bases para una disciplina relacionada, conocida como minería de datos.

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). Una de las técnicas más comunes de la minería de datos es la clasificación, que consiste en la identificación de un conjunto de categorías o etiquetas para un conjunto de datos no etiquetados (Hernández Orallo *et al.*, 2004).

La clasificación es una técnica muy utilizada en diferentes áreas, como la detección de spam en el correo electrónico (Méndez *et al.*, 2007), la clasificación de imágenes (Borràs *et al.*, 2017) y la identificación de transacciones fraudulentas en tarjetas de crédito (Dhankhad *et al.*, 2018). Al utilizar algoritmos de clasificación es posible predecir la categoría a la que pertenece un nuevo conjunto de datos, lo que puede ser de gran utilidad en la toma de decisiones y la mejora de los procesos. Esta tarea, a su vez, es uno de los principales enfoques del Aprendizaje Automático (Machine Learning), una disciplina dentro de la inteligencia artificial, que se basa en la idea de que las computadoras pueden aprender a reconocer patrones y tomar decisiones precisas y acertadas, a través de la experiencia y la retroalimentación continua de los datos.

El Aprendizaje Automático 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. La implementación efectiva de estas técnicas requiere la intervención humana, incluyendo la selección de algoritmos adecuados, el pre-procesamiento 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 esta tarea 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 creación, 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 pre-procesado de datos, la minería de datos y la modelización de datos (KNIME, 2023b). En KNIME, el AutoML se puede implementar a través de una extensión de H2O y dos componentes AutoML; la extensión de H2O, que acoge múltiples nodos para tareas concretas; y el componente *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, aunque permite a los usuarios seleccionar automáticamente los algoritmos, no se tiene control directo sobre estos procesos y tampoco 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. Mientras, el componente *AutoML (Componente AutoML Clasificación (pre-procesado))* (Carrazana Ruiz, 2022), desarrollado en la CUJAE, ejecuta y compara el desempeño de múltiples flujos de AutoML en tareas de clasificación. En este componente se desarrollaron subcomponentes enfocados en tareas concisas de pre-procesado. Sin embargo, no contempla la optimización de hiperparámetros y la automatización de algunas de las actividades esenciales en el pre-procesado, como discretización y normalización, lo que dificulta el procesamiento de datos y la precisión de los modelos de Aprendizaje Automático. De esta manera se genera un **problema**: la inexistencia de un componente en KNIME para AutoML que contenga un pre-procesado con las tareas automatizadas e incorpore la optimización de hiperparámetros.

En aras de resolver la problemática planteada se propone una nueva versión del componente *AutoML (Componente AutoML Clasificación (pre-procesado))* (Carrazana Ruiz, 2022). Por tanto, se determina como **objetivo general** desarrollar una nueva versión del componente

KNIME que permita automatizar tareas en el pre-procesado y la selección de hiperparámetros. Este objetivo general se desglosa en los siguientes **objetivos específicos y tareas**:

1. Analizar el estado del arte de las principales técnicas de Aprendizaje Automático Automatizado para el pre-procesado y la optimización de hiperparámetros.
  - 1.1 Describir las características del proceso KDD y Minería de Datos.
  - 1.2 Caracterizar las técnicas de Aprendizaje Automático y las principales tareas de Automatización del Aprendizaje Automático.
  - 1.3 Asimilar componente AutoML Clasificación (pre-procesado).
2. Desarrollar flujos de AutoML en KNIME en tareas de clasificación.
  - 2.1 Desarrollar subcomponentes en KNIME para la automatización de actividades en el pre-procesado (discretización, normalización, tratamiento de valores faltantes y de valores únicos).
  - 2.2 Desarrollar un componente KNIME para la optimización de hiperparámetros.
3. Validar subcomponentes de actividades del pre-procesado de datos y componente AutoML Clasificación (Optimización de Hiperparámetros).
  - 3.1 Desarrollar los casos de pruebas que permitan validar los subcomponentes propuestos de pre-procesado de datos.
  - 3.2 Desarrollar los casos de pruebas que permitan validar el componente AutoML Clasificación (Optimización de Hiperparámetros).
  - 3.3 Evaluar los resultados arrojados por los casos de prueba.
4. Validar integración al componente AutoML Clasificación (pre-procesado).
  - 4.1 Diseñar y ejecutar los casos de pruebas que permitan validar la integración de los subcomponentes de pre-procesado de datos y el componente AutoML Clasificación (Optimización de Hiperparámetros) al componente AutoML Clasificación (pre-procesado).
  - 4.2 Evaluar los resultados arrojados por los casos de prueba.

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 pre-procesado 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. 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: Es donde se decide qué datos serán utilizados en el proceso de KDD. Esto implica la selección de fuentes de datos relevantes y la adquisición de los conjuntos de datos necesarios. 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 tengan la mejor calidad posible, logrando obtener la vista minable con los datos listos para la aplicación del algoritmo.

- Algoritmos de Minería de datos: En esta etapa central del proceso, se aplican algoritmos de minería de datos para identificar patrones, tendencias o estructuras en los datos. Esto puede incluir la clasificación, la segmentación, la regresión y otras técnicas de análisis.
- 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 comprender 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.

En el contexto de la creciente acumulación de datos en instituciones que han digitalizado sus registros históricos en bases de datos, la extracción de información valiosa a través de patrones ocultos se convierte en un desafío crucial. La magnitud de esta información a menudo supera las capacidades de análisis de los expertos, lo que hace imperativo recurrir a técnicas automatizadas. Por lo tanto, la Minería de Datos emerge como una necesidad vital dentro del proceso KDD, ya que permite desentrañar conocimiento significativo y no evidente en estos vastos conjuntos de datos.

### 1.1.1. 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 inferidas 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 de la siguiente manera:

- 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: 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.

La minería de datos, como etapa inicial en la exploración y extracción de conocimiento a partir de grandes conjuntos de datos, sienta las bases para el aprendizaje automático, una disciplina de la inteligencia artificial que se centra en el desarrollo de algoritmos y modelos capaces de aprender patrones y realizar tareas de toma de decisiones basadas en datos.

## 1.2. Aprendizaje Automático

Uno de los campos más destacados dentro de la Inteligencia Artificial 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 (Praba *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) (Bonacorso, 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 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).

A medida que la cantidad de datos disponibles continúa creciendo exponencialmente, y la complejidad de los modelos de Machine Learning aumenta, surge una necesidad cada

vez mayor de contar con herramientas y técnicas que permitan a los usuarios automatizar el proceso de construcción de modelos. Es aquí donde entra en juego el Aprendizaje Automático Automatizado.

### 1.3. *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, por tanto 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 como tareas principales las siguientes:

- 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, 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 pará-

metros 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.

En el marco de la presente investigación, se aborda de manera exhaustiva el enfoque fundamental de AutoML, centrándonos en dos aspectos cruciales que se explorarán en los epígrafes posteriores. En particular, se analizarán con detalle el pre-procesamiento de datos, que se enfoca en la preparación y limpieza de conjuntos de datos, y la optimización de hiperparámetros (HPO), que se concentra en la búsqueda de configuraciones óptimas para los modelos. Estos temas esenciales constituyen la base de esta investigación y servirán como pilares para la posterior discusión y análisis de resultados en este estudio.

### 1.3.1. Pre-procesamiento

Una tarea importante en el proceso de AutoML es el pre-procesamiento de datos, siendo 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 1.1. El pre-procesamiento de datos ayuda a mejorar la calidad de los datos de entrada y puede mejorar el rendimiento del modelo. La automatización del pre-procesamiento 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. Entre las técnicas de pre-procesamiento de datos se encuentran la discretización, la normalización, el tratamiento de valores faltantes y de atributos de tipo *string*.

#### 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.

La elección del número de bins o intervalos para la discretización de datos es un proceso importante en el análisis de datos cuantitativos. El número de bins determina cómo se agrupan los valores continuos en categorías o intervalos discretos y puede influir en la interpretación de los resultados. Normalmente, este número es a elección del usuario, a pesar de que no existe una regla única para determinar el número de bins. En la literatura se han encontrado otros métodos, como:

- Regla de Sturges (Coria *et al.*, 2013): Propuesta por Herbert Sturges en 1926 (Sturges, 1926), es una regla práctica para la selección del número de clases que se deben considerar al elaborar un histograma. Este número ( $k$ ) viene dado por la fórmula 1.1, donde  $n$  es el tamaño de la muestra.

$$k = 1 + \log_2 n \quad (1.1)$$

- Validación cruzada (Torgo & Gama, 1997): La idea básica es probar diferentes configuraciones de parámetros y elegir la que proporcione la mejor precisión estimada. Esta mejor configuración encontrada se utilizará luego en el algoritmo de aprendizaje en la evaluación real, utilizando un conjunto de pruebas independiente. En cuanto al componente de búsqueda, se utiliza el escalador de colinas (hill-climbing) junto con un parámetro de anticipación configurable para minimizar el conocido problema de los mínimos locales. Para la estrategia de validación se emplea la validación cruzada.
- Tamaño de la muestra (Dougherty *et al.*, 1995): Se calcula  $k$  teniendo en cuenta el número de valores distintos observados en el conjunto de entrenamiento, utilizando

$$k = \max(1, 2 * \log l) \quad (1.2)$$

donde  $l$  es el número de valores distintos observados para cada atributo.

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. Existen diversos algoritmos y enfoques utilizados para realizar esta conversión:

- 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*) o 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).
- Basado en cuantiles de muestra (*quantiles-based*): Un método simple y similar a los anteriores, 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.
- CAIM (Clustering and Mutual Information): El objetivo 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).

Existen otros métodos, que según el estudio realizado en (Garcia *et al.*, 2012) son los más empleados, aparte de los mencionados anteriormente, como MDLP (Fayyad & Irani, 1993),

ID3 (Quinlan, 1993), ChiMerge (Kerber, 1992), 1R (Holte, 1993), D2 (Catlett, 1991), y Chi2 (Liu & Setiono, 1997). Sin embargo, en este contexto de investigación, es importante destacar que estos no serán empleados para la discretización de los atributos. La decisión se basa en la disponibilidad de implementaciones en KNIME de los métodos de agrupamiento, el basado en cuantiles (quantiles-based) y el algoritmo CAIM. Esta selección se realiza en consonancia con la revisión de la literatura y la investigación existente en el campo.

## Normalización

Las variables del conjunto de datos pueden variar en términos de magnitud, rango y unidad. Esto puede afectar negativamente el rendimiento de algoritmos, como SVM y redes neuronales, que emplean, en su mayoría, datos numéricos. En aras de resolver este problema, se realiza la normalización. Esta consiste en ajustar los valores de una variable para que estén dentro de un rango específico o sigan una distribución particular. Se ejecuta principalmente para garantizar que las diferencias en la escala de las variables no afecten negativamente el rendimiento de los algoritmos de aprendizaje automático; y para facilitar la interpretación de los datos.

Existen varias técnicas de normalización comunes, donde algunas de las más utilizadas son la normalización Min-Max, Z-Score y Escala Decimal (Garcia *et al.*, 2012). En la literatura se pueden encontrar otras técnicas, descritas a continuación:

- Normalización Min-Max: Transforma los valores de una variable para que estén en un rango específico, generalmente entre 0 y 1. Desventaja: Es sensible a valores atípicos.
- Normalización Z-score (estandarización): Transforma los valores para que tengan una media de 0 y una desviación estándar de 1. Desventaja: No escala los valores a un rango específico.
- Normalización de Escala Decimal: Transforma los valores desplazando el punto decimal, usando una división de potencia de diez, de modo que el valor absoluto máximo sea siempre inferior a 1 después de la transformación. Desventaja: Puede llevar a una pérdida significativa de precisión en los valores pequeños.
- Normalización sólida (Polatgil, 2022): Este método es más útil especialmente cuando hay un valor atípico en los datos porque utiliza el valor mediano en lugar de la media y el rango de cuartiles en lugar del rango de valores. Sin embargo, la desventaja es que esta reducción de influencia de valores atípicos puede llevar a una pérdida de sensibilidad a la variabilidad de los datos.
- Escalador de Máximo Absoluto (Max Abs): En este método cada característica de los datos se obtiene dividiendo su valor máximo absoluto (Polatgil, 2022). Desventaja: No escala los valores a un rango específico, lo que puede dificultar la interpretación.

- Normalización AMZD: Propuesta en (Patro & Sahu, 2015) y similar a Min-Max, transforma los valores de datos de manera que tengan una representación en un rango de 0 y 1, con la diferencia de que funciona a nivel individual, lo que significa que se aplica a cada elemento de datos por separado; es independiente del tamaño del conjunto de datos; y es independiente del número de dígitos en cada elemento de datos. Sin embargo, presenta como desventaja que solo puede ser aplicada a datos enteros.

En el contexto de AutoML (Aprendizaje Automático Automatizado), donde se busca una automatización eficiente del proceso de construcción de modelos, es crucial identificar un normalizador que sea coherente con los algoritmos utilizados. Algunos algoritmos pueden ser más sensibles a la escala y la distribución de las variables que otros. Por lo tanto, en esta investigación se realizan experimentos para evaluar el rendimiento de los modelos con diferentes técnicas de normalización y se selecciona la que maximice la calidad de las predicciones. En el capítulo 2 se profundiza en esta tarea.

### Tratamiento de valores faltantes

Los valores faltantes o perdidos son datos que no aparecen en la celda de la columna que les corresponde. Esto puede ocurrir por diversas razones, como errores en la recolección de datos, omisiones intencionales, fallos en la medición o simplemente porque no se disponía de información en el momento de la recopilación. Esto trae consigo varias dificultades: disminución de la eficacia, complicaciones en la gestión y análisis de datos; así como a resultados sesgados. Sin embargo, los datos incompletos son objeto de investigación debido a su influencia en la precisión de la clasificación. Normalmente, los valores perdidos pueden ser manejados de varios métodos (García *et al.*, 2015), (Ventevogel, 2020):

- Descartar los valores: Consiste en eliminar por completo los registros que contienen valores faltantes. Descartar completamente un atributo es un enfoque que se puede utilizar cuando se observan muchos valores perdidos para el mismo. Una regla general es que si faltan más del 60-70 % de los valores, es mejor eliminar el atributo (Ventevogel, 2020). Sin embargo, este método puede resultar en la pérdida de información valiosa y reducir el tamaño de la muestra.
- Imputación mediante muestreo: Este enfoque es más adecuado para funciones no categóricas. Al aplicarlo, se utilizan procedimientos de máxima verosimilitud para estimar los parámetros de la porción completa de los datos. Utilizando estos parámetros, los valores perdidos se completan mediante muestreo de esta distribución estimada.
- Imputación múltiple: Implica reemplazar los valores faltantes por un valor constante, como la media, la mediana o la moda de la variable en cuestión. Este enfoque es útil cuando los valores faltantes son aleatorios y no se espera que sigan un patrón

específico. Para datos faltantes más complejos, es posible utilizar modelos predictivos para estimar los valores perdidos. Esto puede incluir regresiones, árboles de decisión o métodos de Machine Learning más avanzados.

En el contexto de este análisis, nos enfocamos en este último método. Existe una amplia familia de métodos de imputación, desde técnicas de imputación simples como sustitución de medias, kNN, etc.; a aquellos que analizan las relaciones entre atributos tales como: basados en SVM, basados en agrupamiento, regresiones logísticas, procedimientos de máxima verosimilitud e imputación múltiple. En esta investigación se emplean los métodos kNN y k-Means para la imputación de estos valores, dado que en KNIME se tiene la posibilidad de implementarlos y en la literatura se ha demostrado que son robustos para esta tarea (Tsai & Hu, 2022), (Batista & Monard, 2003), (Patil *et al.*, 2010), (Li *et al.*, 2004).

K-Vectinos Más Cercanos (K-Nearest Neighbor o kNN) es un algoritmo de aprendizaje supervisado, que se utiliza comúnmente para la imputación de valores faltantes en conjuntos de datos. La idea básica es encontrar las  $k$  observaciones más cercanas a la observación con el valor faltante, y utilizar los valores de esas observaciones para estimar el valor faltante. El valor  $k$  en el algoritmo kNN define cuántos vecinos se verificarán para determinar la clasificación de un punto de consulta específico. Por ejemplo, si  $k=1$ , la instancia se asignará a la misma clase que su vecino más cercano. Definir  $k$  puede ser un acto de equilibrio ya que diferentes valores pueden llevar a un ajuste excesivo o insuficiente. Los valores más bajos de  $k$  pueden tener una varianza alta, pero un sesgo bajo, y los valores más grandes de  $k$  pueden generar un sesgo alto y una varianza más baja. La elección de  $k$  depende en gran medida de los datos de entrada, ya que los datos con más valores atípicos o ruido probablemente funcionarán mejor con valores más altos de  $k$ . Varios enfoques han sido propuestos y discutidos en la literatura:

- Validación cruzada: Este método se basa en obtener diferentes valores de  $k$  para diferentes conjuntos de datos, sin embargo consume mucho tiempo. Los diferentes valores de  $k$  se pueden obtener según el número de instancias en el conjunto de entrenamiento o la distribución de clase (Zhang *et al.*, 2017).
- Validación cruzada generalizada (GCV): En el estudio de (Wahba, 1975), se presenta el método 'Generalized Cross Validation (GCV)' como una alternativa para evaluar la elección de  $k$ . Este método considera la influencia promedio de las observaciones excluidas en la estimación de cada punto de muestra y aproxima el error cuadrático predictivo de la estimación.
- Raíz cuadrada de la muestra: En (Lall & Sharma, 1996) se sugiere utilizar  $k=\sqrt{n}$ , siendo  $n$  el tamaño de la muestra y  $n \geq 100$ . Esta elección es recomendada cuando los recursos computacionales son limitados. Normalmente, con un tamaño de muestra

$n$  de 50 a 200, esto corresponde a una elección de  $k$  que oscila entre 7 y 14. Cuando se utilizan los criterios GCV con el mismo tamaño de muestra, según la experiencia de (Lall & Sharma, 1996),  $k$  varía entre 5 y 10.

- $kTree^2$ : (Zhang & Song, 2019) primero propone el método  $kTree^2$  para aprender rápidamente un valor de  $k$  óptimo para cada muestra de prueba, agregando una etapa de entrenamiento al método kNN tradicional y generando así un modelo de entrenamiento, es decir, construyendo un árbol de decisión ( $kTree$ ) para predecir los valores  $k$  óptimos para todas las muestras de prueba.
- Método Monte Carlo: Se propone emplear el método de validación Monte Carlo para seleccionar un parámetro de suavizado óptimo  $k$  para cada muestra de prueba. Este método implica realizar muestreos aleatorios múltiples del conjunto de datos y evaluar el rendimiento del modelo para diferentes valores de  $k$ . A través de este enfoque, se busca determinar un valor óptimo de  $k$  que mejore la precisión de la clasificación para cada muestra de prueba (Zhang *et al.*, 2017).

En resumen, la elección del parámetro  $k$  en kNN se aborda desde diversas perspectivas, incluyendo métodos tradicionales como la validación cruzada, y propuestas innovadoras como el método  $kTree^2$  que destaca por su eficiencia computacional.

Por otra parte, k-Medias (k-Means) es un algoritmo de aprendizaje no supervisado que se utiliza para la agrupación de datos. Se puede usar como parte de un enfoque más amplio para la imputación, por ejemplo, para agrupar observaciones similares y luego imputar valores faltantes dentro de cada grupo utilizando técnicas específicas, como la imputación de la media o la mediana del grupo. La elección del número óptimo de clústeres  $k$  en el algoritmo k-Means, al igual que con kNN, es un paso crítico y a menudo no trivial. Existen varios métodos para ello (Bonaccorso, 2017):

- Método del Codo (Elbow Method): Calcula la suma de los cuadrados de las distancias intra-clúster para diferentes valores de  $k$  y busca el 'codo' en la gráfica. El valor de  $k$  en el codo se considera una buena estimación del número óptimo de clústeres.
- Método de la Silueta: Calcula la calidad de un *clustering* asignando a cada punto un valor de silueta, que mide cuán similar es un punto a su propio clúster en comparación con otros clústeres. Este valor, que se encuentra en el rango de -1 a 1, tiene una interpretación específica: cuando está cerca de 1, indica una buena condición, dado que los puntos están muy cerca de su propio clúster y lejos de otros clústeres; un valor cercano a 0 sugiere una superposición de clústeres, ya que la diferencia entre las medidas intra e inter clúster es mínima; mientras, un valor cercano a -1 señala una asignación incorrecta de la muestra a un clúster, ya que indica que los puntos están cerca de los clústeres vecinos.

- Método de la Calinski-Harabasz: Calcula una puntuación que mide la relación entre la dispersión intra-clúster y la dispersión inter-clúster. Un valor más alto indica una mejor separación de clústeres.

## Tratamiento de atributos de alta cardinalidad

Los atributos categóricos, en contraste con los atributos numéricos, representan categorías o etiquetas. Se dividen en dos tipos principales: nominales, que representan categorías sin orden específico (como colores o países), y ordinales, que muestran un orden jerárquico (como niveles educativos). En muchos conjuntos de datos, es común encontrarse con atributos categóricos que presentan una alta cardinalidad. La cardinalidad de un atributo categórico está definida por el número de valores distintos que un atributo puede tomar (Moeyersoms & Martens, 2015). Las variables categóricas de alta cardinalidad son variables para las cuales el número de niveles diferentes es grande en relación con el tamaño de la muestra de un conjunto de datos.

Estos atributos presentan desafíos significativos en el análisis de datos y requieren un manejo cuidadoso. Uno de los problemas clave que se enfrentan es la complejidad computacional, ya que una gran cantidad de categorías puede aumentar el tiempo de procesamiento y el consumo de memoria. Además, pueden llevar a problemas de sobreajuste en modelos de aprendizaje automático y dificultar la interpretación de resultados. Por ejemplo, en un país existen muchos importadores y exportadores diferentes. Al entrenar un modelo de aprendizaje automático, este tipo de datos no son útiles, ya que dificulta la generalización del modelo, pero con un método de codificación adecuado, se ha demostrado que estas características mejoran el rendimiento del modelo de clasificación (Hooi *et al.*, 2022), (Cerda & Varoquaux, 2020). Para abordar estos problemas, existen varias estrategias, como el agrupamiento semántico, que consiste en identificar grupos significativos desde el punto de vista semántico (Cerda *et al.*, 2018); y distintos métodos de codificación, como One-Hot, Target, Peso de Evidencia (WOE) y Radio Supervisado.

One-Hot Encoding transforma una característica categórica de  $q$  categorías únicas en representaciones numéricas, construyendo  $q$  atributos binarios, uno para cada categoría; (Avanzi *et al.*, 2023). En el ejemplo de la figura 1.1, los códigos ZIP de clientes individuales (C1, C2, etc.) se transforman utilizando este método, donde se crea una variable para cada código ZIP.

En otras palabras, cada dimensión representa la ausencia o presencia de una categoría dada. En (Hooi *et al.*, 2022) se realizan pruebas para comparar el desempeño de varias técnicas de codificación con respecto a algoritmos de Aprendizaje Automático, como SVM, árboles de decisión y redes neuronales, lo cual, con respecto al codificador One-Hot, se obtuvieron muy buenos resultados.

The diagram illustrates the process of One-Hot Encoding (also known as Dummy Encoding). On the left, there is a small table titled "Zip code" with rows C1, C2, ..., C100, ..., C3M and their corresponding values 2000, 1800, ..., 2000, ..., 4000. An arrow labeled "Dummy encoding" points to the right, where a larger table shows the transformed data. This transformed table has columns for values 1000, 1100, ..., 1800, ..., 2000, ..., 4000. Each row corresponds to a zip code from C1 to C3M. For each value in the original table, there is a column in the transformed table with a 1 at the position corresponding to that value and 0s elsewhere. For example, for C1 (value 2000), the 2000 column has a 1, while all other columns have 0s.

	1000	1100	...	1800	...	2000	...	4000
C1	0	0		0		1		0
C2	0	0		1		0		0
...	...							
C100	0	0		0		1		0
...	...							
C3M	0	0		0		0		1

Figura 1.1: Ejemplo de One-Hot Encoding (también conocido como Dummy Encoding) para el atributo ZIP Code

Este método, aunque eficaz para atributos de baja cardinalidad, presenta la desventaja de generar una alta dimensionalidad, lo que puede ocasionar desafíos computacionales y estadísticos. Para evitarlo, es necesario aplicar reducción de dimensionalidad en la matriz de características resultante. Un enfoque natural es utilizar el Análisis de Componentes Principales (PCA) (Mahmood *et al.*, 2022), (Kasemtaweechok *et al.*, 2021) , donde la idea básica es encontrar un conjunto de transformaciones lineales de las variables originales que puedan describir la mayor parte de la varianza utilizando un número relativamente menor de variables (García *et al.*, 2015). Este enfoque permite la combinación de la esencia de los atributos originales para formar un nuevo subconjunto más pequeño de atributos. La reducción de la cardinalidad comprende las transformaciones aplicadas para obtener una representación reducida de los datos originales. Un ejemplo de reducción de cardinalidad es la introducción de un valor *otros*, que se asigna a los valores que representan menos de un cierto umbral en el atributo (Casas, 2019). Esto es especialmente útil cuando muchos valores ocurren muy pocas veces (Ventevogel, 2020).

### 1.3.2. Optimización de hiperparámetros

La optimización de hiperparámetros es un componente esencial en el campo del aprendizaje automático. En el proceso de entrenar modelos, los hiperparámetros desempeñan un papel crítico al influir en el rendimiento y la capacidad de generalización de un algoritmo. La tarea de optimizar estos hiperparámetros implica encontrar la combinación óptima que maximice la eficiencia, precisión y rendimiento de un 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 1.2.

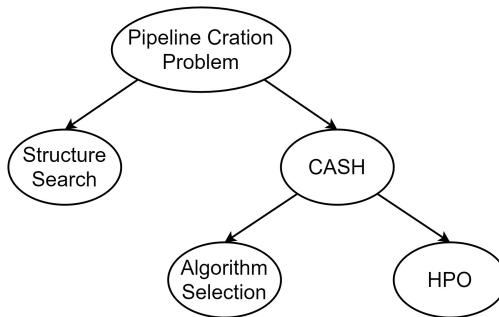


Figura 1.2: Desglose de los subproblemas de AutoML (Zöller & Huber, 2021)

Encontrar los valores óptimos para estos hiperparámetros es crucial para obtener resultados precisos y generalizables. Para lograr esto, existen diversas estrategias de optimización de hiperparámetros que permiten explorar eficientemente el espacio de búsqueda y encontrar las mejores configuraciones. A continuación, se examinan algunas de estas estrategias:

- 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).

Una vez identificadas las variables clave del modelo, es crucial evaluar su rendimiento de manera sólida y confiable. Encontrar los valores óptimos de hiperparámetros es crucial para lograr un modelo eficaz. En este contexto, las estrategias de validación desempeñan un papel esencial, ya que permiten evaluar y comparar diferentes combinaciones de hiperparámetros de manera sistemática y eficiente, garantizando que el modelo esté bien ajustado. En la literatura se encuentran estrategias de validación, como:

- Validación cruzada: Implica dividir el conjunto de datos en múltiples subconjuntos, entrenando y evaluando el modelo en diferentes particiones para estimar su capacidad de generalización. Esta estrategia permite determinar cómo se comporta el modelo con diferentes combinaciones de datos de entrenamiento y prueba, lo que es especialmente importante cuando se trata de la selección de hiperparámetros y la evaluación de su impacto en el rendimiento (Hastie *et al.*, 2009), (Bishop & Nasrabadi, 2006).
- Validación holdout: En esta estrategia, se divide el conjunto de datos en tres partes: un conjunto de entrenamiento, un conjunto de validación y un conjunto de prueba. El conjunto de entrenamiento se utiliza para ajustar los hiperparámetros, el conjunto de validación se utiliza para seleccionar los mejores hiperparámetros y el conjunto de prueba se utiliza para evaluar el rendimiento final del modelo. Esta estrategia es útil cuando se dispone de suficientes datos y se desea tener una evaluación más realista del rendimiento del modelo (Bishop & Nasrabadi, 2006).
- Validación escalonada: Esta estrategia es similar a la validación holdout, pero se realiza en múltiples etapas. El conjunto de datos se divide en varios conjuntos, y se realiza un proceso iterativo en el que se entrena y se evalúa el modelo utilizando diferentes combinaciones de hiperparámetros. Esto permite obtener una evaluación más robusta del rendimiento del modelo y reducir el sesgo introducido por una única partición de los datos (Hastie *et al.*, 2009).
- Validación Bootstrap: En esta estrategia, se generan múltiples muestras de entrenamiento mediante muestreo con reemplazo de los datos originales. Luego, se entrena y evalúa el modelo para cada muestra y se promedian los resultados. Esto permite tener una estimación del rendimiento del modelo en el conjunto de datos original (Davison & Hinkley, 1997).

La elección de la estrategia adecuada depende del tamaño del conjunto de datos, la complejidad del modelo y la disponibilidad de recursos computacionales. Es importante seleccionar la estrategia que mejor se ajuste a las necesidades específicas del problema y los datos.

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 dos ejemplos destacados en la literatura. En primer lugar, (Hernández & Ortiz-Hernández, 2017) aplica HPO en SVM y bosques aleatorios para predecir enfermedades cardiovasculares. En segundo lugar, (Waring *et al.*, 2020) aborda el desarrollo del problema de HPO en redes neuronales en un contexto de análisis de salud.

## 1.4. Herramienta de minería de datos KNIME

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 (KNIME, 2023b).

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, pre-procesamiento de datos, reportes, entre otros. Las flechas indican el orden de ejecución y el flujo de la información. En la figura 1.3, 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.

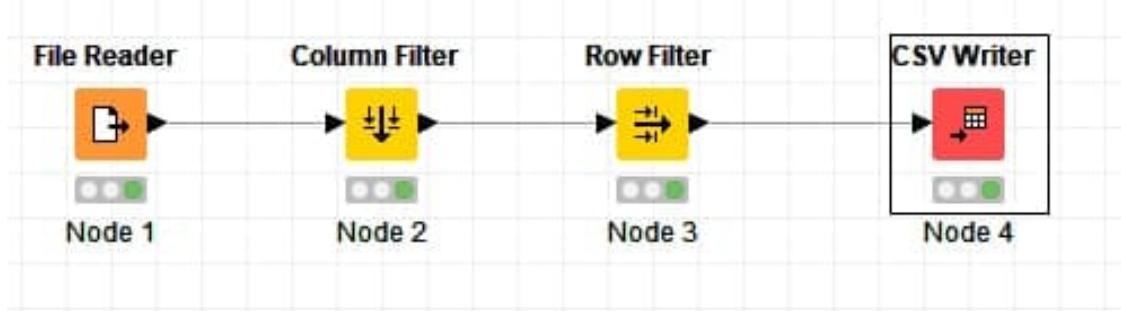


Figura 1.3: Ejemplo de flujo de trabajo en la herramienta KNIME.

Los metanodos son un tipo de nodo que contiene un subflujo, que pueden contener varios nodos y metanodos (Berthold *et al.*, 2009), comportándose como un flujo regular. Generalmente se emplea para organizar grandes flujos en varios subflujos que agrupen pequeñas metas, ganando así en claridad. Los componentes en KNIME pueden conformarse por flujos y metanodos, incluso por otros componentes. Estos, a diferencia de los metanodos, poseen la capacidad de configurar parámetros.

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 contendoras 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.

El AutoML en KNIME puede ayudar a simplificar y agilizar el flujo de trabajo de aprendizaje automático, facilitando a los usuarios el desarrollo de modelos predictivos sin ne-

cesidad de una intervención manual extensa. Dado que se realiza la modificación a un componente en esta herramienta, se continúa el desarrollo en la misma.

## 1.5. *AutoML* en KNIME

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 ese objetivo se han desarrollado extensiones y componentes, que permiten a los usuarios de KNIME automatizar tareas de aprendizaje automático, lo que ahorra tiempo y esfuerzo, especialmente para aquellos que pueden no ser expertos en Machine Learning. Sin embargo, al igual que con cualquier herramienta de AutoML, también pueden presentar desafíos, como la selección de algoritmos y la interpretación de modelos. Por lo tanto, es importante comprender las capacidades y limitaciones de las extensiones de AutoML en KNIME y cómo se integran en su flujo de trabajo de análisis de datos antes de su implementación. En las siguientes secciones se analizan de forma breve estas implementaciones.

### 1.5.1. Extensión H2O.ai

H2O es una plataforma de aprendizaje automático distribuida de código abierto, diseñada para escalar a conjuntos de datos muy grandes, con API en R, Python, Java y Scala. H2O AutoML (LeDell & Poirier, 2020) es un algoritmo automatizado de aprendizaje automático incluido en el marco H2O, que es fácil de usar y produce modelos de alta calidad que son adecuados para su implementación en un entorno empresarial. H2O AutoML admite el entrenamiento supervisado de modelos de regresión, clasificación binaria y clasificación multiclas en conjuntos de datos tabulares.

H2O.ai, la empresa detrás de H2O AutoML, ha desarrollado una integración con KNIME. Esta permite a los usuarios de la plataforma aprovechar las capacidades de H2O para construir modelos de aprendizaje automático en el entorno de KNIME, lo que facilita la creación de flujos de trabajo de análisis de datos completos que incluyan tareas de preparación de datos y modelado predictivo. Dentro de la extensión H2O de KNIME, se halla el nodo H2O AutoML, que automatiza el proceso de aprendizaje automático. Este nodo lleva a cabo tareas como el entrenamiento y ajuste automático de diversos modelos, tales como la Máquina de Aumento de Gradiente, el Modelo Lineal Generalizado, las Redes Neuronales Profundas y el Bosque Aleatorio, todo ello dentro de un intervalo de tiempo predeterminado por el usuario.

H2O AutoML selecciona el modelo principal y, como parte del proceso de aprendizaje, opti-

miza automáticamente los hiperparámetros mediante una búsqueda de cuadrícula aleatoria. Además, implementa una validación cruzada con un valor predeterminado de 5 particiones como estrategia de validación. Cabe destacar que este nodo no contiene opciones de pre-procesado de los datos ni permite la visualización ni la modificación de su contenido, únicamente se puede acceder a la interfaz de configuración. Por otra parte, es necesaria la intervención humana para la implementación de los flujos.

### 1.5.2. Componente AutoML KNIME

Este componente de KNIME, incorporado en el 2021, tiene la capacidad de entrenar automáticamente modelos de aprendizaje automático supervisados para clasificación binaria y multiclas. Automatiza el ciclo de aprendizaje automático, abarcando desde la preparación de datos, la optimización de parámetros con validación cruzada, la puntuación, la evaluación y hasta la selección de modelos. Además, este componente captura de manera integral todo el proceso y genera un flujo de trabajo de implementación mediante la extensión integrada de KNIME. Su funcionamiento se puede resumir de la siguiente manera:

- Preparación de datos: Antes de entrenar los modelos, se inicia la fase de preparación de datos. Durante esta etapa, se realiza una limpieza de los datos que implica reemplazar los valores faltantes en las columnas categóricas por la moda y en las columnas numéricas por la media. Existe la opción de aplicar codificación One-Hot a los datos categóricos y de eliminar columnas con un alto número de valores únicos, según un parámetro definido por el usuario. Además, todas las características numéricas se convierten en *double* y se normalizan utilizando la normalización Z-Score. La partición de datos en conjuntos de entrenamiento y prueba se lleva a cabo automáticamente mediante muestreo estratificado en la clase objetivo.
- Entrenamiento de modelos: La etapa de entrenamiento de modelos implica la capacitación de diversos tipos de modelos, como Naive Bayes, Regresión Logística, Redes Neuronales, Árboles de Decisión, Bosques Aleatorios y otros. Cada modelo tiene parámetros ajustables que pueden ser optimizados utilizando validación cruzada y una métrica de evaluación personalizada en los datos de entrenamiento.
- Selección del mejor modelo: Una vez que se ha completado el entrenamiento de los modelos, se procede a la selección del mejor. Todos los modelos entrenados se aplican al conjunto de pruebas y se obtienen predicciones. Estas predicciones se comparan con los valores reales, lo que permite calcular diversas métricas de rendimiento. El mejor modelo se selecciona basándose en una métrica de rendimiento especificada por el usuario.

Dentro de su configuración, se incluye la selección de los modelos a utilizar, permite seleccionar la partición de entrenamiento y la estrategia de validación, que tiene un valor predeterminado de 5. Sin embargo, debido a su formato caja negra, no se puede visualizar ni modificar los flujos correspondientes a las tareas de AutoML.

### 1.5.3. Componente KNIME de AutoML Clasificación (pre-procesado)

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. Este componente está enfocado en el pre-procesado de datos, donde se desarrollaron subcomponentes orientados a la realización de las tareas de pre-procesamiento de datos numéricos, *string*, valores faltantes y el ajuste de tipos de columna. Los algoritmos de clasificación implementados son ID3, C4.5, CART, Redes Neuronales por Retropropagación (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. Este componente consta de tres etapas clave:

- Pre-procesado de los datos: Durante esta etapa se ejecutan varios flujos para limpieza de datos, en función de los requerimientos de cada algoritmo implementado. Entre estos flujos se encuentran el ajuste de tipo de columna, para llevar a cabo modificaciones que adecúen los datos al tipo de información que deberían representar; el manejo de columnas de tipo *string*, para eliminar las inconsistencias que se pueden presentar, como caracteres extraños y el uso indistinto de mayúsculas y minúsculas en un mismo valor, así como la eliminación de valores únicos por columna; el manejo de valores faltantes, donde los atributos numéricos y categóricos se sustituyen por la media y moda respectivamente; el tratamiento de valores numéricos, para la conversión de valores nominales en numéricos, y proceder a su normalización, empleando el método Min-Max; y la discretización de variables numéricas, utilizando el método Decimal Scaling.
- Entrenamiento de modelos y evaluación: Implica la ejecución de los algoritmos previamente mencionados. En cada cual se ejecuta su evaluación para ser comparados posteriormente, en base a las métricas exactitud y AUC.
- Graficado de resultados y selección del modelo: Los resultados obtenidos en la fase anterior se grafican y el usuario tiene la posibilidad de elegir el que satisfaga sus necesidades. En esta visualización se muestra un gráfico de barras con las métricas

mencionadas y una curva ROC , para evaluar el rendimiento de los modelos de clasificación.

En su configuración se encuentran la selección de la columna objetivo, los caracteres especiales a eliminar durante el pre-procesado de *string*, el umbral de valores únicos por columna y de valores perdidos permitidos, el porcentaje de partición mediante un muestreo estratificado y la selección de los algoritmos de clasificación que se emplearán. Cabe destacar que este componente permite la visualización y modificación de los flujos implementados. No obstante, a pesar de estar enfocado en el pre-procesado, presenta algunas deficiencias en esta tarea:

- Discretización: este proceso se encuentra estático, es decir, solamente se ejecuta un método de discretización con el nodo AutoBinner, el cual presenta otros métodos que pueden tener un mejor funcionamiento en función de los datos y modelos; incluso, existe otro nodo con el método CAIM para la discretización. Además, esta tarea solo se encuentra implementada en el modelo ID3, cuando C4.5, al ser un árbol de decisión, también trabaja y, a su vez, tolera mejor los datos discretizados.
- Normalización: al igual que la discretización, esta se encuentra de forma estática, cuando en el nodo Normalizer empleado, presenta tres métodos para normalizar. Por otra parte, en este componente solamente está presente la normalización para el modelo Redes Neuronales por Retropropagación; sin embargo, los modelos PNN y SVM, aunque no lo tienen como requisito en la herramienta KNIME, tienen mejor desempeño con los datos normalizados.
- Imputación de valores faltantes: el método empleado es la sustitución por la media, en caso de los atributos numéricos, y la sustitución del valor más frecuente, en caso de los valores nominales. Este enfoque, sin estar erróneo, puede sustituirse por la aplicación de métodos de imputación más avanzados, como lo son los que emplean algoritmos de Aprendizaje Automático.
- Tratamiento de valores únicos por columna: la interpretación a esta tarea es que, dado un número de valores únicos que existan en una columna de datos nominales, si al contarlos superan este umbral, se eliminan estos valores. No obstante, en este componente se implementa de forma errónea esta tarea, dado que en realidad se cuenta la cantidad de valores distintos que puede tomar un atributo y, si esta cantidad supera a la indicada por el usuario, se elimina la columna. Tras el análisis efectuado en la sección 1.3.1, se puede concluir que ambos enfoques afectan el desempeño del modelo.

De igual forma, la 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.

#### 1.5.4. Análisis comparativo de implementaciones para AutoML en KNIME

Tras analizar de forma independiente cada uno de estas implementaciones, en la tabla 1.1 se resumen las mismas, evaluando aspectos fundamentales para su modificación. La disponibilidad de documentación desempeña un papel crucial, ya que la existencia de recursos claros y accesibles puede determinar la capacidad de los usuarios para comprender y aprovechar al máximo la herramienta. Además, la posibilidad de modificación y adaptación a necesidades específicas es un factor clave, permitiendo a los usuarios personalizar la herramienta según sus requerimientos. El nivel de automatización impacta directamente en la eficiencia del proceso de modelado, lo que es esencial para ahorrar tiempo y recursos. Sin embargo, es importante destacar que todas estas implementaciones presentan desventajas y limitaciones propias, que deben ser consideradas cuidadosamente al tomar una decisión informada sobre la elección de una solución.

Tabla 1.1: Comparativa de las implementaciones para AutoML en KNIME

Herramienta	Disponibilidad de documentación	Nivel de automatización	Posibilidad de modificación	Desventajas
Extensión H2O	Limitada	Baja	Media	<ul style="list-style-type: none"> <li>- Es necesaria la intervención humana para la implementación de flujos.</li> <li>- No contiene posibilidad de pre-procesado.</li> </ul>
Componente AutoML	Limitada	Alta	Baja	<ul style="list-style-type: none"> <li>- Las actividades de AutoML están predeterminadas.</li> <li>- No se toma en cuenta el algoritmo ni las características de los datos para el pre-procesado.</li> <li>- Funciona en formato de caja negra.</li> </ul>
Componente AutoML Clasificación (pre-procesado)	Asequible	Alta	Alta	<ul style="list-style-type: none"> <li>- No está implementada la optimización de hiperparámetros.</li> <li>- Algunas actividades del pre-procesado están estáticas.</li> </ul>

Durante el transcurso de esta investigación, el componente AutoML de KNIME fue actualizado, a la fecha del 20 de octubre de 2023 (KNIME, 2023a). Estas actualizaciones incluyen algunas mejoras, donde la más relevante es el acceso a los flujos implementados y

su modificación. No obstante, tras analizar a más detalle el funcionamiento del componente, se encuentra que se mantiene la predeterminación de las tareas de AutoML. Por ejemplo, la estrategia de optimización de hiperparámetros está establecida a Búsqueda Aleatoria, y para cambiar esta configuración, la única posibilidad es que el usuario se adentre en los flujos implementados; siendo el mismo problema en la etapa del pre-procesado con las actividades de discretización, normalización y manejo de valores faltantes. Por otra parte, el pre-procesado específico de cada algoritmo no se toma en cuenta, lo cual es fundamental en la etapa de preparación de los datos. Asimismo, no se incluyen hiperparámetros que si se toman en cuenta en el desarrollo de esta investigación.

En este caso se decide continuar el desarrollo del componente *AutoML Clasificación (pre-procesado)*, para lo cual se propone la automatización total del pre-procesado de los datos y la implementación de la optimización de hiperparámetros como parte esencial para mejorar el rendimiento de los modelos de aprendizaje automático, ahorrar tiempo y recursos, aumentar la robustez, adaptarse a cambios en los datos y avanzar en la investigación y el desarrollo de algoritmos.

## 1.6. Métodos para la evaluación en clasificación

En el campo de la clasificación, existen varios métodos de evaluación que se utilizan para comparar diferentes algoritmos o enfoques en esta tarea. Estos métodos ayudan a medir el rendimiento y la eficacia de los modelos de clasificación, así como a tomar decisiones informadas sobre cuál es el mejor enfoque para una tarea específica. Cuando se trabaja en problemas de clasificación con atributos numéricos, es fundamental contar con una base de datos de prueba adecuada. Una buena base de datos debe ser representativa de los datos del mundo real, y contener una variedad de instancias y etiquetas de clase para evaluar el rendimiento de los algoritmos de clasificación.

### 1.6.1. 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 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*<sup>1</sup> y *UCI Machine Learning Repository*<sup>2</sup> son dos de los repositorios en línea más populares para conjuntos de datos de prueba y de aprendizaje automático.

---

<sup>1</sup><https://www.kaggle.com/datasets>

<sup>2</sup><https://archive.ics.uci.edu/datasets>

*Kaggle Datasets* es un sitio web de aprendizaje automático que ofrece una amplia variedad de conjuntos 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 los propios. *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 esta investigación.

La tabla 1.2 resume las bases de datos empleados en este estudio. Muestra, para cada una, el número de instancias (#Instancias), la cantidad de atributos (#Atributos), número de atributos cuantitativos y cualitativos, distribución de atributos de clase y la ausencia o presencia de valores perdidos y atributos con alta cardinalidad.

Tabla 1.2: Bases de datos empleadas en la experimentación

Base de datos	#Instancias	#Atributos (cuantitativos, cualitativos)	Clases	%Clases	Valores perdidos	Alta cardinalidad
HUMAN RESOURCES	19 158	14 (3, 11)	yes	24.93 %	si	si
			no	75.07 %		
CANCER DATA	569	32 (31, 1)	M	37.26 %	no	no
			B	62.76 %		
CENSUS INCOME	32 561	15 (6, 9)	<=50k	75.92 %	si	no
			>50k	24.08 %		

### 1.6.2. Métricas

Al comparar algoritmos de clasificación, hay varias métricas que se utilizan para evaluar y comparar su rendimiento (Géron, 2022), (Hastie *et al.*, 2009). A continuación, se presentan algunas de las métricas más comunes, empleadas en esta investigación:

- Exactitud (Accuracy): Mide la proporción de todas las predicciones correctas realizadas por el modelo en relación con el tamaño total del conjunto de datos. En otras palabras, mide cuántas de todas las instancias (positivas y negativas) se han clasificado correctamente.. Sin embargo, la exactitud puede ser engañosa cuando hay desequilibrio de clases en el conjunto de datos.
- Precisión (Precision): La precisión es una métrica que evalúa la proporción de predicciones positivas hechas por un modelo que son verdaderamente positivas. Es decir, mide cuántas de las instancias clasificadas como positivas por el modelo realmente pertenecen a la clase positiva.
- Exhaustividad (Recall): La exhaustividad, también conocida como sensibilidad o recall, es la proporción de ejemplos positivos que se clasifican correctamente en relación con el total de ejemplos positivos en el conjunto de datos. Mide la capacidad del modelo para identificar correctamente los ejemplos positivos.
- Puntuación F1 (F1 Score): La puntuación F1 es la media armónica de la precisión y la exhaustividad. Proporciona una medida equilibrada del rendimiento del modelo y es especialmente útil cuando hay un desequilibrio entre las clases.
- Matriz de confusión (Confusion Matrix): La matriz de confusión es una tabla que muestra el número de predicciones correctas e incorrectas realizadas por un modelo de clasificación. A partir de la matriz de confusión, se pueden calcular métricas como la precisión, la exhaustividad y la puntuación F1.
- Curva ROC y área bajo la curva (ROC Curve y AUC): La curva ROC es una representación gráfica del rendimiento del modelo a diferentes niveles de umbral. Muestra la tasa de verdaderos positivos (sensibilidad) en función de la tasa de falsos positivos (1 - especificidad). El área bajo la curva (AUC) es una métrica que resume la curva ROC y proporciona una medida del rendimiento general del modelo.

## 1.7. 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 pre-procesamiento 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 pre-procesado 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.
- Se implementó un componente KNIME en (Carrazana Ruiz, 2022) que brinda soporte para tareas de AutoML, enfocándose en la etapa de pre-procesado, en el cual se basa el desarrollo de los componentes en este proyecto.

## **Capítulo 2**

# **Propuesta de modificaciones al componente AutoML para pre-procesado**

En el presente capítulo, se propone una serie de modificaciones destinadas a mejorar el componente *AutoML Clasificación (pre-procesado)*. Estas modificaciones se centran en el pre-procesamiento de datos, incluyendo la automatización de tareas como la discretización, normalización y el tratamiento de valores únicos, valores de alta cardinalidad y valores faltantes. Además, se explorará la inclusión de técnicas de Optimización de Hiperparámetros (HPO) para encontrar configuraciones óptimas para los modelos. El enfoque principal de esta propuesta es la integración de estas dos facetas con el componente AutoML existente para clasificación. A lo largo de este capítulo, se describirá cómo estas modificaciones buscan mejorar la eficiencia y precisión del proceso de AutoML, creando modelos de clasificación más sólidos y adaptados a las necesidades específicas de los datos.

### **2.1. Modificaciones en el pre-procesado**

Con el fin de abordar las problemáticas presentadas en el subepígrafe 1.5.3, es decir, la falta de automatización de la discretización y normalización, el tratamiento de valores únicos, valores de alta cardinalidad y valores faltantes, se llevaron a cabo una serie de modificaciones sustanciales en el proceso de pre-procesamiento de datos.

En lo que respecta a la discretización y normalización, se implementaron técnicas automatizadas para garantizar una uniformidad en la escala y distribución de los datos, lo que contribuye a mejorar la precisión del modelado. Para abordar los valores únicos y de alta cardinalidad, se desarrollaron estrategias específicas que permitieron una gestión más efectiva de estas categorías, evitando la pérdida de información esencial y reduciendo el riesgo de sobreajuste. Por último, se implementaron métodos especializados para tratar los valores faltantes, asegurando que los datos incompletos no comprometieran la calidad del

análisis. En las secciones siguientes, se detalla el funcionamiento y modelado de estas modificaciones con mayor profundidad. Para ello, se crean subcomponentes con el objetivo de integrarse al componente *AutoML Clasificación (pre-procesado)*.

### 2.1.1. Discretización de variables numéricas

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. En aras de automatizar este proceso, se propone el subcomponente *Discretizer*.

El subcomponente *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. Tiene una única restricción: no pueden tener valores perdidos como entrada. El diagrama de flujo de la figura 2.1 expone el flujo general del componente *Discretizer*. El flujo KNIME correspondiente está presente en el anexo ??.

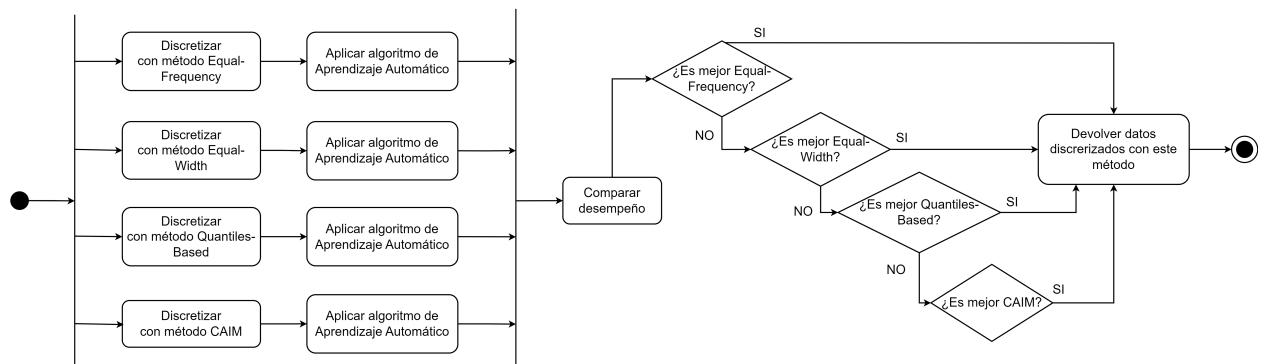


Figura 2.1: Diagrama de flujo general de *Discretizer*

Para la discretización se emplean los nodos presentes en la figura 2.2. 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 se emplea la Regla de Sturges, descrita en la sección 1.3.1, debido a su sencilla implementación en la herramienta. El nodo *CAIM Binner* no requiere una configuración en específico, y para el método *Quantile-Based* se escogieron los cuantiles 0.0, 0.25, 0.5, 0.75 y 1.0. Esta decisión se basa en varios aspectos como:

1. Distribución de los datos: para capturar diversos puntos en la distribución de los datos. Estos cuantiles proporcionan una representación equitativa de la variabilidad de los

valores en el conjunto de datos. El cuantil 0.0 representa el valor mínimo, el cuantil 1.0 representa el valor máximo y los cuantiles 0.25, 0.5 y 0.75 dividen los datos en cuartiles. Esta selección asegura que se están considerando tanto los extremos como los valores centrales de la distribución.

2. Robustez ante valores atípicos: la elección de estos cuantiles también está respaldada por la necesidad de que los modelos sean robustos ante valores atípicos. Los cuantiles 0.0 y 1.0 permiten considerar directamente los valores mínimos y máximos, lo que puede ser crítico para lidiar con datos extremos que pueden afectar significativamente el rendimiento de nuestros modelos.
3. Interpretabilidad: para mejorar la interpretabilidad de los resultados, se opta por seleccionar los cuantiles 0.25, 0.5 y 0.75, que representan los cuartiles. Los cuartiles son puntos de referencia bien conocidos en estadísticas y facilitan la comunicación de resultados a personas no técnicas.
4. Consistencia y comparabilidad: utilizar cuantiles específicos como 0.0, 0.25, 0.5, 0.75 y 1.0 puede hacer que los resultados sean más consistentes y comparables entre diferentes análisis o modelos, ya que se utilizan puntos de corte estándar en la distribución de los datos.



Figura 2.2: Nodos empleados para la discretización

Tras discretizar el conjunto de datos con cada método correspondiente, se aplica el modelo en cuestión a cada uno de los datos resultantes, con el objetivo de compararlos en cuanto a exactitud y Cohen's Kappa, y escoger el algoritmo con mejor desempeño. Estas métricas se emplean debido que son brindadas por el componente *Scorer* de KNIME, que tiene como objetivo la evaluación de modelos de clasificación. Se toma en cuenta la exactitud y, en caso de empate, el Cohen's Kappa. Este esquema de comparación se continua empleando a lo largo de este estudio.

### 2.1.2. Pre-procesado de variables de tipo *string*

Los valores nominales únicos, o categorías con un solo ejemplo en una columna, pueden parecer insignificantes a primera vista, pero su correcto manejo es esencial para evitar

posibles problemas de calidad de datos y garantizar la integridad de los análisis. El componente *AutoML Clasificación (pre-procesado)* poseía un tratamiento erróneo de estos valores al eliminar las columnas nominales que contenían un número de categorías diferentes que superaban un umbral. Este umbral era determinado por el usuario en la configuración del componente.

En aras de depurar estas inconsistencias, el diagrama de la figura 2.3 muestra un nuevo flujo para el pre-procesado de *string*, donde la actividad de color amarillo expresa que se modificó la que anteriormente se encontraba en el componente; mientras que la actividad en verde refleja una nueva implementación.

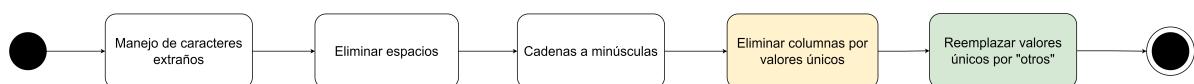


Figura 2.3: Diagrama de flujo para el pre-procesado de *string*

A continuación se exponen los cambios realizados al componente *String proprocs*:

- Eliminar valores únicos por columna: se enmienda el error antes expuesto, al modificar el subcomponente *Filtrar valores únicos*, implementando la eliminación de las columnas donde más del 80 % de los valores son únicos. En la figura 2.4 se muestra una parte del flujo KNIME donde se implementa esta tarea.

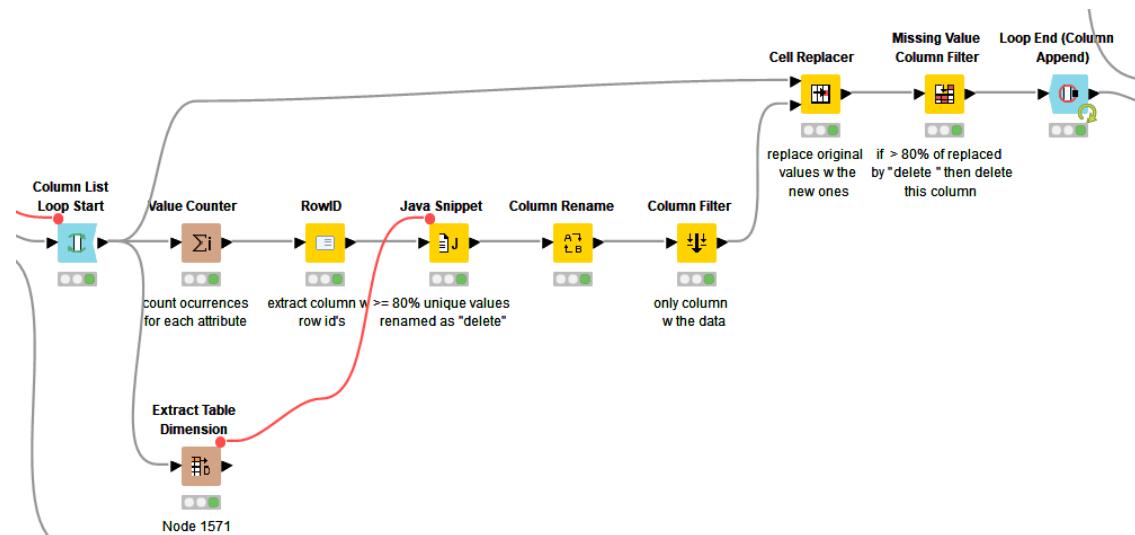


Figura 2.4: Vista previa de flujo KNIME para el filtrado de valores únicos

- Reemplazar con otros: en este nuevo componente, los valores únicos en una columna que representan una minoría, son reemplazados por la categoría 'other'. Para ello, iterando por cada columna dentro de un ciclo, se calculan la frecuencia absoluta y relativa de cada atributo (nodo *Math Formula*), y aquellos que representen menos del

1 % son los elegidos para la sustitución por la nueva categoría (nodo *Java Snippet*). En la figura 2.5 se muestra una parte del flujo KNIME donde se implementa esta tarea.

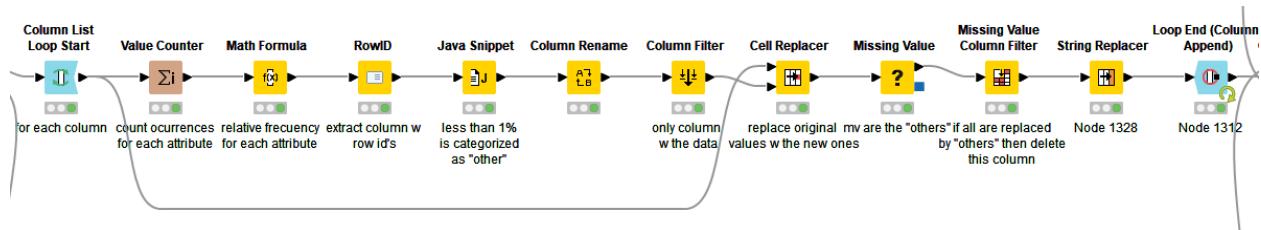


Figura 2.5: Vista previa de flujo KNIME para el reemplazo por 'other'

### 2.1.3. Manejo de valores faltantes

El tratamiento de valores faltantes en conjuntos de datos es un paso crítico en la preparación y análisis de datos. La presencia de datos faltantes puede afectar significativamente la calidad y la fiabilidad de cualquier análisis o modelo que se derive de ellos. Adicionalmente, algunos algoritmos requieren que no existan valores faltantes para su funcionamiento. En aras de mejorar la imputación de estos valores, se propone modificar el subcomponente *Valores faltantes*, cuyo diagrama de flujo se presenta en la figura 2.6.



Figura 2.6: Diagrama de flujo para el manejo de valores faltantes

La imputación de valores faltantes ha sido modificada, dado que anteriormente se sustituían por la media los valores faltantes numéricos, y por la moda los atributos categóricos. En la figura 2.7 se presenta el diagrama de flujo de la nueva implementación para la sustitución de estos valores.

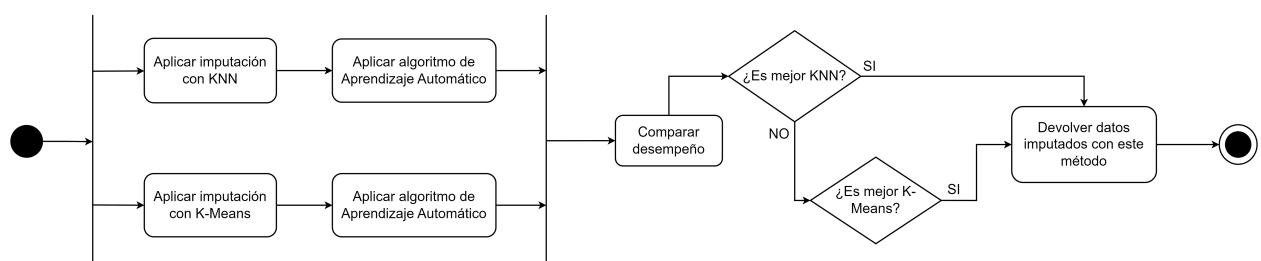


Figura 2.7: Diagrama de flujo para la imputación de valores faltantes

Primeramente se aplican en paralelo las técnicas de imputación kNN y k-Means, al terminar se aplica el algoritmo de aprendizaje automático en cuestión con los datos imputados por cada método y, finalmente, se comparan los resultados acorde al esquema descrito en el subcomponente *Discretizer* para devolver los datos con la mejor sustitución.

Para la implementación de kNN y k-Means, se crearon dos subcomponentes: kNN Imputation y kMI (K-Means Imputation). Se emplean los nodos *kNN* y *k-Means*, ambos nativos de KNIME. Para escoger  $k$ , en k-Means se utiliza el método de la silueta, ya que KNIME contiene el nodo *Silhouette Coefficient*; mientras que para kNN se decide emplear la raíz cuadrada de la muestra, debido a su simpleza en la implementación. Además, este método podría ofrecer cierta estabilidad en la elección del número de vecinos, independientemente del tamaño específico del conjunto de datos. Esto podría hacer que el modelo sea menos sensible a variaciones en el tamaño de la muestra. A continuación se describe el funcionamiento de k-NNI y kMI:

■ kMI:

1. Convertir los valores nominales en numéricos para que el nodo trabaje con ellos, ya que este algoritmo solamente trata este tipo de valores.
2. Separar las columnas con valores perdidos para que el algoritmo pueda realizar el proceso de clustering con los datos sin estos valores, ya que no los tolera el nodo.
3. Calcular valor óptimo de  $k$  con el método de la silueta.
4. Aplicar k-Means a estos datos.
5. Unir las columnas que contienen valores perdidos con los datos etiquetados con su respectivo clúster.
6. Aplicar el nodo *Missing Value*, nativo de KNIME, tras filtrar por clúster, sustituyendo los valores perdidos por la media de esa columna, es decir el valor del centroide de ese clúster.
7. Retornar las variables numéricas a categóricas (las que se transformaron inicialmente) para recuperar su valor original.

■ kNNI:

1. Convertir los valores nominales en numéricos para que el nodo trabaje con ellos, ya que este algoritmo solamente trata este tipo de valores.
2. Extraer los nombres de las columnas que contienen valores perdidas.
3. Por cada columna, se separan los valores perdidos del resto de los datos, éstos se reemplazan por 0 si son numéricos, si son de tipo *string* se ignoran.

4. Calcular el valor de  $k$  mediante la raíz cuadrada de la muestra.
5. Los datos sin valores perdidos se emplean para el entrenamiento de kNN, mientras los datos con estos se emplean para la predicción.
6. Retornar las variables numéricas a categóricas (las que se transformaron inicialmente) para recuperar su valor original.

En las figuras 2.8 y 2.9 se muestran los flujos de trabajo que se implementan para la imputación con los algoritmos kNNI y kMI, respectivamente.

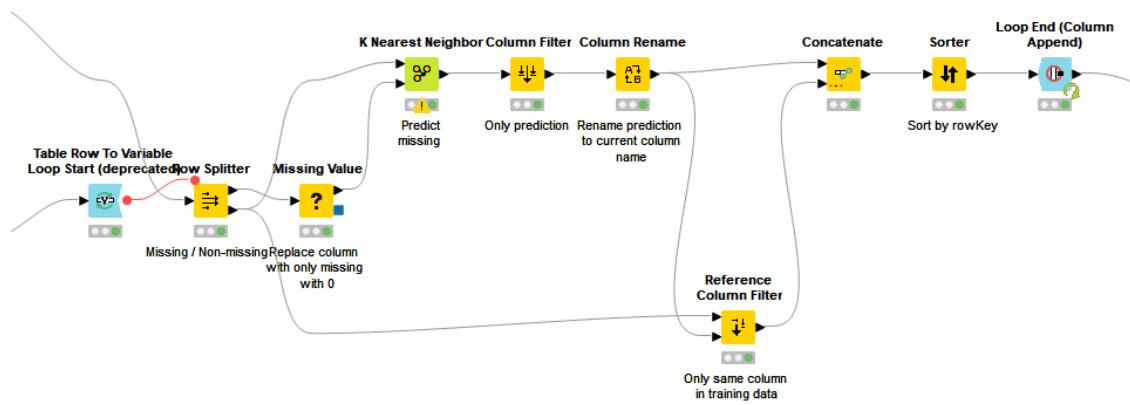


Figura 2.8: Vista previa de flujo KNIME para kNNI

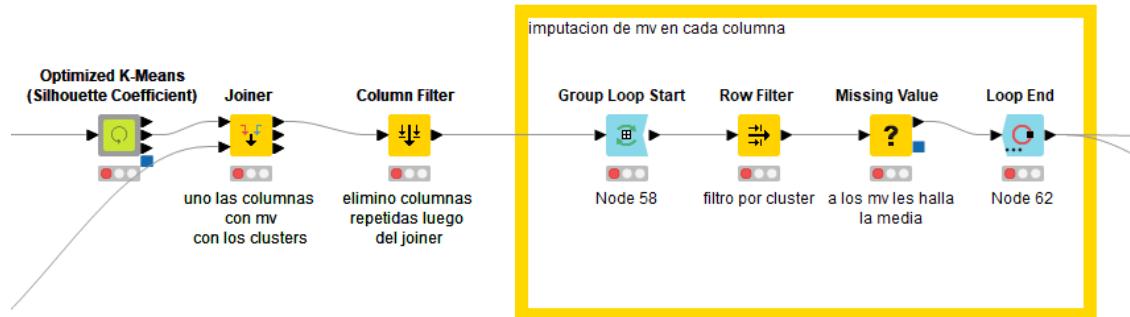


Figura 2.9: Vista previa de flujo KNIME para kMI

#### 2.1.4. Codificación y normalización

La codificación de variables categóricas implica asignar valores numéricos a estas categorías para que los algoritmos de aprendizaje automático puedan trabajar con ellas. Este es el propósito del subcomponente *Pre-procesar números*, además de la normalización de variables numéricas. En aras de clarificar el funcionamiento de este proceso, se decide renombrarlo a *Codificar y normalizar*.

La normalización son proporciones sin unidades de medida (adimensionales o invariantes de escala) que nos permiten poder comparar elementos de distintas variables y unidades de medida. Esta es necesaria para cambiar los valores de las columnas numéricas del conjunto de datos para usar una escala común, sin distorsionar las diferencias en los intervalos de valores ni perder información. La normalización es fundamental para que algunos algoritmos modelen los datos correctamente.

En KNIME es posible implementar la normalización a partir del nodo *Normalizer*. En este nodo se encuentran tres métodos para normalizar, a elección del usuario: *Decimal Scaling*, *Z-Score* y *Min-Max*. Este proceso se encuentra en el subcomponente para el pre-procesado de números, cuyo diagrama de flujo se presenta en la figura 2.10.

Por otro lado, los valores de alta cardinalidad, aquellos que se repiten con frecuencia y pueden ser numerosos, son cruciales para comprender tendencias y patrones en los datos. La codificación de estas variables se refiere a técnicas especiales de codificación que se utilizan cuando se tienen variables categóricas con un gran número de categorías o niveles distintos. La alta cardinalidad puede dificultar la gestión de estas variables en modelos de aprendizaje automático, y es importante abordarlas de manera eficiente.

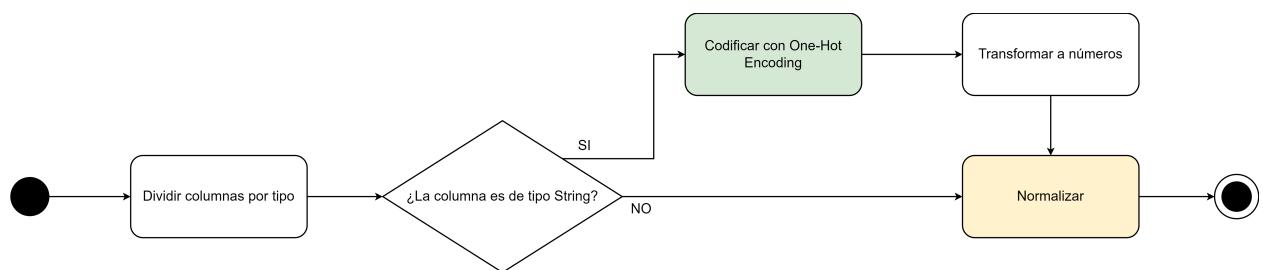


Figura 2.10: Diagrama de flujo del pre-procesado para la codificación y normalización

A continuación se exponen los cambios realizados al pre-procesado para la codificación y normalización:

- Codificar con One-Hot Encoding: para el tratamiento de valores con alta cardinalidad, se emplea la codificación One-Hot (ver epígrafe 1.3.1), ya que KNIME brinda el nodo *One To Many* para esta tarea. Para esto se escogen los atributos que tienen como mínimo 15 categorías distintas. One-Hot Encoding se utiliza para codificar variables categóricas en una forma que no impone un orden implícito en las categorías. Cuando se tienen más de 15 categorías, es poco probable que haya un orden natural o jerarquía en esas categorías, por lo que codificarlas como variables binarias evita interpretaciones erróneas de orden o importancia. Dado que esta técnica produce gran dimensionalidad, se utiliza para su reducción el método PCA, con un límite de conservación de la información del 90 %. Este proceso se encuentra implementado en la figura 2.11.

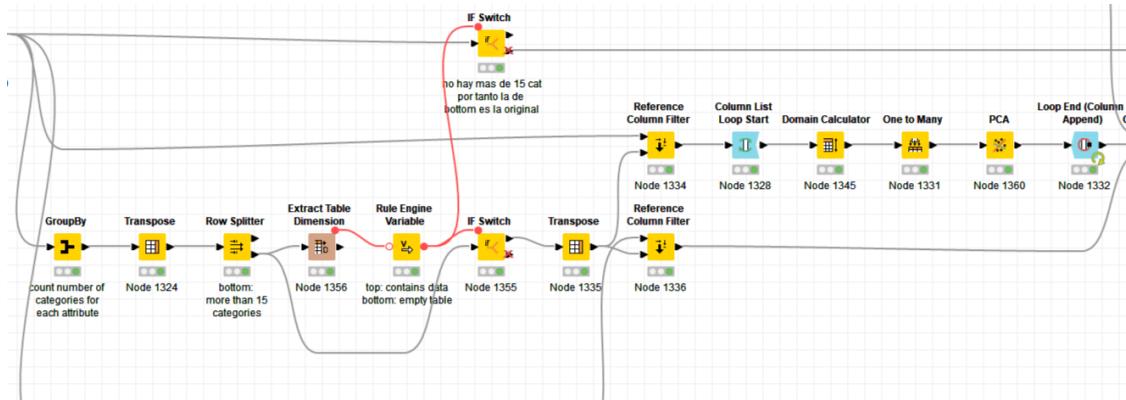


Figura 2.11: Vista previa de flujo KNIME para codificación One-Hot

- Normalizar: en aras de automatizar este proceso, se propone el subcomponente *Normalizer*, de igual nombre al nodo nativo de KNIME, en donde se encuentra el mismo para la ejecución de los métodos que contiene, en función de un modelo predeterminado. En la figura 2.12 se presenta el diagrama de flujo de este subcomponente.

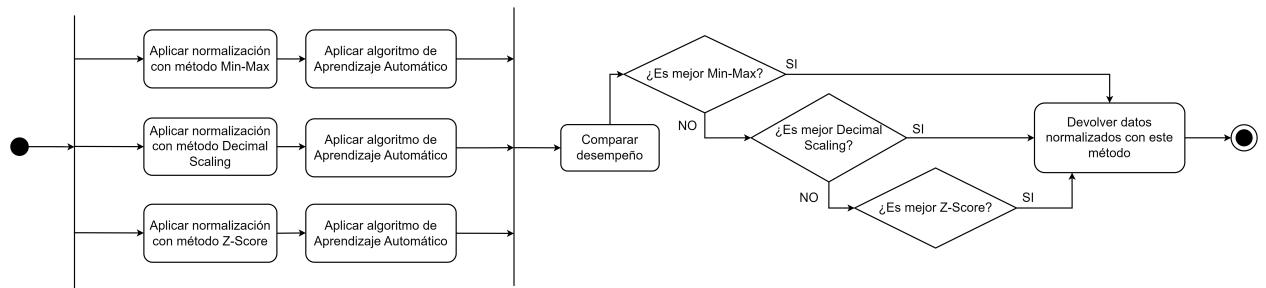


Figura 2.12: Diagrama de flujo para la normalización

Siguiendo el mismo esquema del subcomponente *Discretizer*, tras aplicar la normalización con los distintos métodos ofrecidos por la herramienta KNIME, se aplica el algoritmo de aprendizaje automático que requiere los datos normalizados y posteriormente, se realiza la evaluación del desempeño de cada uno, acorde a las métricas Precisión y Cohen's Kappa. Luego de escoger el mejor, se devuelve la tabla con los datos normalizados.

## 2.2. Componente AutoML Clasificación (Optimización de hiperparámetros)

Antes de explorar en detalle las adaptaciones realizadas en cada modelo, es fundamental presentar un nuevo componente para la optimización de hiperparámetros, con el objetivo de facilitar el entendimiento en los epígrafes posteriores.

Los modelos a optimizar en el componente propuesto en este acápite, se derivan del componente *AutoML Clasificación (pre-procesado)*, dichos modelos son: redes neuronales mediante retropropagación (RProp), redes neuronales probabilísticas (PNN) y máquinas de soporte vectorial (SVM). Es importante señalar que, a diferencia del componente *AutoML Clasificación (pre-procesado)*, no se incorporan los modelos ID3, CART y C4.5. Esto se debe a que no permiten la optimización de hiperparámetros, ya que son nodos de la extensión KNIME WEKA, que carecen de configuración de variables de flujo. Por esta razón, se ha tomado la decisión de implementar otra variante: Random Forest.

La selección adecuada de variables desempeña un papel fundamental en el desarrollo de modelos de aprendizaje automático y estadísticos. Las variables que se incluyen en un modelo no solo afectan su capacidad para comprender patrones y tomar decisiones precisas, sino que también pueden influir significativamente en la eficiencia computacional y los recursos requeridos. Al elegirlas correctamente, se simplifica y mejora la interpretación de los modelos, reduce el riesgo de sobreajuste y acelera el tiempo de entrenamiento. Algunas variables que influyen en el rendimiento de los modelos son:

- Random Forest: Número de árboles en el bosque, criterio de división, profundidad máxima, número mínimo de muestras requeridas para dividir un nodo, peso asignado a las clases en el conjunto de datos, número mínimo de muestras requeridas para estar en un nodo hoja.
- SVM: Tipo de kernel, número máximo de iteraciones, parámetro de regularización, pesos a las clases, sesgo de la ecuación del hiperplano.
- Redes Neuronales por Retropropagación: Tasa de aprendizaje, número de épocas, número de capas, cantidad de neuronas por capa, inicialización de pesos, regularización.
- Redes Neuronales Probabilísticas: Conexión sináptica entre dos neuronas, tasa de aprendizaje, número de épocas, número de capas y la cantidad de neuronas por capa.

Es importante destacar que la selección de variables debe basarse en un conocimiento sólido del problema en cuestión y en un análisis cuidadoso de su impacto en la calidad y eficacia del modelo. La selección de variables a optimizar por algoritmo se desarrolla en los epígrafes posteriores.

Ajustar las configuraciones de un modelo es clave para lograr un rendimiento óptimo, lo que a su vez mejora la precisión y la eficacia de sus análisis. En este contexto, se propone el componente *AutoML Clasificación (Optimización de hiperparámetros)*, presente en la figura 2.13, conteniendo la siguiente configuración:

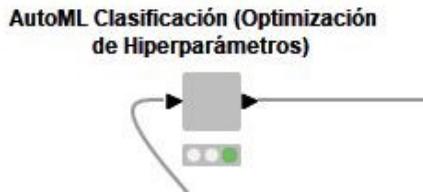


Figura 2.13: Componente *AutoML Clasificación (Optimización de Hiperparámetros)*

1. Puerto de entrada: recibe los datos de entrada en formato tabular.
2. Elementos de la configuración:
  - Selección de modelos: Lista de modelos a entrenar y optimizar disponibles (Redes Neuronales por Retropropagación, Redes Neuronales Probabilísticas, Random Forest y SVM).
  - Selección de porcentaje de la partición de entrenamiento: el valor introducido determina el factor en que se divide el conjunto de datos para entrenar y probar cada algoritmo, selección disponible en un rango de 1 a 99.
  - 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 selección múltiple de modelos con hiperparámetros optimizados.

### **2.2.1. 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 permitir seleccionar uno o varios de los algoritmos listados.
- RF5: El componente debe permitir seleccionar el porcentaje de la partición de entrenamiento.
- RF6: El componente debe entrenar y optimizar hiperparámetros para Redes Neuronales de Retropropagación.
- RF7: El componente debe entrenar y optimizar hiperparámetros para Redes Neuronales Probabilísticas.
- RF8: El componente debe entrenar y optimizar hiperparámetros para SVM.
- RF9: El componente debe entrenar y optimizar hiperparámetros para Random Forest.
- RF10: El componente debe graficar los modelos seleccionados.
- RF11: El componente debe retornar el modelo seleccionado por el usuario.

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 numéricos deben estar previamente normalizados.

### 2.2.2. Modelación del componente AutoML Clasificación (Optimización de Hiperparámetros)

El diagrama de flujo de la figura 2.14 expone el flujo general del componente *AutoML Clasificación (Optimización de Hiperparámetros)*.

En la fase inicial, se procede a la definición de la configuración de parámetros, durante la cual se especifican las variables determinantes que guiarán el flujo de ejecución del proceso. Luego, se procede con la optimización de los hiperparámetros de los modelos seleccionados por el usuario, lo que da como resultado la generación de los modelos que serán representados gráficamente. Según si la columna objetivo presenta múltiples clases o no, se realizan las representaciones gráficas de los modelos y se otorga al usuario la capacidad de elegir aquel que mejor concuerde con sus requisitos particulares.

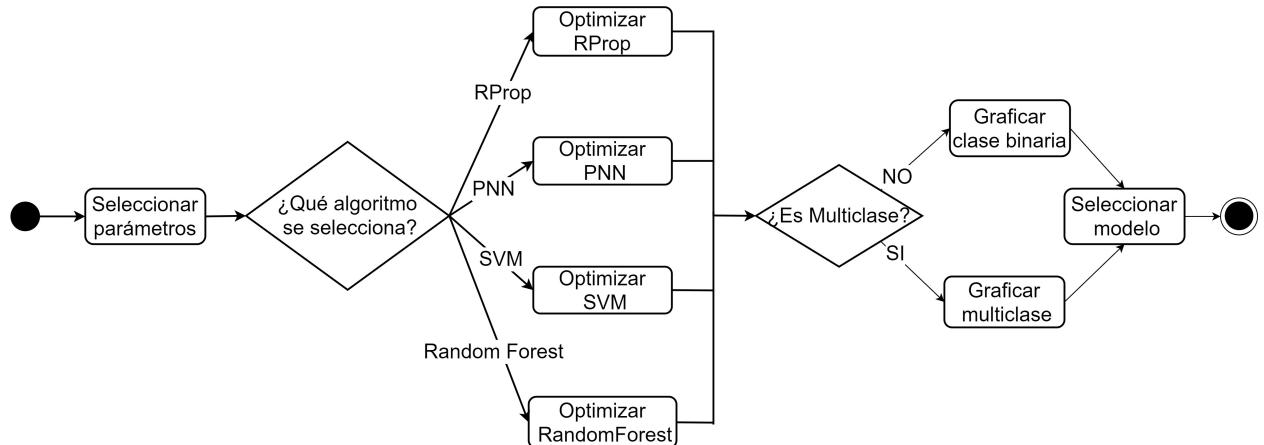


Figura 2.14: Diagrama de flujo general del componente *AutoML Clasificación (Optimización de Hiperparámetros)*

### 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 2.15 expone el flujo para la selección de parámetros.

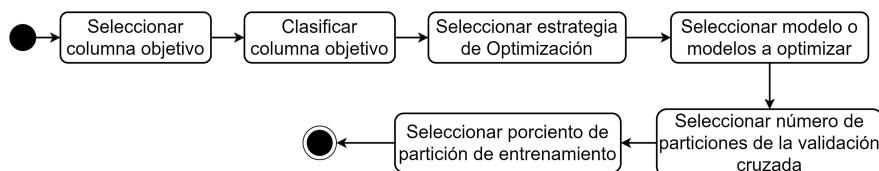


Figura 2.15: Diagrama de actividades de selección de parámetros

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*, 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*.
- Seleccionar estrategia de optimización de hiperparámetros: la selección de la estrategia se lleva a cabo empleando el nodo *Single Selection Configuration*. Devuelve la variable *strategy* con el valor seleccionado previamente.
- 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.* Este devuelve la variable de flujo resultante de la selección, en este caso presenta la configuración para limitar el rango entre 5 y 10. La selección de la partición de entrenamiento se lleva a cabo con el mismo nodo anteriormente mencionado.

- Evaluar columna objetivo: se evalúa en clase binaria o multiclas la columna objetivo mediante los nodos listados en la **fig \*\*\* (Foto para anexos)**
- Seleccionar modelo o modelos a optimizar: la selección se realiza mediante el nodo *Multiple Selection Configuration*, el cual devuelve la variable `modelo` con la selección.

## Optimización de hiperparámetros

El diagrama de flujo de la figura 2.16 expone el flujo general del apartado de optimización del componente *AutoML Clasificación (Optimización de Hiperparámetros)*.

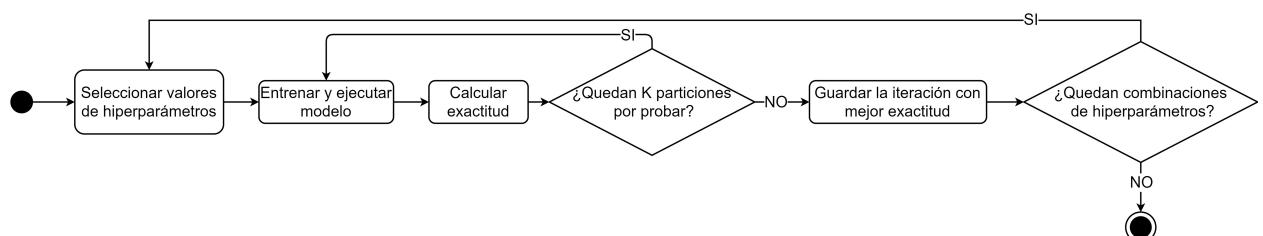


Figura 2.16: Diagrama de flujo de la optimización de hiperparámetros

En la figura 2.16, la optimización de hiperparámetros implica la iteración sistemática a través de un conjunto predefinido de hiperparámetros, evaluando exhaustivamente todas las combinaciones posibles. Este proceso se detiene una vez que se han evaluado todas las combinaciones. Cada iteración se somete a una validación cruzada, que implica la evaluación de los hiperparámetros en múltiples subconjuntos de prueba. La métrica de rendimiento (en este caso, la exactitud) se calcula para cada iteración de la validación cruzada, y se registra la iteración con la mayor exactitud. Finalmente, una vez que se han evaluado todas las combinaciones posibles, se devuelve la iteración del modelo que obtuvo la mejor exactitud.

En KNIME es posible implementar el ciclo para comprobar las combinaciones de hiperparámetros mediante los nodos *Parameter Optimization Loop Start* y *Parameter Optimization Loop End*, los cuales permiten almacenar las iteraciones realizadas por el algoritmo con las diferentes combinaciones de hiperparámetros dentro de un rango previamente establecido. En el **anexo \*\*\*\*** se muestra el flujo correspondiente al funcionamiento genérico de un flujo de optimización.

## Graficar y seleccionar modelos

Una parte esencial de cualquier flujo de trabajo en KNIME es la etapa en la que se exploran, comparan y seleccionan los modelos. Esta fase se convierte en el núcleo de la toma de decisiones en la analítica de datos, ya que permite identificar cuál de los modelos propuestos se ajusta de manera óptima a los datos y objetivos. KNIME brinda herramientas para visualizar y evaluar el rendimiento de los modelos, lo que permite tomar decisiones informadas y seleccionar el modelo que mejor se adapte a las necesidades del usuario. En el proceso de modelado y selección, se emplean los nodos *Binary Classification Inspector* y *ROC Curve* para problemas de clasificación binaria **fig\*** y, en cambio, se utilizan los nodos *Bar Chart* y *Table View* cuando se enfrentan problemas de clasificación multiclase **fig\***. (Las figuras de anexo q serian flujos)

### 2.2.3. Optimización de hiperparámetros para RProp

Para el entrenamiento y prueba de las Redes Neuronales por Retropropagación, se emplean los nodos *RProp MLP Learner* y *MultiLayerPerceptron Predictor*, donde debido a las limitaciones de KNIME y su importancia en el rendimiento del modelo se seleccionan los hiperparámetros a optimizar siguientes:

1. *Número máximo de iteraciones*: Generalmente llamado "número de épocas"(number of epochs), se define como un pase completo a través de todo el conjunto de datos durante el proceso de entrenamiento. Es un hiperparámetro crítico que controla cuántas veces la red pasará por el conjunto de datos de entrenamiento para ajustar sus pesos y mejorar su rendimiento. **Esta variable se comparte para el algoritmo Redes Neuronales Probabilísticas** (Montavon *et al.*, 2012).
2. *Cantidad de neuronas*: Número de neuronas o unidades en una capa específica de una red neuronal, afecta la capacidad de la red para aprender y representar patrones complejos en los datos (Montavon *et al.*, 2012).
3. *Número de capas*: Número de capas ocultas que se encuentran entre la capa de entrada y la capa de salida de la red. Estas se utilizan para aprender y representar patrones y características en los datos de entrada (Montavon *et al.*, 2012).

El diagrama de actividades de la figura 2.17, expone el flujo para el procesamiento necesario para la ejecución del algoritmo Redes Neuronales por Retropropagación, con optimización de hiperparámetros.

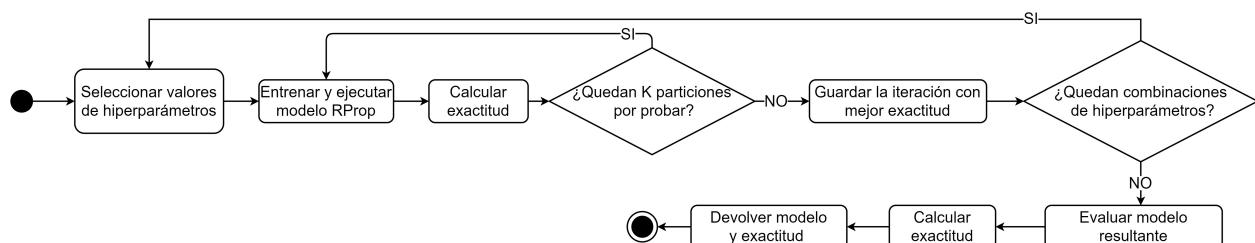


Figura 2.17: Diagrama de flujo para la optimización de hiperparámetros de RProp

#### 2.2.4. Optimización de hiperparámetros para PNN

En el contexto del algoritmo de Redes Neuronales Probabilísticas (PNN), se hacen uso de dos nodos: *PNN Learner* y *PNN Predictor*. Estos nodos son utilizados para configurar y ejecutar el algoritmo, y se enfocan en la selección y optimización de diversas variables críticas. En particular, debido a las limitaciones de KNIME se ajustan los siguientes hiperparámetros:

1. *Theta Minus ( $\theta$ -)*: Representa la disminución de la fuerza de una conexión sináptica entre dos neuronas. Si dos neuronas están activas simultáneamente con frecuencia baja, la conexión sináptica entre ellas disminuirá, lo que se conoce como "depresión sináptica". Se usa para evitar que las conexiones se fortalezcan en exceso y se vuelvan saturadas (Montavon *et al.*, 2012).
  2. *Theta Plus ( $\theta$  +)*: Representa el aumento de la fuerza de una conexión sináptica entre dos neuronas. Si dos neuronas están activas juntas con frecuencia alta, la conexión entre ellas se fortalecerá, lo que se conoce como "potenciación sináptica". Esto ayuda a fortalecer las conexiones que son relevantes (Montavon *et al.*, 2012).

Estos parámetros son esenciales para el rendimiento y la convergencia del algoritmo. El diagrama de actividades representado en la figura 2.18 proporciona una visualización estructurada del flujo de procesos y operaciones, necesarios para la ejecución del algoritmo de Redes Neuronales por Retropropagación con un enfoque específico en la optimización de hiperparámetros.

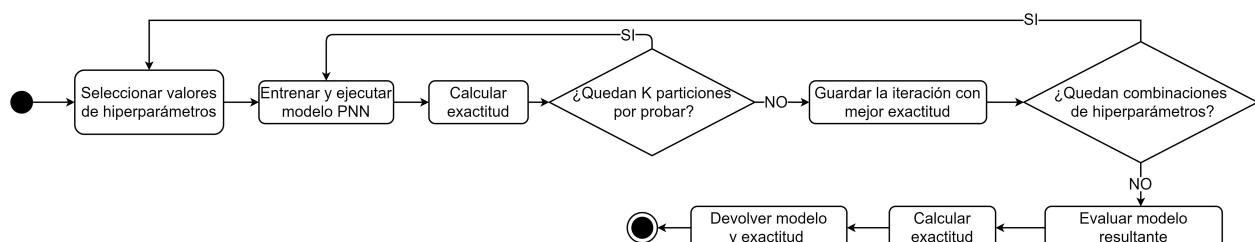


Figura 2.18: Diagrama de flujo para la optimización de hiperparámetros de PNN

## 2.2.5. Optimización de hiperparámetros para SVM

Para la Máquina de Soporte Vectorial (SVM), se realiza la selección de los hiperparámetros siguientes:

1. *Kernels*: función matemática que se utiliza para realizar una transformación no lineal de los datos de entrada. Permiten la clasificación efectiva de datos que no son linealmente separables en el espacio de características original. Mapean los datos a un espacio de características de mayor dimensión donde es más probable que sean linealmente separables (Scholkopf & Smola, 2018), (Bishop & Nasrabadi, 2006). En esta investigación se utilizan tres tipos de kernels (Scholkopf & Smola, 2018):
  - 1.1 *RBF*: Utiliza una función gaussiana para mapear los datos en un espacio de características de mayor dimensión, lo que es útil para problemas de clasificación no lineales.
  - 1.2 *Polinómico*: Transforma los datos utilizando funciones polinómicas, adecuado para problemas donde los datos pueden ser separados por una frontera polinómica.
  - 1.3 *Hiperbólico tangente*: Utiliza la función tangente hiperbólica para realizar la transformación no lineal de los datos.
2. *Bias*: sesgo de la ecuación del hiperplano de decisión, controla la posición del hiperplano y asegura que se ajuste adecuadamente entre las clases en un problema de clasificación (Scholkopf & Smola, 2018), (Joachims, 2002).
3. *Gamma*: El parámetro gamma controla la flexibilidad del modelo y la capacidad de ajustar los datos. Los valores bajos de gamma indican un gran radio de similitud, que da como resultado que se agrupen más puntos; mientras que los valores altos de gamma indican un radio de similitud más pequeño y una mayor complejidad del modelo (Bishop & Nasrabadi, 2006), (Scholkopf & Smola, 2018).

El proceso se encuentra representado en un diagrama de flujo, como se ilustra en la figura 2.19. Este diagrama de flujo proporciona una visión general de cómo se lleva a cabo la optimización de SVM, detallando el flujo de decisiones y operaciones involucradas en la selección del kernel y sus hiperparámetros correspondientes.

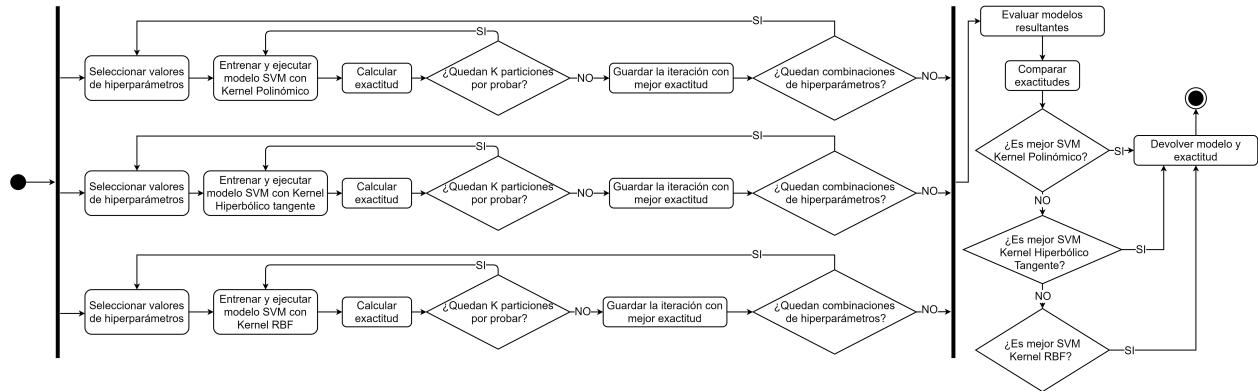


Figura 2.19: Diagrama de flujo para la optimización de hiperparámetros de SVM

## 2.2.6. Optimización de hiperparámetros para Random Forest

En el marco de la implementación de Random Forest, se hacen uso de los nodos *Random Forest Learner* y *Random Forest Predictor*, donde se seleccionan los hiperparámetros a optimizar:

1. *Profundidad máxima*: Es la profundidad de los árboles en el bosque. Controla la cantidad de niveles o divisiones que tiene desde el nodo raíz hasta las hojas y ayuda a evitar el sobreajuste (Lakshmanan *et al.*, 2021), (Hastie *et al.*, 2009).
2. *Cantidad de modelos*: Cantidad de árboles de decisión que se construyen en el bosque aleatorio. Cada árbol se entrena en una submuestra aleatoria del conjunto de datos de entrenamiento. Aumentar su valor generalmente hace que el modelo sea más robusto y preciso, pero también puede aumentar el costo computacional (Lakshmanan *et al.*, 2021).
3. *Tamaño mínimo del nodo*: Establece un límite en la cantidad mínima de ejemplos necesarios en un nodo para que se considere una división. Si el número de ejemplos en un nodo es menor que el valor especificado, no se realizará una división en ese nodo, lo que ayuda a evitar una partición excesiva y a reducir la complejidad del árbol (Lakshmanan *et al.*, 2021).
4. *Criterio de división*: medida utilizada para evaluar la calidad de una partición en un nodo del árbol. Determina cómo se eligen las características para dividir los nodos. En la presente investigación se tratan tres criterios de división (Gupta *et al.*, 2017):
  - 4.1 Ganancia de Información (Information Gain): evalúa cómo una división particular afecta la entropía o la impureza del conjunto de datos. Al seleccionar la característica que maximiza la ganancia de información, se elige la división que

proporciona la mayor claridad en términos de la distribución de las clases en los nodos hijos.

- 4.2 Gini Index: cuantifica qué tan a menudo un elemento seleccionado al azar sería incorrectamente etiquetado. Minimizar su valor durante la construcción del árbol lleva a la creación de nodos donde la probabilidad de error de clasificación es más baja.
- 4.3 Radio de Ganancia de Información (Information Gain Ratio): ajusta la ganancia de información dividiéndola por la información intrínseca de la característica. Esto promueve la selección de características que no solo ofrecen alta ganancia de información, sino que también tienen información intrínseca moderada.

La figura 2.20 brinda una representación organizada del flujo de procesos y operaciones necesarias para la ejecución de la optimización del algoritmo Random Forest, detallando la selección del criterio de división de Random Forest.

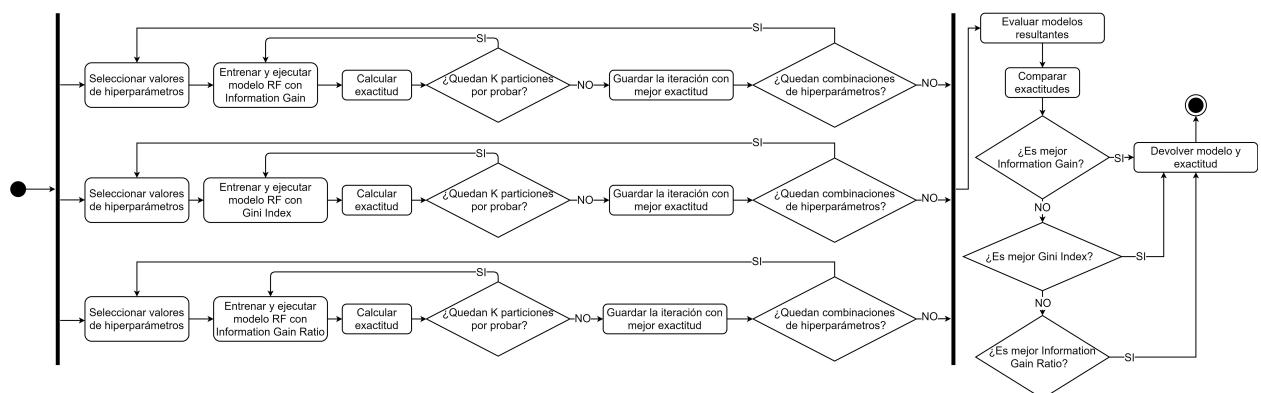


Figura 2.20: Diagrama de flujo para la optimización de hiperparámetros de Random Forest

## 2.3. Modelación de nueva versión del componente AutoML Clasificación

El enfoque principal de esta propuesta es la integración de los subcomponentes desarrollados para el pre-procesado y el componente *AutoML Clasificación (Optimización de hiperparámetros)* con el componente AutoML existente para clasificación, cuyo diagrama de flujo se presenta en la figura 2.21.

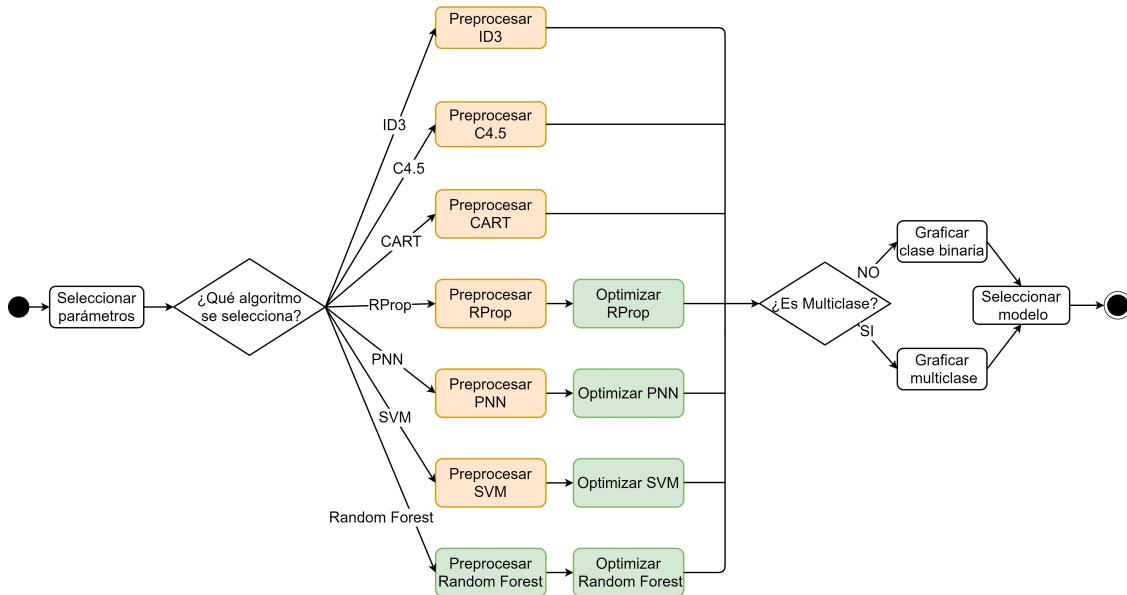


Figura 2.21: Diagrama de flujo de Componente AutoML (clasificación)

Como se puede apreciar en la figura 2.21, se implementa un nuevo modelo para la clasificación: Random Forest. Además, se realizan cambios en la personalización del componente: la eliminación del umbral de valores únicos por columna y un nuevo campo para la elección de la estrategia de optimización que se utilizará para la optimización de hiperparámetros. Con el propósito de abordar la integración con las nuevas implementaciones, se discutirán detalladamente en las secciones siguientes.

### 2.3.1. Procesado de ID3

El algoritmo ID3 se puede ejecutar en KNIME a través del nodo *Id3 (3.7)*, el cual forma parte de una extensión de la herramienta Weka. Al no ser un nodo nativo en KNIME, no es posible optimizar los hiperparámetros del mismo, sin embargo se realizaron las integraciones con los subcomponentes de pre-procesado discutidos en las secciones anteriores. Esto se manifiesta en el diagrama de flujo presente en la figura 2.22.

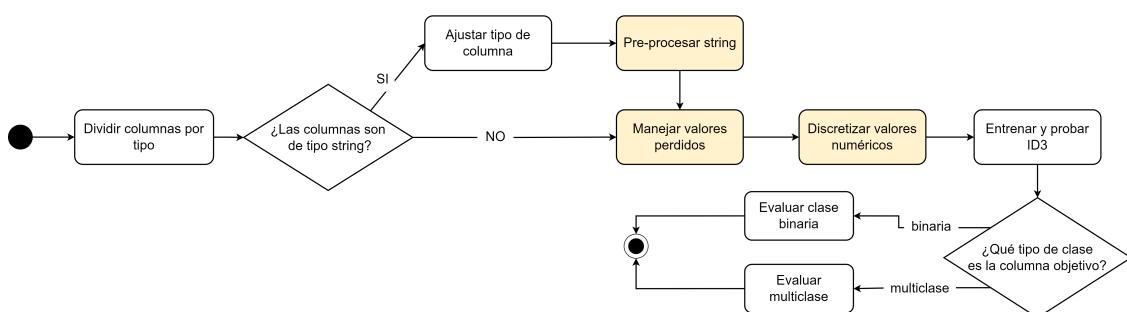


Figura 2.22: Diagrama de flujo del procesamiento del modelo ID3

Tal como se puede apreciar en la figura 2.22, simplemente se realizaron las modificaciones pertinentes en el manejo de valores faltantes, la discretización y el pre-procesado de *string*. No se incorpora la codificación y normalización, dado que este algoritmo no tolera variables numéricas.

### 2.3.2. Procesado para C4.5

El algoritmo C4.5 puede ejecutar en KNIME a través del nodo *J48*. Este, al igual que Id3, forma parte de la extensión Weka y no permite la incorporación de la optimización de hiperparámetros. Las modificaciones efectuadas en el pre-procesado están presentes en el diagrama de flujo de la figura 2.23.

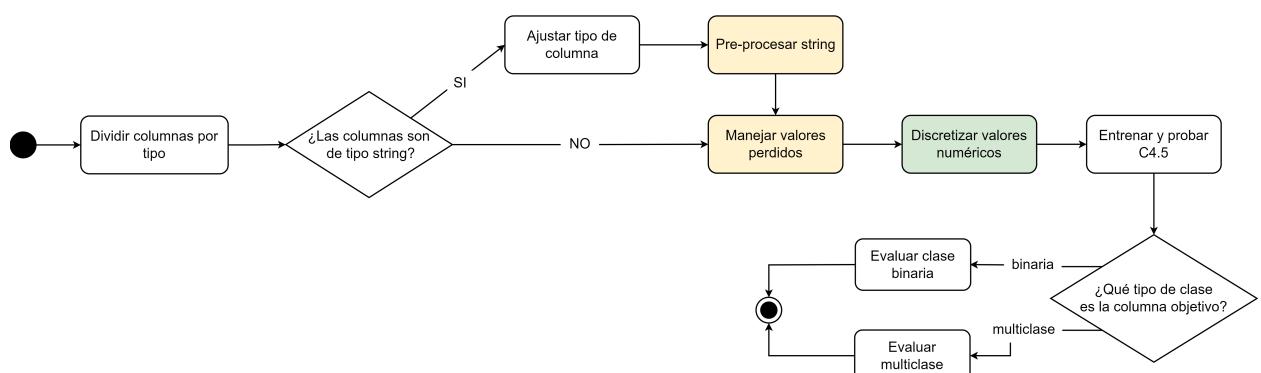


Figura 2.23: Diagrama de flujo del procesamiento del modelo C4.5

Como se observa, estas modificaciones fueron las mismas que a su predecesor, con la diferencia de que, en la versión anterior del componente *AutoML Clasificación (pre-procesado)*, en este modelo no estaba presente la discretización, ya que es un algoritmo versátil y puede manejar tanto variables categóricas como numéricas. Sin embargo, se decide incorporarla ya que, a pesar de lo anteriormente expuesto, C4.5 puede ser menos eficaz cuando se utiliza en conjuntos de datos con datos numéricos muy dispersos. En tales casos, los árboles de decisión pueden requerir una mayor profundidad para capturar patrones en las variables numéricas, lo que puede llevar a árboles más complejos y propensos al sobreajuste.

### 2.3.3. Procesado para CART

El algoritmo CART tiene los mismos requisitos que C4.5 para su ejecución, por lo que la diferencia entre ambos es el nodo que ejecuta el algoritmo, pues emplea el nodo *SimpleCart* (3.7). La única diferencia es que no se emplea la discretización de variables numéricas, ya que este árbol de decisión puede ser empleado tanto en tareas de clasificación como de regresión, lo que provoca que funcione de manera efectiva tanto con datos numéricos como nominales. Por otra parte, al formar parte de la extensión Weka, este nodo tampoco

presenta la optimización de hiperparámetros.

En la figura 2.24 se presenta el diagrama de flujo del procesado del algoritmo CART.

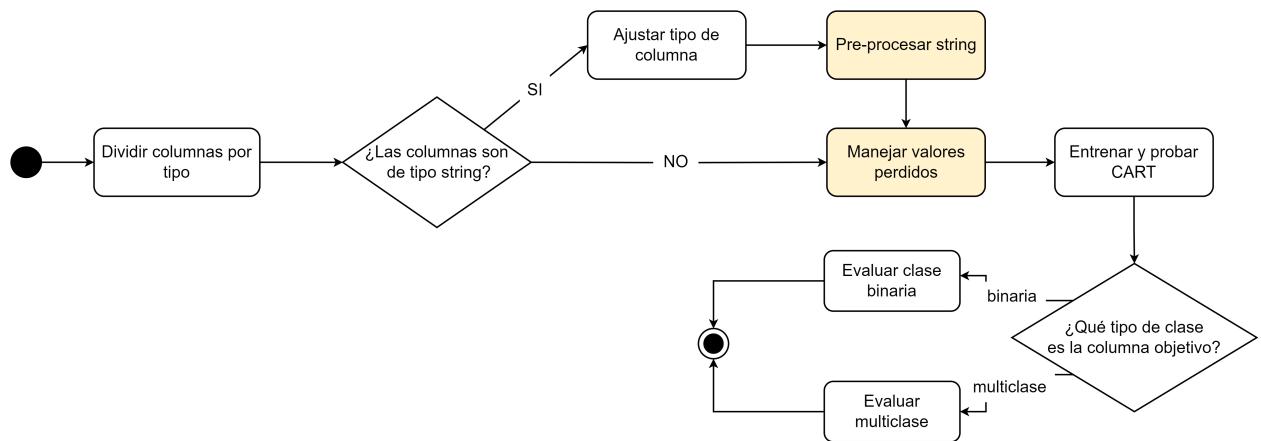


Figura 2.24: Diagrama de flujo del procesado de CART

### 2.3.4. Procesamiento para Random Forest

Dado que los modelos de árboles de decisión empleados en el componente *AutoML Clasificación (pre-procesado)* forman parte de extensiones Weka y, por tal motivo, no se puede ejecutar la optimización de hiperparámetros, se decide agregar un nuevo modelo: Random Forest. Este algoritmo puede trabajar con una amplia variedad de tipos de datos, ya sean datos numéricos o categóricos. Esto hace que Random Forest sea una elección versátil para tareas de clasificación y regresión. Por ello, el pre-procesado de este algoritmo es igual al de CART.

El diagrama de flujo del procesado de este nuevo modelo se muestra en la figura 2.25.

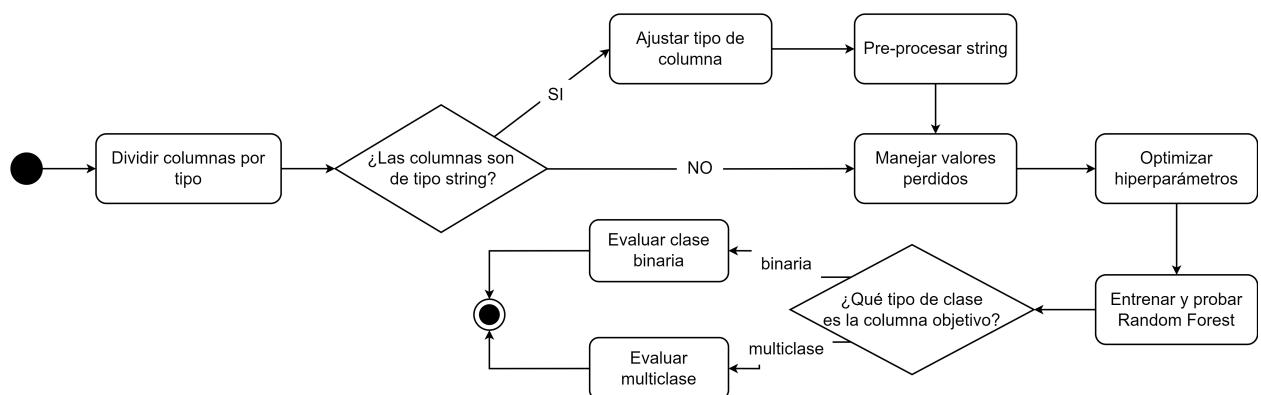


Figura 2.25: Diagrama de flujo del procesamiento de Random Forest

### 2.3.5. Procesamiento para Redes Neuronales por Retropropagación

Las Redes Neuronales por Retropropagación necesitan procesar los valores numéricos, además de los procesamientos realizados para C4.5 y CART, dado que solo permiten atributos de ese tipo. Por esta razón, se incluye el subcomponente para la codificación y normalización, tal como se muestra en el diagrama de flujo de la figura 2.26. Este algoritmo tiene la restricción de que solo se admiten valores entre 0 y 1, por tanto solo se realizan las normalizaciones Min-Max y Decimal Scaling. Por otra parte, también se implementa la optimización de hiperparámetros al integrar el componente *AutoML Clasificación (Optimización de hiperparámetros)*.

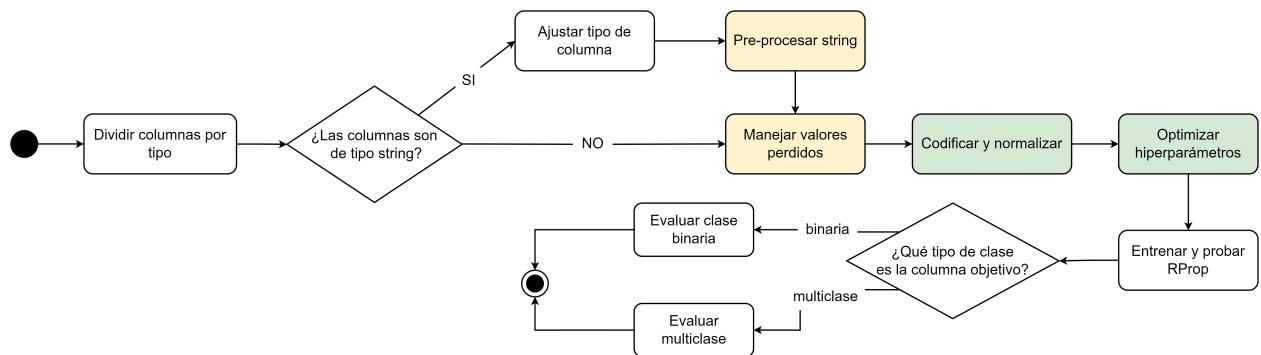


Figura 2.26: Diagrama de flujo del procesamiento de RProp

### 2.3.6. Procesamiento para Redes Neuronales Probabilísticas

Las Redes Neuronales Probabilísticas, al igual que RProp, trabajan con atributos numéricos. Por ello, se decide incorporar la codificación y normalización, la cual estaba en la versión anterior, sin embargo la normalización no se encontraba. Como en RProp, se implementa la optimización de hiperparámetros al integrar el componente *AutoML Clasificación (Optimización de hiperparámetros)*. En la figura 2.27 se muestra el diagrama de flujo de procesamiento de PNN.

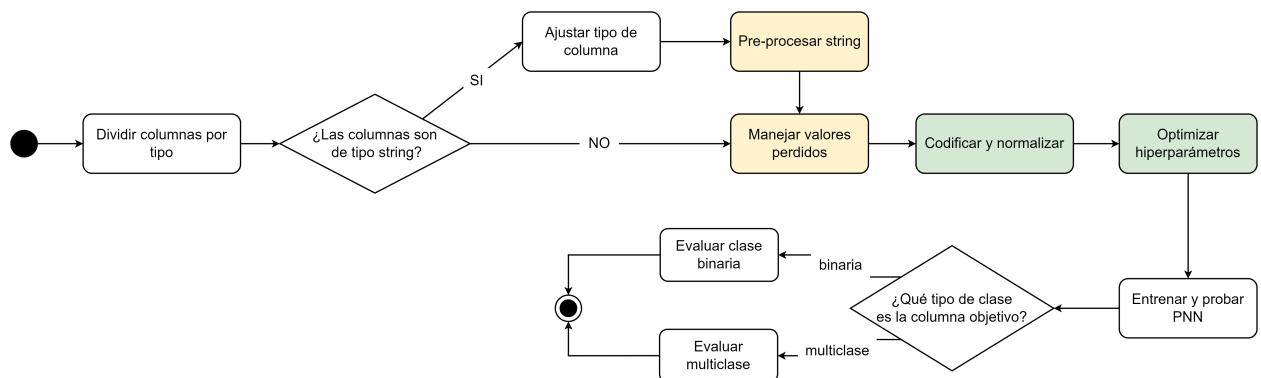


Figura 2.27: Diagrama de flujo para el procesado de PNN

### 2.3.7. Procesamiento para SVM

En este algoritmo, al igual que en las redes neuronales, se trabaja con atributos numéricos. No obstante, como fue el caso de PNN, no se encontraba la normalización de variables numéricas, que a pesar de no ser un requisito por el algoritmo, es un paso clave para su desempeño. Por otra parte, se implementa la optimización de hiperparámetros al integrar el componente *AutoML Clasificación (Optimización de hiperparámetros)*. El diagrama de flujo para el procesamiento de SVM se muestra en la figura 2.28.

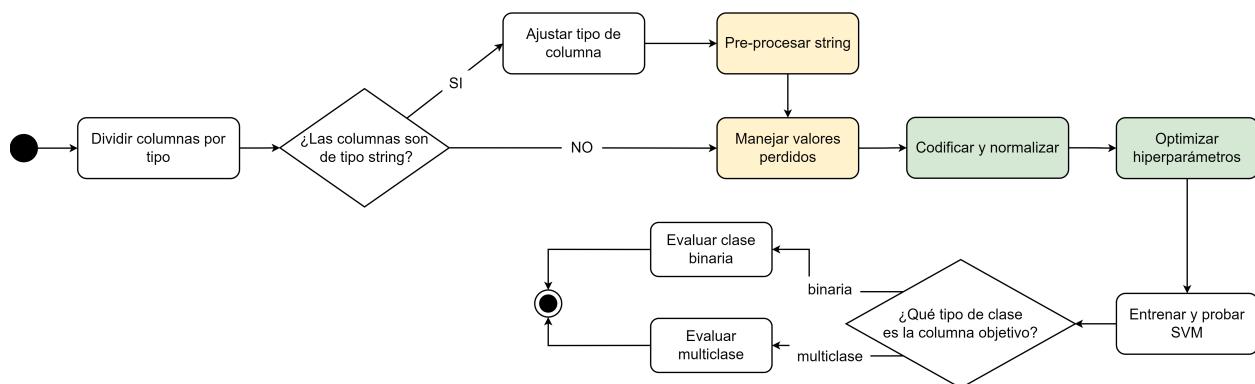


Figura 2.28: Diagrama de flujo del procesamiento de SVM

## 2.4. Conclusiones parciales

En el Capítulo 2, se presentaron una serie de modificaciones destinadas a mejorar el componente de *AutoML Clasificación (pre-procesado)*. Estas modificaciones se centraron en el pre-procesamiento de datos y la optimización de hiperparámetros, con el objetivo de crear un sistema más eficiente y preciso para la construcción de modelos de clasificación. El enfoque principal de dichas propuestas fue la integración de estas mejoras con el componente AutoML existente, con el fin de proporcionar modelos de clasificación adaptados a las necesidades específicas de los datos.

A partir de lo analizado en este capítulo, se arriba a las siguientes conclusiones:

- Se obtiene el diseño de un subcomponente para la discretización de variables numéricas.
- Para la discretización, se implementan los métodos Equal-Width, Equal-Frequency, Quantile-Based y CAIM.
- La implementación de los métodos de discretización se realiza con los nodos nativos de KNIME *Auto-Binner* y *CAIM Binner*.

- Se obtiene el diseño del componente AutoML Clasificación (Optimización de hiperparámetros).
- Se describe el uso y los diferentes requisitos y restricciones del componente AutoML Clasificación (Optimización de hiperparámetros).
- Se define el rango de hiperparámetros (Iteraciones, capas y número de neuronas) del algoritmo Rprop.

# **Capítulo 3**

## **Validación de Soluciones Propuestas al Componente de AutoML**

El presente capítulo se adentra en la fase crucial de validación de las soluciones propuestas en el capítulo 2. Durante el transcurso de este, se llevará a cabo una evaluación exhaustiva de las soluciones propuestas, centrándose en la comparación de los resultados obtenidos con y sin estas modificaciones. Se analizarán métricas de rendimiento, tiempos de ejecución y la capacidad del sistema para adaptarse a diferentes conjuntos de datos y necesidades específicas de clasificación. En última instancia, este capítulo constituirá un pilar fundamental en la evaluación de las contribuciones presentadas en esta tesis, demostrando la efectividad y utilidad de las soluciones propuestas en el contexto del AutoML para la clasificación.

### **3.1. Pruebas de caja negra a subcomponentes para el pre-procesado**

En pos de validar el correcto funcionamiento de las soluciones propuestas en el capítulo 2, a continuación se presentan las pruebas realizadas a cada componente y, por último, a la integración de los mismos con el componente *AutoML Clasificación (pre-procesado)*. Para ello se emplean las bases de datos descritas en la tabla 1.2.

#### **3.1.1. Caso de prueba al subcomponente *Discretizer***

Para la realización de esta prueba se diseña el caso de prueba de la tabla 3.1. Para esta prueba en particular se utiliza la base de datos CANCER DATA y el modelo ID3. Se emplea esta base de datos dado que la mayoría de sus atributos son numéricos, por tanto todos serán discretizados.

Tabla 3.1: Caso de prueba al componente *Discretizer*

Caso de prueba			
Objetivo de la prueba	Comprobar la efectividad de las transformaciones al discretizar las variables numéricas		
Descripción de la prueba	Se debe proporcionar una tabla con variables numéricas al componente <i>Discretizer</i> y evaluar la tabla resultante		
Condiciones	1. Debe estar presente la columna objetivo para la clasificación. 2. La columna objetivo debe ser de tipo nominal.		
Combinaciones de valores de entrada			
CP	Escenario	Resultado esperado	Resultado real
CP1	Se proporciona una tabla con atributos numéricos	Se comparan los discretizadores acorde al algoritmo de ML	Se comparan los discretizadores acorde al algoritmo de ML
CP2	Se proporciona una tabla con atributos numéricos	Se devuelven los datos discretizados acorde a los resultados del CP1	Se devuelven los datos discretizados acorde a los resultados del CP1

En la figura 3.1 se presentan los resultados del CP1, donde el mejor discretizador resulta ser CAIM con 0.953 de precisión y 0.903 de Cohen's Kappa, siendo unos resultados notables para una base de datos desbalanceada. Para el CP2, se presenta en la figura 3.2 una vista previa de la tabla de salida del componente *Discretizer* con los datos discretizados.

Row ID	Accuracy	Cohen'...	rank
id3-caim	0.953	0.903	1
id3-quantile	0.935	0.857	2
id3-width	0.907	0.796	3
id3-freq	0.857	0.687	4

Figura 3.1: Resultados del CP1 de *Discretizer*

Outport 1 - 5:2 - Discretizer											
File Edit Hilita Navigation View											
Table "default" - Rows: 569 Spec - Columns: 31 Properties Flow Variables											
Row ID	diagnosis	radius...	texture...	perimet...	area_m...	smooth...	compact...	concav...	concav...	symmet...	
Row0	M	Interval_1	Interval_0	Interval_1							
Row1	M	Interval_1	Interval_0	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_0	
Row2	M	Interval_1									
Row3	M	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	
Row4	M	Interval_1	Interval_0	Interval_1	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_0	
Row5	M	Interval_0	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	
Row6	M	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_0	
Row7	M	Interval_0	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	
Row8	M	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	
Row9	M	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	
Row10	M	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	
Row11	M	Interval_1	Interval_0	Interval_1	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_0	
Row12	M	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	
Row13	M	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_0	
Row14	M	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	
Row15	M	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	

Figura 3.2: Vista previa de la salida del componente *Discretizer*

### 3.1.2. Caso de prueba al subcomponente *String preprocs*

Para la realización de esta prueba se diseña el caso de prueba de la tabla 3.2. Para esta prueba en particular se crea la base de datos de la figura 3.3, dado que solamente se realizan transformaciones en los datos. La columna 1 se emplea para el CP1, así como la columna 2 para el CP2. La columna 3 es utilizada para comprobar el correcto funcionamiento del componente con una columna objetivo.

Tabla 3.2: Caso de prueba al componente *String preprocs*

Caso de prueba			
Objetivo de la prueba	Comprobar la efectividad de las transformaciones a las columnas de tipo <i>string</i> y evaluar la tabla resultante		
Descripción de la prueba	Se debe proporcionar una tabla con atributos con valores únicos		
Condiciones	1. Solo se deben proporcionar columnas de tipo <i>string</i> 2. Debe estar presente una columna objetivo nominal para la clasificación		
Combinaciones de valores de entrada			
CP	Escenario	Resultado esperado	Resultado real
CP1	Se proporciona una columna con más del 80 % de valores únicos	La columna es eliminada	La columna es eliminada
CP2	Se proporciona una columna con varias categorías diferentes donde existan valores únicos	Se reemplazan los valores únicos por la categoría 'other'	Se reemplazan los valores únicos por la categoría 'other'

Row ID	column1	column2	column3
Row0	a	a	si
Row1	b	a	si
Row2	c	a	si
Row3	d	a	si
Row4	e	a	si
Row5	f	a	si
Row6	g	a	si
Row7	h	a	si
Row8	i	a	no
Row9	j	a	si
Row10	k	a	no
Row11	l	b	no
Row12	m	b	si
Row13	n	b	no
Row14	o	b	si
Row15	p	b	no
Row16	q	b	si
Row17	r	b	no
Row18	s	b	si
Row19	t	b	si
Row20	u	b	no
Row21	v	b	no
Row22	w	b	no
Row23	x	c	no
Row24	y	c	no
Row25	z	c	si
Row26	aa	d	no
Row27	bb	e	no
Row28	cc	f	si
Row29	dd	g	si

Figura 3.3: Base de datos empleada para las pruebas del componente String-preprocs

En la figura 3.4a se muestra el resultado del CP1, con el objetivo de filtrar los valores únicos. Como se observa, la columna 1 es eliminada. Por otra parte, en la figura 3.4b se

muestra el resultado del CP2, con el objetivo de sustituir los valores únicos de una columna por la categoría 'other'. Se puede observar que en la columna 2, los últimos valores que anteriormente eran únicos, ahora fueron sustituidos.

Row ID	column3	column2
Row0	si	a
Row1	si	a
Row2	si	a
Row3	si	a
Row4	si	a
Row5	si	a
Row6	si	a
Row7	si	a
Row8	no	a
Row9	si	a
Row10	no	a
Row11	no	b
Row12	si	b
Row13	no	b
Row14	si	b
Row15	no	b
Row16	si	b
Row17	no	b
Row18	si	b
Row19	si	b
Row20	no	b
Row21	no	b
Row22	no	b
Row23	no	c
Row24	no	c
Row25	si	c
Row26	no	d
Row27	no	e
Row28	si	f
Row29	si	g

(a) Resultado del CP1, para la eliminación de valores únicos

Row ID	column2	column3
Row0	a	si
Row1	a	si
Row2	a	si
Row3	a	si
Row4	a	si
Row5	a	si
Row6	a	si
Row7	a	si
Row8	a	no
Row9	a	si
Row10	a	no
Row11	b	no
Row12	b	si
Row13	b	no
Row14	b	si
Row15	b	no
Row16	b	si
Row17	b	no
Row18	b	si
Row19	b	si
Row20	b	no
Row21	b	no
Row22	b	no
Row23	c	no
Row24	c	no
Row25	c	si
Row26	other	no
Row27	other	no
Row28	other	si
Row29	other	si

(b) Resultado del CP2, para la sustitución por la categoría 'other'

Figura 3.4: Resultados de los casos de prueba del componente *String preprocs*

### 3.1.3. Caso de prueba al subcomponente *MV Imputation*

Para la realización de esta prueba se diseña el caso de prueba de la tabla 3.3. Para esta prueba se emplea la base de datos CENSUS INCOME y el algoritmo C4.5. Se escoge esta base de datos

Tabla 3.3: Caso de prueba al componente *MV Imputation*

Caso de prueba			
Objetivo de la prueba	Comprobar la efectividad en el tratamiento de valores faltantes en una tabla		
Descripción de la prueba	Se debe proporcionar una tabla con valores perdidos al componente <i>MV Imputation</i> y evaluar la tabla resultante		
Condiciones	1. Debe estar presente la columna objetivo para la clasificación. 2. La columna objetivo debe ser de tipo nominal.		
Combinaciones de valores de entrada			
CP	Escenario	Resultado esperado	Resultado real
CP1	Se proporciona una tabla con valores faltantes	Se comparan los métodos de imputación acorde al algoritmo de ML	Se comparan los métodos de imputación acorde al algoritmo de ML
CP2	Se proporciona una tabla con valores faltantes	Se devuelven los valores imputados acorde a los resultados del CP1	Se devuelven los valores imputados acorde a los resultados del CP1

En la figura 3.5 se muestran los resultados del CP1, donde el mejor método de imputación para esta base de datos con este algoritmo es KMI, con una precisión de 0.848 y Cohen's Kappa de 0.557. Para el CP2 se presenta en la figura 3.6 una vista previa de la tabla de salida del componente *MV Imputation* con los datos imputados.

Row ID	Accuracy	Cohen'...	rank
kmi	0.848	0.557	1
knni	0.847	0.545	2

Figura 3.5: Resultados del CP1 del componente *MV Imputation*

Row ID	S Column3	S Column5	S Column7	S Column8	S Column9	S Column14	S Column1	S Column6	S Column13	D Column2	D Column4	D Column10	D Column11	D Column12	
Row0	bachelors	never-married	not-in-family	white	male	<=50k	state-gov	adm-clerical	united-states	39	77,516	13	2,174	0	40
Row1	bachelors	married-civ...	husband	white	male	<=50k	self-emp-no...	exec-manag...	united-states	50	83,311	13	0	0	13
Row2	masters	never-married	not-in-family	white	female	>50k	private	prof-specialty	united-states	31	45,781	14	14,084	0	50
Row3	bachelors	never-married	own-child	white	female	<=50k	private	adm-clerical	united-states	23	122,272	13	0	0	30
Row4	assoc-voc	married-civ...	husband	asian-pac-isl...	male	>50k	private	craft-repair	cuba	40	121,772	11	0	0	40
Row5	11th	married-civ...	husband	white	male	<=50k	private	sales	united-states	38	28,887	7	0	0	50
Row6	9th	married-civ...	husband	black	male	<=50k	federal-gov	farming-fishing	united-states	35	76,845	5	0	0	40
Row7	11th	married-civ...	husband	white	male	<=50k	private	transport-m...	united-states	43	117,037	7	0	0	2,042
Row8	hs-grad	divorced	unmarried	white	female	<=50k	private	tech-support	united-states	59	109,015	9	0	0	40
Row9	some-college	married-civ...	own-child	white	male	<50k	federal-gov	adm-clerical	united-states	30	59,951	10	0	0	40
Row10	some-college	married-civ...	husband	white	male	>50k	private	sales	cuba	31	84,154	10	0	0	38
Row11	bachelors	married-civ...	husband	white	male	<50k	self-emp-no...	prof-specialty	united-states	53	88,505	13	0	0	40
Row12	hs-grad	separated	unmarried	white	female	<50k	private	adm-clerical	united-states	49	94,638	9	0	0	40
Row13	assoc-voc	married-civ...	husband	white	male	<50k	state-gov	craft-repair	united-states	41	101,603	11	0	0	40
Row14	some-college	married-civ...	wife	other	female	<50k	private	exec-manag...	united-states	25	20,797	10	0	0	40
Row15	prof-school	married-civ...	wife	white	female	>50k	private	prof-specialty	united-states	47	51,835	15	0	1,902	60
Row16	hs-grad	divorced	not-in-family	white	male	<=50k	self-emp-inc	exec-manag...	united-states	47	109,832	9	0	0	60
Row17	assoc-voc	married-civ...	husband	white	male	<50k	private	other-service	puerto-rico	35	56,352	11	0	0	40
Row18	bachelors	married-civ...	husband	white	male	<50k	private	sales	united-states	30	59,496	13	2,407	0	40
Row19	doctorate	married-civ...	husband	white	male	>50k	private	prof-specialty	united-states	42	116,632	16	0	0	45
Row20	some-college	divorced	not-in-family	white	male	<=50k	private	tech-support	united-states	29	105,598	10	0	0	58

Figura 3.6: Resultado del CP2 del componente *MV Imputation*

### 3.1.4. Caso de prueba al subcomponente *Codificar y normalizar*

Para la realización de esta prueba se diseña el caso de prueba de la tabla 3.4. Para esta prueba se emplea la base de datos HUMAN RESOURCES y el algoritmo SVM.

Tabla 3.4: Caso de prueba al componente *Codificar y normalizar*

Caso de prueba			
Objetivo de la prueba	Comprobar la efectividad de la codificación y la normalización en una tabla		
Descripción de la prueba	Se debe proporcionar una tabla con valores nominales de alta cardinalidad y numéricos al componente <i>Codificar y normalizar</i> y evaluar la tabla resultante.		
Condiciones	1. Debe estar presente la columna objetivo para la clasificación. 2. La columna objetivo debe ser de tipo nominal. 3. Debe haber una columna con más de 15 categorías diferentes.		
Combinaciones de valores de entrada			
CP	Escenario	Resultado esperado	Resultado real
CP1	Se proporciona una tabla con valores numéricos	Se comparan los métodos de normalización acorde al algoritmo de ML	Se comparan los métodos de normalización acorde al algoritmo de ML
CP2	Se proporciona una tabla con valores numéricos	Se devuelven los valores normalizados acorde a los resultados del CP2	Se devuelven los valores normalizados acorde a los resultados del CP1
CP3	Se proporciona una columna con más de 15 categorías distintas	Se realiza la codificación One-Hot a estos valores	Se realiza la codificación One-Hot a estos valores

En la figura 3.7 se muestran los resultados del CP1, donde el mejor método para la normalización para esta base de datos con este algoritmo es Z-Score, con una precisión de 0.777 y Cohen's Kappa de 0.335. Para el CP2, se presenta en la figura 3.8 una vista previa de la tabla de salida del componente *Normalizer* con los datos normalizados.

Row ID	Accuracy	Cohen'...	rank
norm-z-score	0.777	0.335	1
norm-min-max	0.777	0.332	2
norm-decimal	0.776	0.329	3

Figura 3.7: Resultados del CP1 del componente Codificar y Normalizar

Row ID	satisfaction	Gender	Customer Type	Type of Travel	Class	Column0	Age	Flight Distance	Inflight wi fi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink
Row0	neutral or dissatisfied	-1.015	-2.115	-0.671	-0.957	-1.732	-0.951	-0.957	0.204	-0.695	0.174	0.018	-1.656
Row1	neutral or dissatisfied	0.985	0.473	-0.671	-0.957	-1.732	-0.951	-0.629	-0.55	1.272	1.603	1.584	-0.904
Row2	neutral or dissatisfied	0.985	-2.115	-0.671	0.654	-1.732	-1.018	-0.007	0.957	1.272	1.603	0.801	-0.904
Row3	neutral or dissatisfied	0.985	0.473	1.491	2.265	-1.732	-1.811	-0.884	-0.55	0.616	0.541	-0.765	-1.656
Row4	neutral or dissatisfied	-1.015	0.473	-0.671	0.654	-1.731	-2.01	-0.015	-0.55	0.616	0.541	0.801	-0.904
Row5	neutral or dissatisfied	0.985	0.473	1.491	0.654	-1.731	-1.481	-0.984	0.204	-1.351	0.174	0.018	1.352
Row6	neutral or dissatisfied	0.985	0.473	1.491	0.654	-1.731	0.24	-0.439	0.204	1.272	0.174	1.584	1.352
Row7	satisfied	0.985	0.473	-0.671	-0.957	-1.731	-0.422	0.373	-1.303	-1.351	-1.256	-1.547	-1.656
Row8	satisfied	0.985	0.473	-0.671	2.265	-1.731	0.636	-0.877	0.957	0.616	0.889	0.801	-0.904
Row9	satisfied	0.985	0.473	-0.671	-0.957	-1.731	-0.554	-0.372	0.957	0.616	0.889	0.801	1.352
Row10	neutral or dissatisfied	0.985	-2.115	-0.671	0.654	-1.731	-1.613	-0.147	-0.55	-0.695	0.541	0.018	1.352
Row11	neutral or dissatisfied	0.985	0.473	1.491	0.654	-1.731	1.827	0.003	0.957	1.272	0.889	-1.547	-0.904
Row12	neutral or dissatisfied	-1.015	-2.115	-0.671	-0.957	-1.731	-0.157	-0.007	0.204	-0.04	0.174	0.801	-1.656
Row13	neutral or dissatisfied	-1.015	0.473	1.491	2.265	-1.731	0.041	-0.641	0.957	-0.04	0.889	-0.765	-0.904
Row14	neutral or dissatisfied	-1.015	-2.115	-0.671	-0.957	-1.731	0.107	0.838	-1.303	-1.351	-1.256	-0.765	-1.656
Row15	satisfied	-1.015	0.473	-0.671	-0.957	-1.731	-0.025	0.546	0.957	0.616	0.889	0.801	-0.152

Figura 3.8: Vista previa de la salida del componente Normalizer

Por otra parte, para el CP3, se muestra en la figura 3.9 las categorías con más de 15 valores distintos a las que se les aplica el método One-Hot Encoding. En la figura 3.10, se

muestra una vista previa de la salida del componente *One-Hot Encoding*, siendo el resultado del caso de prueba en cuestión.

Row ID	Row0
city	123
experience	22

Figura 3.9: Categorías a codificar

Row ID	dme line #1	D PCA_dme line_12 ([Iter #1])	D PCA_dme line_13 ([Iter #1])	D PCA_dme line_14 ([Iter #1])	D PCA_dme line_15 ([Iter #1])	S  target	D  relevant_ experience	D  gender	D  enrolled_university	D  education_level	D  major_discipline	D  company_size	D  company_type	D  last_new_job	D  employee_id	D  city_development_index
Row0	0.002	-0.003	0.003	0.002	yes	0	0	0	0	0	0	0	0	0	0.089	0.92
Row1	0.005	-0.006	0.008	0.007	no	1	0	0.1	0	0	0	0	0	0.1	0.15	0.624
Row2	0.005	-0.008	0.008	0.007	no	1	0	0.1	0.1	0	0	0	0	0.1	0.087	0.624
Row3	0.005	-0.008	0.008	0.007	yes	0	0.1	0	0	0.1	0	0	0	0	0.114	0.827
Row4	-0.078	0.199	-0.862	0.323	no	0	0	0	0	0.2	0.1	0.1	0	0.07	0.776	
Row5	-0.015	0.026	-0.029	-0.03	yes	0	0	0	0	0	0.2	0	0.2	0.104	0.624	
Row6	-0.148	0.804	0.399	0.14	no	1	0	0.2	0.1	0	0.3	0.2	0	0.127	0.926	
Row7	0.002	-0.003	0.003	0.002	yes	0	0	0	0.2	0	0.4	0.1	0.3	0.102	0.926	
Row8	-0.053	0.112	-0.202	-0.923	no	1	0	0	0	0	0	0	0.3	0.086	0.92	
Row9	0.005	-0.008	0.008	0.007	yes	0	0	0	0	0	0.4	0.1	0.1	0.09	0.624	
Row10	0.002	-0.003	0.003	0.002	no	0	0	0	0	0	0.4	0.1	0	0.123	0.924	
Row11	0.002	-0.003	0.003	0.002	no	0	0	0	0	0	0	0	0.2	0.132	0.92	
Row12	0.008	-0.013	0.012	0.011	yes	0	0	0.1	0	0	0.5	0	0.1	0.135	0.624	
Row13	0.002	-0.003	0.003	0.002	no	0	0	0	0.3	0.2	0.4	0.1	0.4	0.084	0.887	
Row14	0.006	-0.009	0.009	0.008	no	1	0	0.1	0.1	0	0	0.1	0.1	0.096	0.55	
Row15	0.042	-0.056	0.047	0.036	no	0	0	0	0	0	0	0.4	0.4	0.096	0.91	

Figura 3.10: Vista previa de la salida del componente *Codificar y Normalizar*

## 3.2. Pruebas de caja negra al componente *AutoML Clasificación (Optimización de Hiperparámetros)*

### 3.2.1. Caso de prueba para el modelo RProp

### 3.2.2. Caso de prueba para el modelo PNN

### 3.2.3. Caso de prueba para el modelo SVM

### 3.2.4. Caso de prueba para el modelo Random Forest

## 3.3. Pruebas de integración al componente *AutoML Clasificación (pre-procesado)*

## 3.4. Conclusiones parciales

Al terminar este capítulo, se llega a las siguientes conclusiones:

- Las pruebas al subcomponente *Discretizer* integrado al *Componente AutoML Clasificación (pre-procesado)*, arrojaron mejores resultados con respecto al componente con una discretización previamente configurada.

- El número de intervalos, tras ejecutar las pruebas al *Componente AutoML Clasificación (pre-procesado)*, influyó en los resultados de la clasificación luego de emplear el mismo método de discretización (Equal-width).
- Las pruebas individuales al *Componente AutoML Clasificación (Optimización de Hiperparámetros)*, arrojaron mejores porcentajes de acierto con respecto al algoritmo no optimizado, demostrando su correcto funcionamiento.
- La prueba de integración del *Componente AutoML Clasificación (Optimización de Hiperparámetros)* al *Componente AutoML Clasificación (pre-procesado)* fue satisfactoria.
- Las pruebas realizadas al componente integrado demostraron una mejoría en la clasificación en comparación con el no integrado.

# Conclusiones generales

Con el cumplimiento de los objetivos planteados para la investigación, se puede llegar a las siguientes conclusiones:

- 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.
- El Aprendizaje Automático es una técnica que permite a las computadoras aprender a partir de datos, sin necesidad de ser programadas explícitamente para ello.
- Las principales tareas del AutoML son la selección de características, selección de modelos, pre-procesado de datos, ajuste de hiperparámetros y evaluación e interpretación de modelos.
- KNIME presenta pocas opciones disponibles para soportar AutoML, sobre todo para la etapa de pre-procesado y la optimización de hiperparámetros.
- Se implementó un subcomponente para la discretización automática de variables numéricas, basado en el rendimiento del algoritmo de clasificación ID3.
- Se implementó un componente que brinda soporte para tareas de AutoML, enfocándose en la etapa de optimización de hiperparámetros.
- Se demostró el correcto funcionamiento del componente propuesto y los subcomponentes que lo integran.

# Recomendaciones

- Hacer énfasis en la elección automatizada del número de bins para la discretización.
- Extender el conjunto de métodos de discretización a partir de la creación de plugins para KNIME.
- Desarrollar una nueva versión del componente AutoML Clasificación (Optimización de Hiperparámetros) que contenga:
  - Hiperparametrización de Árboles de Decisión.
  - Hiperparametrización de Redes Neuronales Pre-Alimentadas.
  - Hiperparametrización de SVM.
- Extender el conjunto de modelos disponibles a entrenar, sin delimitarse a la Clasificación.

# Referencias bibliográficas

- Agrawal, Rakesh, Imielinski, Tomasz, & Swami, Arun. 1993. Mining Association Rules between Sets of Items in Large Databases.
- Avanzi, Benjamin, Taylor, Greg, Wang, Melantha, & Wong, Bernard. 2023. Machine Learning with High-Cardinality Categorical Features in Actuarial Applications. *arXiv preprint arXiv:2301.12710*.
- Batista, Gustavo EAPA, & Monard, Maria Carolina. 2003. An analysis of four missing data treatment methods for supervised learning. *Applied artificial intelligence*, **17**(5-6), 519–533.
- 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.
- Bishop, Christopher M, & Nasrabadi, Nasser M. 2006. *Pattern recognition and machine learning*. Vol. 4. Springer.
- Bonaccorso, Giuseppe. 2017. *Machine learning algorithms*. Packt Publishing Ltd.
- Borràs, J, Delegido, Jesús, Pezzola, A, Pereira-Sandoval, M, Morassi, G, & Camps-Valls, G. 2017. Clasificación de usos del suelo a partir de imágenes Sentinel-2. *Revista de Teledetección*, 55–66.
- 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.
- Casas, Pablo. 2019. Data science live book. Retrieved from: [livebook.datascienceheroes.com](http://livebook.datascienceheroes.com).

- Catlett, Jason. 1991. On changing continuous attributes into ordered discrete attributes. *Pages 164–178 of: Machine Learning—EWSL-91: European Working Session on Learning Porto, Portugal, March 6–8, 1991 Proceedings* 5. Springer.
- Cerda, Patricio, & Varoquaux, Gaël. 2020. Encoding high-cardinality string categorical variables. *IEEE Transactions on Knowledge and Data Engineering*, **34**(3), 1164–1176.
- Cerda, Patricio, Varoquaux, Gaël, & Kégl, Balázs. 2018. Similarity encoding for learning with dirty categorical variables. *Machine Learning*, **107**(8-10), 1477–1494.
- Coria, Sergio, Orozco, Ivania, & Luna, Juan. 2013. Minería de datos para perfilamiento de las brechas digital y educativa de ciudades en censos de población y vivienda. *de Cuerpos Académicos*, 18.
- Davison, Anthony Christopher, & Hinkley, David Victor. 1997. *Bootstrap methods and their application*. Cambridge university press.
- Dhankhad, Sahil, Mohammed, Emad, & Far, Behrouz. 2018. Supervised machine learning algorithms for credit card fraudulent transaction detection: a comparative study. *Pages 122–125 of: 2018 IEEE international conference on information reuse and integration (IRI)*. IEEE.
- Dougherty, James, Kohavi, Ron, & Sahami, Mehran. 1995. Supervised and unsupervised discretization of continuous features. *Pages 194–202 of: Machine learning proceedings 1995*. Elsevier.
- Fayyad, U.M., & Irani, K.B. 1993. Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning. 1022–1029.
- 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*. "O'Reilly Media, Inc."
- 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.
- Holte, Robert C. 1993. Very simple classification rules perform well on most commonly used datasets. *Machine learning*, **11**, 63–90.
- Hooi, Eric Khoo Jiun, Zainal, Anazida, Kassim, Mohamad Nizam, & Ayub, Zaily. 2022. *Feature Encoding For High Cardinality Categorical Variables Using Entity Embeddings: A Case Study*. Tech. rept.
- Hutter, Frank, Kotthoff, Lars, & Vanschoren, Joaquin. 2019. *Automated machine learning: methods, systems, challenges*. Springer Nature.
- Jiawei Han, Micheline Kamber, Jian Pei. 2011. *Data Mining. Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. 3rd edn.
- Joachims, Thorsten. 2002. *Learning to classify text using support vector machines*. Vol. 668. Springer Science & Business Media.
- Kasemtaweechok, Chatchai, Pharkdepinyo, Patnaree, & Doungmanee, Patimakorn. 2021. Large-Scale Instance Selection using Center of Principal Components. *Pages 157–160 of: 2021 Joint International Conference on Digital Arts, Media and Technology with ECTI Northern Section Conference on Electrical, Electronics, Computer and Telecommunication Engineering*. IEEE.
- Kerber, R. 1992. ChiMerge: Discretization of Numeric Attributes. *Pages 123–128 of: National Conference on Artificial Intelligence American Association for Artificial Intelligence(AAAI'92)*.
- KNIME. 2023a (Nov.). *AutoML KNIME*. Visitado el 5 de noviembre del 2023.
- KNIME. 2023b (Mar.). *KNIME Official Site*. Visitado el 10 de marzo de 2023.
- Kurgan, Lukasz A, & Cios, Krzysztof J. 2004. CAIM discretization algorithm. *IEEE transactions on Knowledge and Data Engineering*, **16**(2), 145–153.

- Lakshmanan, Valliappa, Görner, Martin, & Gillard, Ryan. 2021. *Practical machine learning for computer vision*. .”Reilly Media, Inc.”.
- Lall, Upmanu, & Sharma, Ashish. 1996. A nearest neighbor bootstrap for resampling hydrologic time series. *Water resources research*, **32**(3), 679–693.
- LeDell, Erin, & Poirier, Sébastien. 2020. H2o automl: Scalable automatic machine learning. In: *Proceedings of the AutoML Workshop at ICML*, vol. 2020. ICML.
- Li, Dan, Deogun, Jitender, Spaulding, William, & Shuart, Bill. 2004. Towards missing data imputation: a study of fuzzy k-means clustering method. Pages 573–579 of: *Rough Sets and Current Trends in Computing: 4th International Conference, RSCTC 2004, Uppsala, Sweden, June 1-5, 2004. Proceedings* 4. Springer.
- Liu, Huan, & Setiono, Rudy. 1997. Feature selection via discretization. *IEEE Transactions on knowledge and Data Engineering*, **9**(4), 642–645.
- Liu, Huan, Hussain, Farhad, Tan, Chew Lim, & Dash, Manoranjan. 2002. Discretization: An enabling technique. *Data mining and knowledge discovery*, **6**, 393–423.
- Mahmood, Jawad, Luo, Ming, & Rehman, Mudassar. 2022. An accurate detection of tool wear type in drilling process by applying PCA and one-hot encoding to SSA-BLSTM model. *The International Journal of Advanced Manufacturing Technology*, 1–20.
- Méndez, José R, Riverola, Florentino Fdez, Díaz, Fernando, & Corchado, Juan M. 2007. Sistemas inteligentes para la detección y filtrado de correo spam: una revisión. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, **11**(34), 63–81.
- Moeyersoms, Julie, & Martens, David. 2015. Including high-cardinality attributes in predictive models: A case study in churn prediction in the energy sector. *Decision support systems*, **72**, 72–81.
- Montavon, Grégoire, Orr, Geneviève, & Müller, Klaus-Robert. 2012. *Neural networks: tricks of the trade*. Vol. 7700. Springer.
- Murphy, Kevin P. 2012. *Machine learning: a probabilistic perspective*. MIT press.
- Patil, Bankat M, Joshi, Ramesh C, & Toshniwal, Durga. 2010. Missing value imputation based on k-mean clustering with weighted distance. Pages 600–609 of: *Contemporary Computing: Third International Conference, IC3 2010, Noida, India, August 9-11, 2010. Proceedings, Part I* 3. Springer.
- Patro, SGOPAL, & Sahu, Kishore Kumar. 2015. Normalization: A preprocessing stage. *arXiv preprint arXiv:1503.06462*.

- Polatgil, Mesut. 2022. Investigation of the effect of normalization methods on ANFIS success: forestfire and diabetes datasets. *Int. J. Inf. Technol. Comput. Sci.*, **14**, 1–8.
- Praba, R, Darshan, G, Roshanraj, T, & Surya, B. 2021. Study On Machine Learning Algorithms. *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, 07, 67–72.
- Quinlan, J Ross. 1993. C 4.5: Programs for machine learning. *The Morgan Kaufmann Series in Machine Learning*.
- Sammut, Claude, & Webb, Geoffrey I. 2011. *Encyclopedia of machine learning*. Springer Science & Business Media.
- Scholkopf, Bernhard, & Smola, Alexander J. 2018. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.
- Sturges, Herbert A. 1926. The choice of a class interval. *Journal of the american statistical association*, **21**(153), 65–66.
- 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.
- Tsai, Chih-Fong, & Hu, Ya-Han. 2022. Empirical comparison of supervised learning techniques for missing value imputation. *Knowledge and Information Systems*, **64**(4), 1047–1075.
- 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.
- Ventevogel, PC. 2020. *Construction of a proactive alert management model by using artificial intelligence*. M.Phil. thesis, University of Twente.
- Wahba, Grace. 1975. Smoothing noisy data with spline functions. *Numerische mathematik*, **24**(5), 383–393.
- 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.

- Zhang, Shichao, Li, Xuelong, Zong, Ming, Zhu, Xiaofeng, & Wang, Ruili. 2017. Efficient kNN classification with different numbers of nearest neighbors. *IEEE transactions on neural networks and learning systems*, **29**(5), 1774–1785.
- Zhang, Xueying, & Song, Qinbao. 2019. Predicting the number of nearest neighbors for the k-NN classification algorithm. *Intelligent Data Analysis*, **18**(3), 449–464.
- 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**