



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:

Dra. Raisa Socorro Llanes

La Habana, Octubre 2023

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

Introducción	1
1. Aprendizaje Automático y AutoML	5
1.1. Proceso de descubrimiento de conocimiento en bases de datos	5
1.2. Minería de Datos	6
1.2.1. Clasificación	7
1.3. Aprendizaje Automático	11
1.4. AutoML	12
1.4.1. Pre-procesado	13
1.4.2. Optimización de hiperparámetros	19
1.5. Herramienta de minería de datos KNIME	23
1.6. Componente KNIME de AutoML para el pre-procesado en tareas de clasificación	24
1.7. Métodos para la evaluación	26
1.7.1. Bases de datos de prueba	26
1.7.2. Métricas	27
1.8. Conclusiones parciales	28
2. Propuesta de modificaciones al componente de AutoML para pre-procesado	30
2.1. Modificaciones en el pre-procesado	30
2.1.1. Discretización de variables numéricas	31
2.1.2. Normalización de variables numéricas	32
2.1.3. Pre-procesado de variables de tipo <i>string</i>	33
2.1.4. Manejo de valores faltantes	34
2.2. Componente AutoML Clasificación (Optimización de hiperparámetros) . . .	35
2.2.1. Uso y configuración del componente AutoML Clasificación (Optimización de Hiperparámetros)	36
2.2.2. Requisitos y restricciones del componente AutoML Clasificación (Optimización de Hiperparámetros)	37

2.2.3. Modelación del componente AutoML Clasificación (Optimización de Hiperparámetros)	38
2.2.4. Optimización de hiperparámetros para RProp	40
2.2.5. Optimización de hiperparámetros para PNN	42
2.2.6. Optimización de hiperparámetros para SVM	42
2.2.7. Optimización de hiperparámetros para Random Forest	42
2.3. Modelación de nueva versión del componente AutoML Clasificación	42
2.3.1. Procesado de ID3	43
2.3.2. Procesado para C4.5 y CART	43
2.3.3. Procesamiento para Redes Neuronales por Retropropagación	44
2.3.4. Procesamiento para Redes Neuronales Probabilísticas	44
2.3.5. Procesamiento para SVM	45
2.3.6. Procesamiento para Random Forest	45
2.4. Conclusiones parciales	46
3. Integración y validación de soluciones propuestas al componente de AutoML	47
3.1. Análisis de las bases de datos de prueba	47
3.2. Casos de prueba al subcomponente <i>Discretizer</i>	49
3.3. Casos de prueba del <i>Componente AutoML (Optimización de Hiperparámetros)</i>	53
3.3.1. Pruebas individuales al <i>Componente AutoML (Optimización de Hiperparámetros)</i>	53
3.3.2. Pruebas de integración al <i>Componente AutoML Clasificación (pre-procesado)</i>	55
3.4. Conclusiones parciales	56
Conclusiones	58
Recomendaciones	59
Referencias bibliográficas	60
A. Anexos	65
A.1. Flujo KNIME del componente AutoML Clasificacion (pre-procesado)	65
A.2. Flujo KNIME del procesamiento de RProp con HPO	65
A.3. Flujo KNIME de subcomponente para la discretización para el algoritmo ID3	66
A.4. Flujo KNIME de la ejecución del método CAIM para la discretización orientada al algoritmo ID3	67
A.5. Flujo KNIME para la comparación de métodos de discretización acorde al algoritmo ID3	67
A.6. Columnas de tipo String en BD3	69

Índice de tablas

Índice de figuras

1.1.	Red Neuronal Pre-Alimentada (Abiodun <i>et al.</i> , 2018)	9
1.2.	Red neuronal por retro-propagación (Abiodun <i>et al.</i> , 2018)	9
1.3.	Ejemplos de posibles hiperplanos de SVM (Sammut & Webb, 2011)	10
1.4.	Ejemplo de One-Hot Encoding (también conocido como Dummy Encoding) para el atributo ZIP Code	18
1.5.	Desglose de los subproblemas de AutoML (Zöller & Huber, 2021)	19
1.6.	Ejemplo de flujo de trabajo en la herramienta KNIME.	23
2.1.	Subcomponente Discretizer	31
2.2.	Diagrama de flujo general de Discretizer	31
2.3.	Nodos empleados para la discretización	32
2.4.	Diagrama de flujo para la normalización	33
2.5.	Diagrama de flujo para el pre-procesado de string	33
2.6.	Diagrama de flujo para el manejo de valores faltantes	34
2.7.	Diagrama de flujo para la imputación de valores faltantes	35
2.8.	Componente AutoML Clasificación (Optimización de Hiperparámetros)	36
2.9.	Ejemplo de configuración del componente AutoML (Optimización de Hiperparámetros)	37
2.10.	Vista de la salida.	37
2.11.	Diagrama de flujo general del componente AutoML Clasificación (Optimización de Hiperparámetros)	38
2.12.	Diagrama de actividades de selección de parámetros	39
2.13.	Nodo Column Selection Configuration	39
2.14.	Nodo Single Selection Configuration	39
2.15.	Nodo Integer Configuration	40
2.16.	Nodos para entrenar y probar Redes Neuronales por retro propagación	40
2.17.	Diagrama de actividades para el procesamiento de RProp con HPO	40
2.18.	Nodos Parameter Optimization Loop Start y Parameter Optimization Loop End	41
2.19.	Configuración del nodo Parameter Optimization Loop Start	41

2.20. Nodos X-Partitioner y X-Aggregator	41
2.21. Nodo Scorer	42
2.22. Nodo Table View	42
2.23. Diagrama de flujo del procesamiento del modelo ID3	43
2.24. Diagrama de flujo del procesamiento del modelo C4.5	43
2.25. Diagrama de flujo del procesado de CART	44
2.26. Diagrama de flujo del procesamiento de RProp	44
2.27. Diagrama de flujo para el procesado de PNN	45
2.28. Diagrama de flujo del procesamiento de SVM	45
2.29. Diagrama de flujo del procesamiento de Random Forest	46
3.1. Vista preliminar de BD1	48
3.2. Vista Preliminar de BD2	48
3.3. Vista preliminar de la BD3	48
3.4. Vista preliminar de la BD4	49
3.5. Integración del subcomponente <i>Discretizer</i>	50
3.6. Comparación de métodos de discretización para la BD1	50
3.7. Salida del suocomponente <i>Discretizer</i> para la BD1	51
3.8. Comparación de los resultados de ambos algoritmos para la BD1	51
3.9. Comparación de métodos de discretización para la BD2	51
3.10. Salida del subcomponente <i>Discretizer</i> para la BD2	52
3.11. Comparación de resultados entre ambos algoritmos para la BD2	52
3.12. Salida el subcomponente <i>Discretizer</i> para la BD3	53
3.13. Flujo para la comparación de algoritmos	53
3.14. Evaluación del rendimiento del modelo con BD1	54
3.15. Curva ROC comparación de modelos con BD1	54
3.16. Evaluación del rendimiento del modelo para la BD1	55
3.17. Curva ROC de comparación de modelos para la BD2	55
3.18. Integración de ambos componentes.	56
3.19. Evaluación de la integración con la BD4	56
A.2. Flujo KNIME del procesamiento de RProp con HPO	65
A.1. Flujo KNIME del Componente AutoML Clasificación (pre-procesado) (Carra-zana Ruiz, 2022)	66
A.3. Flujo KNIME de subcomponente para la discretización para el algoritmo ID3	66
A.4. Flujo KNIME de la ejecución del método CAIM para la discretización ori-en-tada al algoritmo ID3	67
A.5. Flujo KNIME para la comparación de métodos de discretización acorde al algoritmo ID3	67

A.6. Columnas de tipo String en BD3 69

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, 2023). 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, una de las principales desventajas que presenta es la falta de control y personalización. Aunque permite a los usuarios seleccionar automáticamente los algoritmos, no tienen control directo sobre estos procesos y no pueden ajustar los parámetros de manera manual. Esto puede limitar la capacidad de los usuarios para personalizar y optimizar los modelos para satisfacer sus necesidades específicas. 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, quedaron pendientes, 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 compo-

nente AutoML (*Componente AutoML Clasificación (pre-procesado)*) (Carrazana Ruiz, 2022), dado que este permite modificaciones, a diferencia del resto. 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 de pre-procesado de datos propuestos.
 - 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: Se requiere poseer todos los datos que sean de utilidad a partir de las necesidades de la organización. Determina las fuentes de información que pueden ser útiles y dónde conseguirlas. Como parte del desarrollo de esta fase es necesario diseñar o conocer el modo en que se van a organizar e integrar los datos; con el fin de eliminar redundancias e inconsistencias.
- Selección, limpieza y transformación: Se seleccionan los datos más relevantes y que aporten mejor información, garantizando que el dato tenga la mejor calidad posible logrando obtener las vistas minables, con los datos listos para la aplicación del

algoritmo.

- Algoritmos de Minería de datos: A través de la vista minable obtenida en la fase anterior se aplican los algoritmos de extracción del conocimiento.
- Evaluación e Interpretación: El objetivo de esta etapa es medir la calidad de los modelos obtenidos utilizando diferentes métricas de calidad. Las cuales dependen de las técnicas de minería de datos que se utilicen. La interpretación de los resultados se apoya en el uso de técnicas de visualización y de representación con el fin de entender mejor el conocimiento aportado.
- Difusión y uso: En esta etapa, se integra el conocimiento obtenido de la comprensión del negocio, con el conocimiento de los modelos de minería de datos usado en la toma de decisiones de los especialistas. La monitorización de los patrones debe realizarse, pues en ocasiones resulta necesaria la reevaluación del modelo, su reentrenamiento o incluso su reconstrucción total.

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

1.2. Minería de Datos

Acorde a (Hernández Orallo *et al.*, 2004), la minería de datos es definida como el proceso de extraer conocimiento útil y comprensible, previamente desconocido, desde grandes cantidades de datos almacenados en distintos formatos.

El conocimiento extraído se puede presentar en forma de relaciones, patrones o reglas inferidos de los datos, o en forma de descripción un poco más concisa. Estos constituyen un modelo de datos analizados. Estos modelos, o tareas, se categorizan en predictivas y descriptivas (Hernández Orallo *et al.*, 2004).

En las tareas predictivas, los ejemplos están etiquetados y se emplean para estimar valores futuros o desconocidos de variables de interés. En este entorno se encuentra el aprendizaje supervisado. En cambio, las tareas descriptivas son empleadas en el descubrimiento de propiedades de los datos examinados donde los ejemplos no se encuentran etiquetados. Aquí se pone de manifiesto el aprendizaje no supervisado. En (Hernández Orallo *et al.*, 2004) se describen las tareas de minería de datos 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.

1.2.1. Clasificación

En el uso común, la palabra clasificación significa colocar las cosas en categorías, agruparlas de alguna manera útil. El ser humano generalmente hace esto porque las cosas en un grupo, llamado *clase* en aprendizaje automático, comparten características comunes (Sammut & Webb, 2011).

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

Clasificación con Árboles de Decisión

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

- ID3 (Iterative Dichotomiser 3): Basa su funcionamiento en la entropía y la ganancia de información. El árbol se construye iterativamente de arriba hacia abajo, eligiendo en cada caso el atributo con mayor ganancia de información, hasta que la información ganada sea cero o se haya llegado a todas las hojas (Javed Mehedi Shamrat *et al.*, 2022), (Gupta *et al.*, 2017). Maneja datos nominales y no tolera valores faltantes.
- C4.5 (Classification 4.5): Desarrollado con el objetivo de mejorar los defectos de ID3. Añade la poda, desestimando las ramas sin aportes, que reduce los errores al clasificar como resultado de un gran número de ramas en el modelo (Sammut & Webb, 2011), (Javed Mehedi Shamrat *et al.*, 2022), (Gupta *et al.*, 2017). Adicionalmente maneja valores faltantes y numéricos.
- CART (Classification and Regression Trees): Genera un árbol binario siguiendo el mismo enfoque entrópico que ID3, pero empleando el coeficiente de Gini como criterio de selección (Javed Mehedi Shamrat *et al.*, 2022), (Gupta *et al.*, 2017). Es capaz de manejar datos faltantes, al igual que datos numéricos y nominales.
- Random Forest: Es un algoritmo de aprendizaje automático en el que se construye un conjunto de árboles de decisión para realizar predicciones. Cada árbol se entrena con un subconjunto aleatorio de datos y características, y luego se combina su salida para obtener una predicción más robusta y precisa. Reduce el sobreajuste y mejora la generalización al promediar múltiples modelos (Gupta *et al.*, 2017).

Clasificación con Redes Neuronales

Las redes neuronales o redes neuronales artificiales, son algoritmos de aprendizaje basados en una vaga analogía de cómo funciona el cerebro humano. El aprendizaje se logra

ajustando los pesos en las conexiones entre nodos, que son análogas a las sinapsis y las neuronas (Sammut & Webb, 2011).

Las primeras redes neuronales conocidas como pre-alimentadas, como la de la figura 1.1, se caracterizan por tener una arquitectura en la que cada capa de neuronas está conectada completamente con la capa siguiente, pero no con la capa anterior. Esto significa que la información fluye de forma unidireccional, sin retroalimentación (Abiodun *et al.*, 2018).

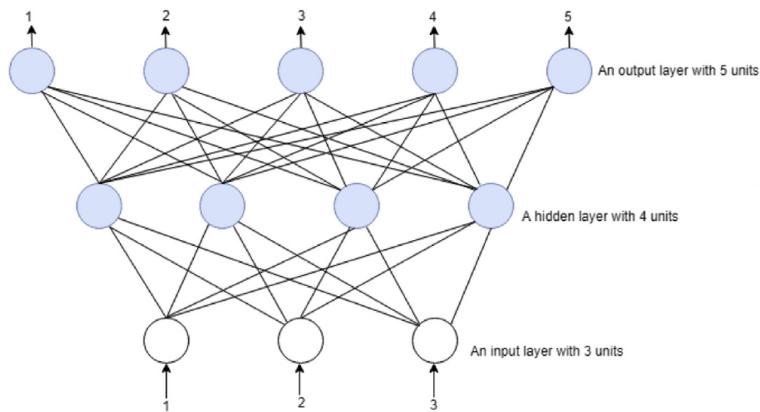


Figura 1.1: Red Neuronal Pre-Alimentada (Abiodun *et al.*, 2018)

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

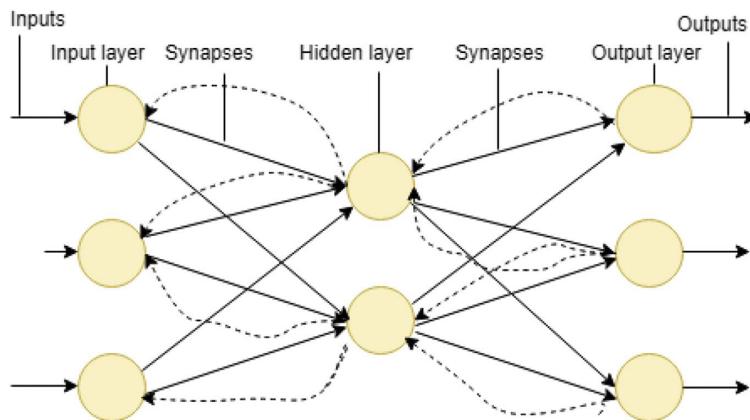


Figura 1.2: Red neuronal por retro-propagación (Abiodun *et al.*, 2018)

Gracias a su gran versatilidad se pueden aplicar en una amplia variedad de campos y disciplinas para resolver problemas complejos de aprendizaje automático, como es el procesamiento del lenguaje natural, reconocimiento de voz e imágenes (Abiodun *et al.*, 2018).

Debido a la complejidad de sus modelos puede ser difícil su entendimiento, por lo que preferiblemente se utilizan en el contexto del reconocimiento de patrones.

Clasificación con Support Vector Machine

Las Máquinas de Soporte Vectorial (Support Vector Machine o SVM, en inglés), son una clase de algoritmos lineales que se pueden emplear para la clasificación (Sammut & Webb, 2011), cuyo objetivo es encontrar el hiperplano que mejor separa dos clases de datos en un espacio de alta dimensionalidad (Guenther & Schonlau, 2016). En la figura 1.3 se representa como el hiperplano H_2 divide con mayor margen las clases que el hiperplano H_1 .

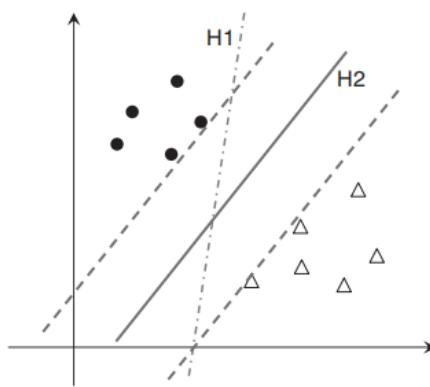


Figura 1.3: Ejemplos de posibles hiperplanos de SVM (Sammut & Webb, 2011)

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

- Uno contra uno (One-vs-One): Durante la fase de predicción, cada clasificador binario vota por la clase que cree que es correcta y la clase con el mayor número de votos se selecciona como la clase final para el punto de datos dado. Este enfoque presenta la ventaja de que cada clasificador binario solo necesita ser entrenado en un subconjunto de los datos, lo que puede ser útil cuando hay grandes conjuntos de datos. Es más resistente a desequilibrios en la distribución de clases que otros enfoques de SVM (Guenther & Schonlau, 2016).
- Uno contra todos (One-vs-All): Entrena un clasificador binario para cada clase posible, donde el conjunto de datos de una clase se considera positivo y los datos de las otras clases se consideran negativos. Durante la fase de predicción, cada clasificador binario vota por su clase correspondiente y la clase con la mayor puntuación se selecciona como la clase final para el punto de datos dado. Fácil de implementar y

puede funcionar bien en conjuntos de datos pequeños, pero puede ser menos preciso en conjuntos de datos grandes y complejos (Guenther & Schonlau, 2016).

Agregar que existen otros dos enfoques: Clasificación jerárquica (Hierarchical classification) y Clasificación por parejas (Pairwise classification). Cada uno de estos tiene sus propias ventajas y desventajas, y la elección de uno de ellos dependerá del tipo de datos y del problema que se esté tratando de resolver. Es importante mencionar que la clasificación es solo una de las muchas técnicas utilizadas en el Aprendizaje Automático.

1.3. 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 AutoML (aprendizaje automático automatizado).

1.4. *AutoML*

El campo del Aprendizaje Automático Automatizado (AutoML), tiene como objetivo tomar decisiones de una manera automatizada, objetiva y basada en datos: el usuario simplemente proporciona datos y el sistema AutoML determina automáticamente el enfoque que funciona mejor para esta aplicación en particular (Hutter *et al.*, 2019).

AutoML (Automated Machine Learning) es una técnica que tiene como objetivo automatizar todo o parte del proceso de Machine Learning, incluyendo la selección de algoritmos, la optimización de hiperparámetros, la selección de características y la evaluación del rendimiento del modelo (He *et al.*, 2021), (Tuggener *et al.*, 2019). La relación entre AutoML y Machine Learning es que AutoML es una técnica que se utiliza para automatizar el proceso de Machine Learning, 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 y la normalización y discretización de datos.
- Selección de modelo: Se basa en identificar el modelo de aprendizaje automático que mejor se ajusta al problema dado.
- Ajuste de hiperparámetros: Los modelos de aprendizaje automático tienen varios parámetros que afectan su rendimiento, y encontrar los valores óptimos de estos parámetros es una tarea importante para mejorar el rendimiento del modelo. El estado del arte ha demostrado que existen varias técnicas para el ajuste de hiperparámetros, incluyendo la búsqueda aleatoria (Zöller & Huber, 2021) y la optimización bayesiana (He *et al.*, 2021), Hutter *et al.* (2019).
- Evaluación del modelo: La evaluación del modelo es una tarea crítica para medir el rendimiento del modelo en datos de prueba para determinar su capacidad para generalizar. Existen varias técnicas para la evaluación del modelo, incluyendo la validación cruzada y la evaluación de curvas de aprendizaje.
- Interpretación del modelo: Consiste en analizar el modelo de aprendizaje automático para comprender cómo se toman las decisiones y qué variables son importantes para la predicción.

1.4.1. Pre-procesado

Una tarea importante en el proceso de aprendizaje automático automatizado es el preprocesamiento de datos, que es el conjunto de técnicas utilizadas para preparar los datos de entrada antes de alimentarlos a un modelo de aprendizaje automático. Esta etapa es la equivalente a la fase de selección, limpieza y transformación del proceso de KDD, descrita brevemente en la sección 1.1.

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 preprocesamiento de datos

a través de herramientas de AutoML, puede mejorar la eficiencia del proceso y ayudar al personal especializado, o sin mucha experiencia en el campo, a trabajar de manera más efectiva. Unas de las técnicas de preprocesado de datos son la discretización y la normalización.

Discretización

Este procedimiento transforma datos cuantitativos en datos cualitativos, es decir, atributos numéricos en atributos discretos o nominales con un número finito de intervalos, obteniendo una partición no superpuesta de un dominio continuo. Luego se establece una asociación entre cada intervalo con un valor numérico discreto. Una vez realizada la discretización, los datos pueden ser tratados como datos nominales durante cualquier proceso de minería de datos (García *et al.*, 2015), (Garcia *et al.*, 2012). Es necesario realizar la discretización de variables porque, entre varios factores, muchos de los algoritmos de Aprendizaje Automático requieren el uso de valores nominales solamente, como es el caso de ID3, Redes Bayesianas y Apriori. Otras ventajas derivadas de la discretización son la reducción y simplificación de datos, agilizando el aprendizaje y arrojando resultados más precisos, compactos y breves; y se reduce el posible ruido presente en los datos (Garcia *et al.*, 2012). No obstante, cualquier proceso de discretización conlleva generalmente una pérdida de información, siendo la minimización de esta pérdida el objetivo principal de un discretizador.

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

1. El número total de intervalos: intuitivamente, cuantos menos puntos de corte, mejor será el resultado de la discretización.
2. El número de inconsistencias causadas por la discretización: no debe ser mucho mayor que el número de inconsistencias de los datos originales antes de la discretización.
3. Precisión predictiva: cómo la discretización ayuda a mejorar la precisión.

En resumen, se necesitan al menos tres dimensiones: simplicidad, consistencia y precisión. Idealmente, el mejor resultado de discretización puede obtener la puntuación más alta en los tres departamentos. En realidad, puede no ser alcanzable o necesario.

El agrupamiento (**binning**), es el método más simple para discretizar un atributo de valor continuo mediante la creación de un número específico de grupos (bins). Los bins se

pueden crear con el mismo ancho (*equal-width*) y la misma frecuencia (*equal-frequency*). Cada bin está asociado con un valor discreto distinto. En *equal-width*, el rango continuo de una característica se divide uniformemente en intervalos que tienen un ancho igual, y cada intervalo representa un bin. En *equal-frequency*, se coloca un número igual de valores continuos en cada bin (Liu *et al.*, 2002), (Yang & Webb, 2009). Un método simple y similar a estos es el basado en cuantiles (*quantiles-based*), donde se producen bins correspondientes a una lista de probabilidades dada. El elemento más pequeño corresponde a una probabilidad de 0 y el más grande a una probabilidad de 1.

Para la selección del número de bins, en esta investigación se sigue el esquema de (Coria *et al.*, 2013), donde se utiliza la Regla de Sturges (Sturges, 1926). Propuesta por Herbert Sturges en 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.

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

Otro algoritmo de discretización es CAIM. El objetivo del algoritmo CAIM es encontrar el número mínimo de intervalos discretos mientras se minimiza la pérdida de interdependencia de atributo de clase. El algoritmo utiliza información de interdependencia de atributo de clase como criterio para la discretización óptima (Kurgan & Cios, 2004).

Normalización

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 ejecuta 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 realiza 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.

Las técnicas para la normalización de los datos tratadas en esta investigación son (García *et al.*, 2012):

- Normalización Min-Max: Transforma los valores de una variable para que estén en un rango específico, generalmente entre 0 y 1.
- Normalización Z-score (estandarización): Transforma los valores para que tengan una media de 0 y una desviación estándar de 1.

- 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.

Al igual que la discretización, la elección del mejor normalizador viene dada por las características de los datos y el algoritmo de aprendizaje automático que se vaya a emplear.

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 muestre: 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 esta investigación 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 clustering, 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.

K-Vecinos 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 (Tsai & Hu, 2022), (Batista & Monard, 2003). 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.

K-Medias (K-Means), por otro lado, 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 (Patil *et al.*, 2010), (Li *et al.*, 2004), 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.

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 1.4, 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. On the left, there is a table titled "Zip code" with rows for C1 (2000), C2 (1800), ..., C100 (2000), ..., and C3M (4000). An arrow labeled "Dummy encoding" points to the right, where another table shows the resulting binary matrix. This matrix has columns for 1000, 1100, ..., 1800, ..., 2000, ..., and 4000. The rows correspond to the categories C1, C2, ..., C100, ..., C3M. For each category, only one column has a value of 1, while all others are 0, demonstrating the "one-hot" nature of the encoding.

	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 1.4: 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), ya que captura el subespacio de máxima varianza. No obstante, esto puede generar un coste computacional alto. Sin embargo, existen otros métodos para la reducción de dimensionalidad menos costosos, como el filtrado por varianza y por correlación.

La varianza es una medida estadística que proporciona información sobre cuán dispersos o agrupados están los valores en un conjunto de datos. La filtración por varianza se aplica para eliminar características que tienen una varianza muy baja o cercana a cero, ya que cuanto menor sea la varianza, más cerca estarán los datos entre sí. Las características con poca variabilidad no aportan información útil al modelo y, por lo tanto, pueden ser eliminadas. Por otra parte, la correlación es una medida estadística que describe la relación o asociación entre dos o más variables. Cuando dos variables están correlacionadas, significa que un cambio en una variable está relacionado de alguna manera con un cambio en la otra variable. Esto, en conjuntos de datos de alta dimensionalidad, puede generar redundancias. Para evitarlo, se pueden eliminar los atributos que representan una alta correlación (Chandrashekhar & Sahin, 2014).

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).

1.4.2. 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 1.5.

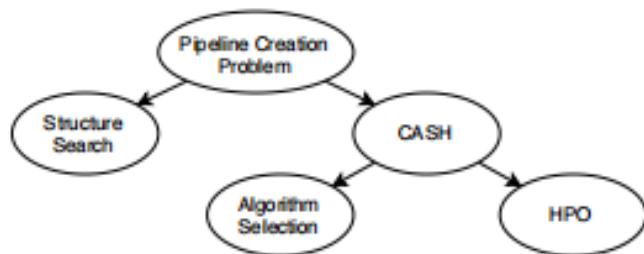


Figura 1.5: Desglose de los subproblemas de AutoML (Zöller & Huber, 2021)

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

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

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 elegir las variables correctas, se simplifica y mejora la interpretación de los modelos, reduce el riesgo de sobreajuste y acelera el tiempo de entrenamiento. Es importante destacar que la elección 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. Las variables a optimizar por algoritmo en el presente trabajo de investigación son (Scholkopf & Smola, 2018):

- Random Forest:
 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 (Scholkopf & Smola, 2018).

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 (Scholkopf & Smola, 2018).
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 (Scholkopf & Smola, 2018).

- Redes Neuronales por Retropropagación:

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** (Hastie *et al.*, 2009).
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 (Hastie *et al.*, 2009).
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 (Hastie *et al.*, 2009).

- Redes Neuronales Probabilísticas:

1. *Theta Minus (θ_-)*: 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 (Hastie *et al.*, 2009).
2. *Theta Plus (θ_+)*: 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 (Hastie *et al.*, 2009).

- SVM:

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 (Joachims, 2002), (Montavon *et al.*, 2012). En esta investigación se utilizan tres tipos de kernels (Joachims, 2002):
 - 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 y es 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 (Bishop & Nasrabadi, 2006), (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 (Montavon *et al.*, 2012), (Joachims, 2002).

Una vez identificadas las variables clave del modelo, es crucial evaluar su rendimiento de manera sólida y confiable. En este contexto, las técnicas de validación son una herramienta esencial para evaluar y comparar diferentes combinaciones de hiperparámetros. Para lograr esto, una técnica esencial es la validación cruzada. La 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 nos 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).

A pesar del incesante estudio vinculado al HPO, este sigue presentando en la actualidad diversos desafíos (Hutter *et al.*, 2019):

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

- Generalización del modelo: la optimización de hiperparámetros puede resultar en un modelo altamente ajustado, que no generaliza bien a nuevos datos. Se torna complejo cuando las bases de datos poseen múltiples tipos de datos.
- Complejidad del espacio de hiperparámetros: el espacio de hiperparámetros puede ser muy complejo y estar altamente interconectado, lo que dificulta la exploración y la selección de los hiperparámetros adecuados.

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

1.5. Herramienta de minería de datos KNIME

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, 2023).

Los flujos se componen de flechas y nodos que se pueden combinar entre sí. Los nodos contienen funcionalidades tales como: algoritmos de minería de datos, formas de conexión a los datos almacenados, preprocesamiento de datos, reportes, entre otros. Las flechas indican el orden de ejecución y el flujo de la información. En la figura 1.6, se muestra un ejemplo de un flujo en KNIME para cargar y filtrar datos de una tabla, y posteriormente guardar los resultados en un fichero .csv.

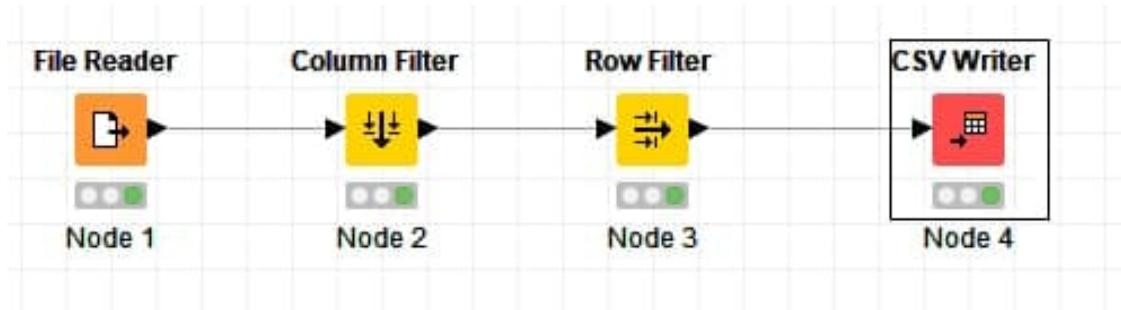


Figura 1.6: Ejemplo de flujo de trabajo en la herramienta KNIME.

Los metanodos son un tipo de nodo que contiene un subflujo, estos pueden contener varios nodos y metanodos (Berthold *et al.*, 2009). Se comporta como un flujo regular. Ge-

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

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

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

AutoML en KNIME se refiere a la capacidad de la plataforma para automatizar el proceso de modelado de aprendizaje automático. Esto significa que los usuarios pueden cargar datos y permitir que la plataforma seleccione y optimice automáticamente el modelo que mejor se ajuste a los datos. Con tal objetivo, en (Carrazana Ruiz, 2022) se desarrolla un componente KNIME de AutoML para el pre-procesado en tareas de clasificación. Este, a partir de un conjunto de datos y una columna objetivo, ejecuta diferentes flujos de pre-procesado,

en aras de cumplir con los requisitos de los diferentes algoritmos de clasificación, siendo capaz de entrenarlos y probarlos, para posteriormente puntuar y graficar los resultados (Carrazana Ruiz, 2022). En el anexo A.1 se muestra el flujo KNIME del Componente AutoML Clasificación (pre-procesado).

Este componente está enfocado en el pre-procesado de datos, donde se desarrollaron sub-componentes enfocados en la realización de las tareas de procesamiento de datos numéricos, *string*, valores faltantes y el ajuste de tipos de columna. Como se observa en el anexo A.1, se aplican los algoritmos de clasificación 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.

No obstante, este componente, 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 esta 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 numero 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 cuentan 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 1.4.1, 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.

1.7. Métodos para la evaluación

En el campo de la clasificación, existen varios métodos de evaluación que se utilizan para comparar diferentes algoritmos o enfoques en una tarea de clasificación. Estos métodos ayudan a medir el rendimiento y la eficacia de los modelos de clasificación y 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.

1.7.1. 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 Datasets¹ y UCI Machine Learning Repository² son dos de los repositorios en línea más populares para conjuntos de datos de prueba y de aprendizaje automático. Kaggle Datasets es un sitio web de aprendizaje automático que ofrece una amplia variedad de 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

¹<https://www.kaggle.com/datasets>

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

amplia gama de conjuntos de datos, desde reconocimiento de voz hasta predicción de precios de viviendas. Los usuarios pueden descargar los conjuntos de datos de forma gratuita y utilizarlos para probar y desarrollar sus modelos de aprendizaje automático.

Ambos repositorios ofrecen una amplia variedad de conjuntos de datos, lo que los hace ideales para desarrolladores, estudiantes y profesionales de la ciencia de datos que buscan mejorar sus habilidades en el modelado de bases de datos y en la creación de modelos de aprendizaje automático precisos y efectivos. Por tal motivo, se emplearán ambos repositorios para la obtención de bases de datos para las pruebas que se realizarán en este proyecto.

hablar de las bd que se van a emplear

1.7.2. 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): La exactitud es una métrica básica que calcula la proporción de predicciones correctas sobre el total de predicciones realizadas por el modelo. Sin embargo, la exactitud puede ser engañosa cuando hay desequilibrio de clases en el conjunto de datos.
- Precisión (Precision): La precisión es la proporción de predicciones positivas correctas con respecto al total de predicciones positivas realizadas por el modelo. Mide la capacidad del modelo para predecir correctamente los ejemplos positivos.
- 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 and 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.

1.8. Conclusiones parciales

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

- El Aprendizaje Automático es una técnica que permite a las computadoras aprender a partir de datos, sin necesidad de ser programadas explícitamente.
- El proceso de descubrimiento de conocimiento en bases de datos (KDD) es un proceso iterativo que consiste en varias etapas, incluyendo la selección de datos, la limpieza de datos, la transformación de datos y la minería de datos.
- La Minería de Datos es el proceso de descubrir patrones y relaciones interesantes en grandes conjuntos de datos, utilizando técnicas de aprendizaje automático, estadísticas y visualización de datos. Algunas de las técnicas utilizadas en la Minería de Datos incluyen la clasificación, la agrupación, la regresión y la asociación.
- La Clasificación es una técnica de aprendizaje automático que se utiliza para predecir la etiqueta o clase de un objeto a partir de un conjunto de características.
- El AutoML se puede utilizar para mejorar la eficiencia y la precisión del proceso de modelado, reducir la necesidad de conocimientos especializados y permitir a los usuarios enfocarse en la interpretación de los resultados. Las etapas del AutoML incluyen la selección automática de algoritmos, el preprocesamiento de datos, la optimización de hiperparámetros y la evaluación automática del modelo.
- El 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-procesamiento 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 de 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 preprocesamiento 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 epígrafe 1.6, se llevaron a cabo una serie de modificaciones sustanciales en el proceso de pre-procesamiento de datos. Estas adaptaciones se centraron en la automatización de la discretización y normalización, así como en el tratamiento de valores únicos, valores de alta cardinalidad y valores faltantes. 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.

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* (Figura 2.1).

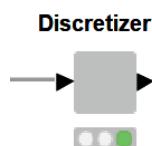


Figura 2.1: Subcomponente *Discretizer*

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 2.2 expone el flujo general del componente *Discretizer*. El flujo KNIME correspondiente está presente en el anexo A.3.

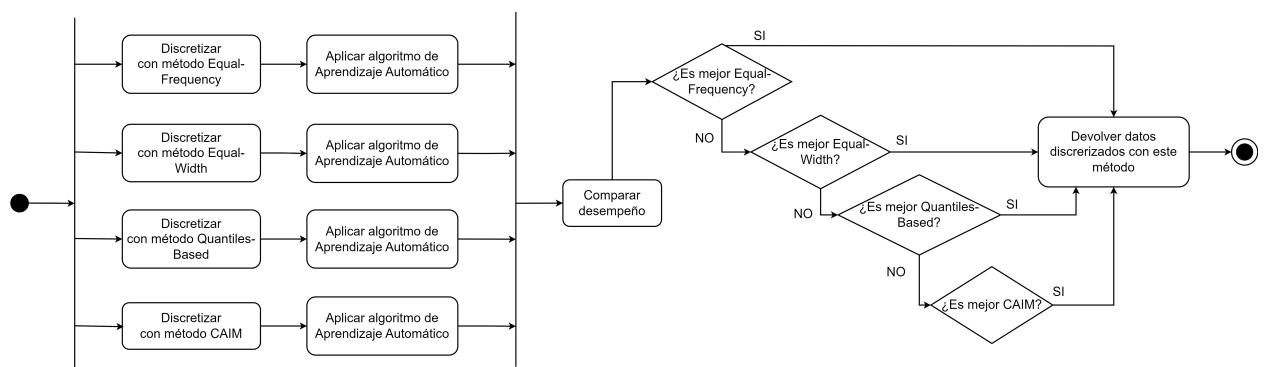


Figura 2.2: Diagrama de flujo general de *Discretizer*

Para la discretización se emplean los nodos presentes en la figura 2.3. 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 1.4.1. El nodo *CAIM Binner* no requiere una configuración en específico, y para el método Quantile-Based se escogieron los cuantiles por defecto, es decir 0.0, 0.25, 0.5, 0.75 y 1.0.

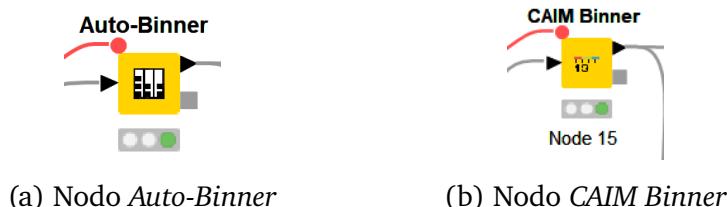


Figura 2.3: Nodos empleados para la discretización

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 precisión y Cohen's Kappa, y escoger el algoritmo con mejor desempeño.

2.1.2. Normalización de variables numéricas

La normalización son proporciones sin unidades de medida (adimensionales o invariantes de escala) que nos permiten poder comparar elementos de distintas variables y distintas 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", presente en la FIGURA X. En este nodo se encuentran tres métodos para normalizar, a elección del usuario: Decimal Scaling, Z-Score y Min-Max. En aras de automatizar este proceso, se propone el subcomponente "Normalizer", de igual nombre al nodo nativo de KNIME, en donde se encuentra la comparación de los métodos anteriormente mencionados en función de un modelo predeterminado. En la figura 2.4 se presenta el diagrama de flujo de este subcomponente.

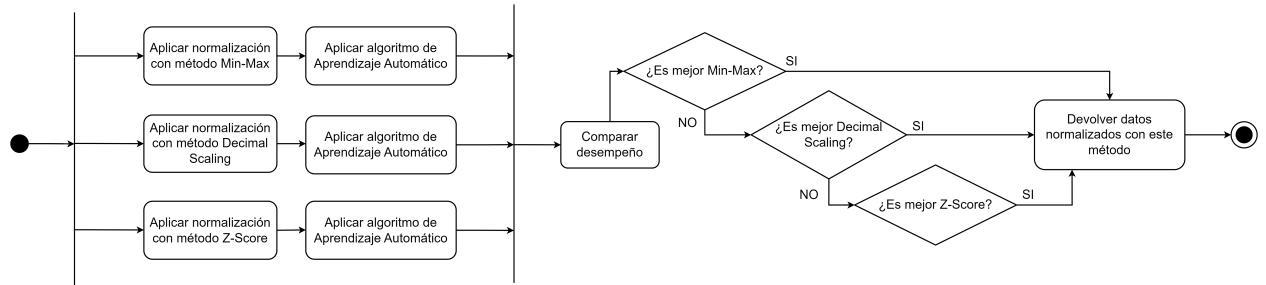


Figura 2.4: Diagrama de flujo para la normalización

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 la precisión y el Cohen's Kappa. Luego de escoger el mejor, se devuelve la tabla con los datos normalizados.

2.1.3. 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. 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. El Componente AutoML Clasificación (pre-procesado) poseía un tratamiento erróneo de estos valores, al eliminar las columnas nominales que superaban un umbral determinado de categorías distintas. En aras de depurar estas inconsistencias, el diagrama de la figura 2.5 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 las actividades en verde reflejan las nuevas implementaciones.

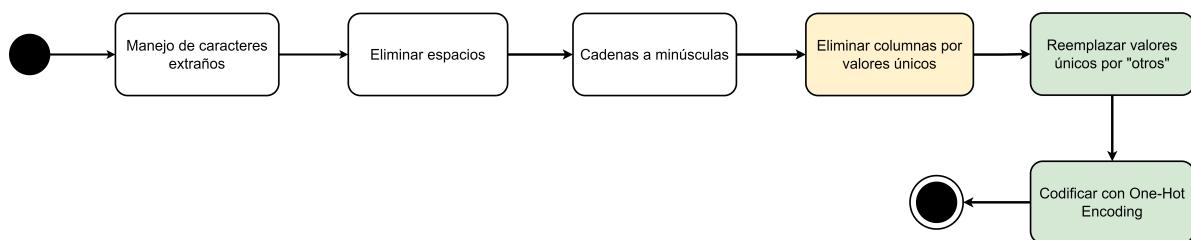


Figura 2.5: Diagrama de flujo para el pre-procesado de string

A continuación se exponen los cambios realizados al componente String preprocs:

- 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 mas del 80 % de los valores son únicos.
- Reemplazar con otros: En este nuevo componente, los valores únicos en una columna que representan una minoría, son reemplazados por la categoría 'otros'. Para ello, iterando por cada columna dentro de un ciclo, se calculan la frecuencia absoluta y relativa de cada atributo, y aquellos que representen menos del 1% son los elegidos para la sustitución por la nueva categoría.
- Codificar con One-Hot Encoding: Para el tratamiento de valores con alta cardinalidad, se emplea la codificación One-Hot, utilizando el nodo One To Many. Para esto se escogen los atributos que por defecto tienen, como mínimo, 15 categorías distintas. Dado que esta técnica produce gran dimensionalidad, se utilizan para su reducción los métodos
 1. Filtrar por varianza, utilizando el nodo Low Variance Filter, donde las columnas creadas con una varianza menor a 0.01 son eliminadas; y
 2. Filtrar por correlación, utilizando el nodo Correlation Filter, donde las columnas con una correlación mayor a 0.9 son eliminadas.

El flujo KNIME de los nuevos subcomponentes se encuentran en los ANEXOS X y Y.

2.1.4. 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 imputacion 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 cambiada, por los motivos expuestos en la sección 1.4.1. En la figura 2.7 se presenta el diagrama de flujo de la nueva implementación para la sustitución de estos valores.

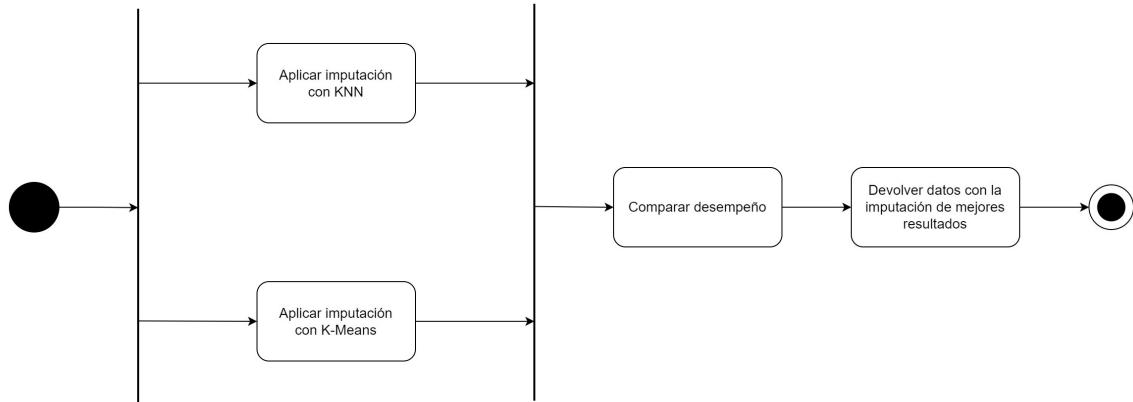


Figura 2.7: Diagrama de flujo para la imputación de valores faltantes

Primeramente se aplican en paralelo las técnicas de imputación KNNI y KMI, 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 a las métricas Precisión y Cohen's Kappa para devolver los datos con la mejor sustitución. En el ANEXO X se muestra el flujo KNIME de estas estrategias.

2.2. Componente AutoML Clasificación (Optimización de hiperparámetros)

Antes de explorar en detalle las adaptaciones realizadas en cada modelo, es fundamental presentar el Componente AutoML Clasificación (Optimización de hiperparámetros), con el objetivo de facilitar el entendimiento en los epígrafes posteriores.

toooooo lo que lleva con sus epígrafes y esa talla, esto es lo que estaba en las practicas

El componente propuesto, presente en la figura 2.8, para la optimización de hiperparámetros, contiene la siguiente configuración:



Figura 2.8: Componente *AutoML Clasificación (Optimización de Hiperparámetros)*

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

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

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

1. Proporcionar el conjunto de datos al puerto de entrada (el componente marcará error en este punto, pues la configuración es obligatoria).
2. Dar click derecho sobre el componente, seleccionar “Configure...”.
3. Seleccionar la configuración deseada (como se observa en el ejemplo de la figura 2.9).
4. Ejecutar el componente (inicialmente el puerto de salida se encuentra vacío).
5. Dar click derecho sobre el componente y seleccionar “Interactive View: AutoML” (Fig3).

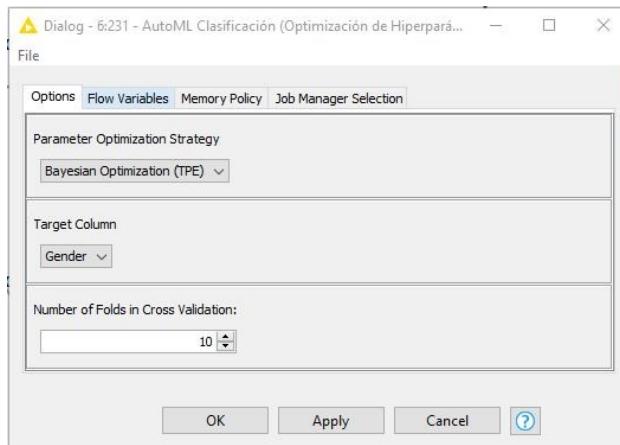


Figura 2.9: Ejemplo de configuración del componente *AutoML (Optimización de Hiperparámetros)*

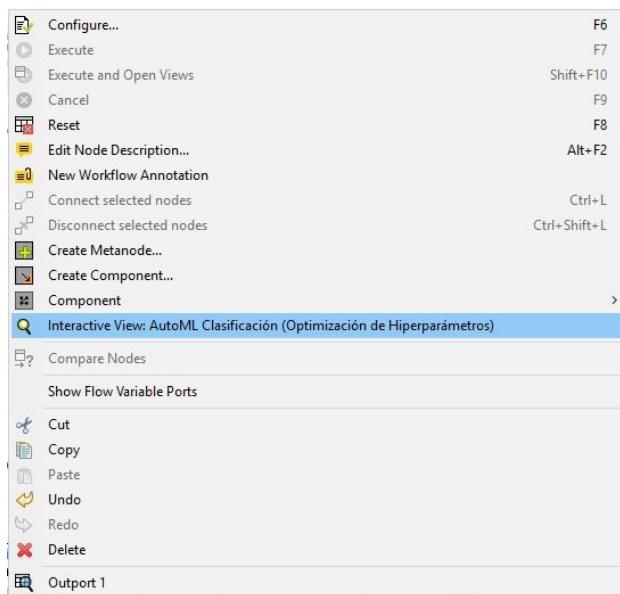


Figura 2.10: Vista de la salida.

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

El componente propuesto debe cumplir los siguientes requisitos funcionales:

- RF1: El componente debe permitir seleccionar columna objetivo
- RF2: El componente debe permitir seleccionar estrategia de optimización de hiperparámetros.

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

El componente propuesto presenta las siguientes restricciones para su funcionamiento:

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

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

El diagrama de flujo de la figura 2.11 expone el flujo general del componente *AutoML Clasificación (Optimización de Hiperparámetros)*.



Figura 2.11: Diagrama de flujo general del componente *AutoML Clasificación (Optimización de Hiperparámetros)*

La configuración de la selección de parámetros es el primer paso, en el cual se crearán las variables que dictarán el comportamiento del flujo. Posteriormente, se ejecuta la optimización de hiperparámetros para RProp, donde recoge los resultados en una tabla y luego los grafica. El flujo KNIME correspondiente se evidencia en el Anexo A.2.

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 2.12 expone el flujo para la selección de parámetros.

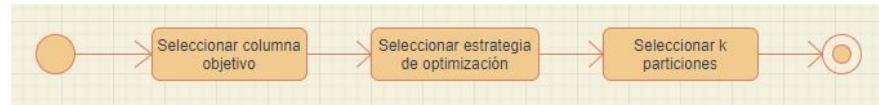


Figura 2.12: Diagrama de actividades de selección de parámetros

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

- Seleccionar columna objetivo: se emplea el nodo *Column Selection Configuration* (Figura 2.13), el cual recibe una tabla y devuelve el nombre de la columna seleccionada como variable de flujo. En este caso, presenta la configuración adicional para solo mostrar las columnas de tipo *string*.

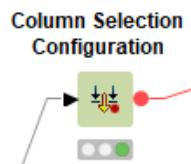


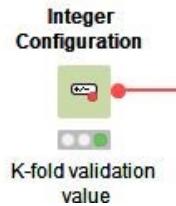
Figura 2.13: Nodo *Column Selection Configuration*

- Seleccionar estrategia de optimización de hiperparámetros: la selección de la estrategia se lleva a cabo empleando el nodo *Single Selection Configuration* (Figura 2.14). Devuelve la variable *strategy* con el valor seleccionado previamente.



Figura 2.14: Nodo *Single Selection Configuration*

- Seleccionar número de subconjuntos en la validación cruzada: la selección de la cantidad de subconjuntos de partición de entrenamiento, se lleva a cabo con el nodo *Integer Configuration* (Figura 2.15). Este devuelve la variable de flujo resultante de la selección, en este caso presenta la configuración para limitar el rango entre 5 y 10.

Figura 2.15: Nodo *Integer Configuration*

2.2.4. Optimización de hiperparámetros para RProp

Las Redes Neuronales por Retro-propagación necesitan que todos los valores sean de tipo numéricos y estos se encuentren normalizados. Para el entrenamiento y prueba de las Redes Neuronales por Retro-propagación, se emplean los nodos *RProp MLP Learner* y *MultiLayerPerceptron Predictor* (Figura 2.16) respectivamente.

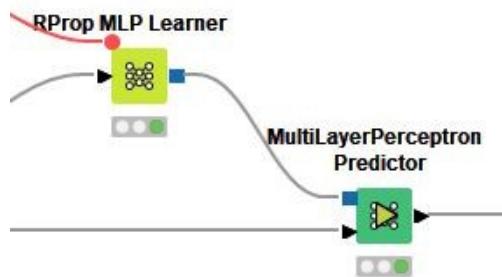


Figura 2.16: Nodos para entrenar y probar Redes Neuronales por retro-propagación

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

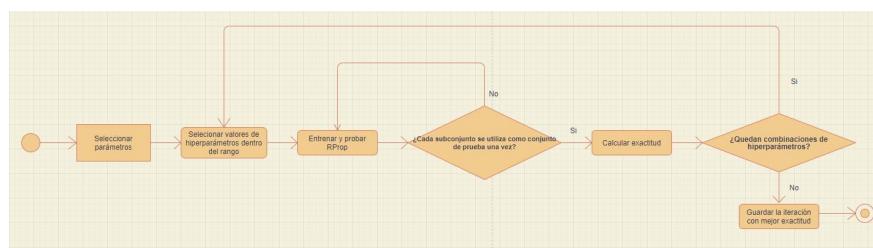


Figura 2.17: Diagrama de actividades para el procesamiento de RProp con HPO

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

- Ciclo para recorrer el rango de hiperparámetros: se emplean los nodos *Parameter Optimization Loop Start* y *Parameter Optimization Loop End* (Figura 2.18). Ambos

permiten guardar todas las iteraciones realizadas por el algoritmo con las diferentes combinaciones de hiperparámetros. Para su configuración, se eligieron como hiperparámetros el número de capas, la cantidad de neuronas y el número máximo de iteraciones (Figura 2.19).

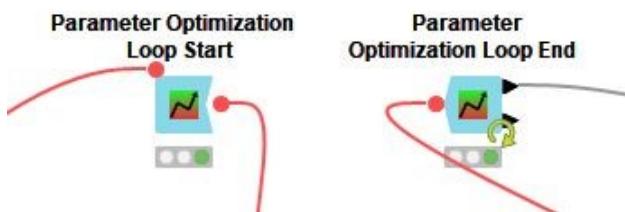


Figura 2.18: Nodos *Parameter Optimization Loop Start* y *Parameter Optimization Loop End*

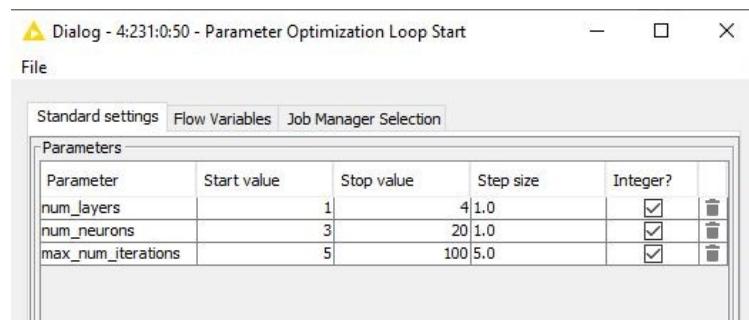


Figura 2.19: Configuración del nodo *Parameter Optimization Loop Start*

- Ciclo para dividir el conjunto de datos: se emplean los nodos *X-Partitioner* y *X-Aggregator* (Figura 2.20), para dividir el conjunto de datos en k particiones y realizar una validación cruzada, donde cada partición se utiliza como conjunto de prueba una vez y las otras $k-1$ particiones se utilizan como conjunto de entrenamiento.

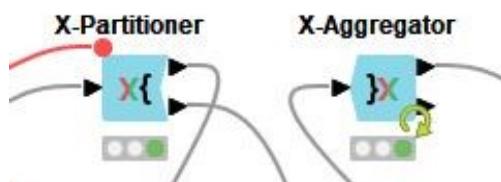
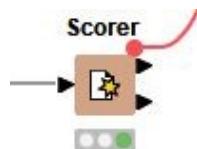
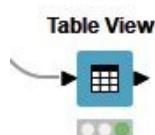


Figura 2.20: Nodos *X-Partitioner* y *X-Aggregator*

- Calcular la exactitud: se emplea el nodo *Scorer* (Figura 2.21), el cual recibe la predicción y la columna objetivo en una tabla para la evaluación.

Figura 2.21: Nodo *Scorer*

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

Figura 2.22: Nodo *Table View*

2.2.5. Optimización de hiperparámetros para PNN

blablabla blablabla

2.2.6. Optimización de hiperparámetros para SVM

blablabla

2.2.7. Optimización de hiperparámetros para Random Forest

blablabla

2.3. Modelación de nueva versión del componente AutoML Clasificación

- componente de ernesto
- diagrama nuevo
- cambios hechos en la personalización: random forest, quitado lo de valores únicos por columna, agregar la estrategia de optimización

2.3.1. Procesado de ID3

blablabla

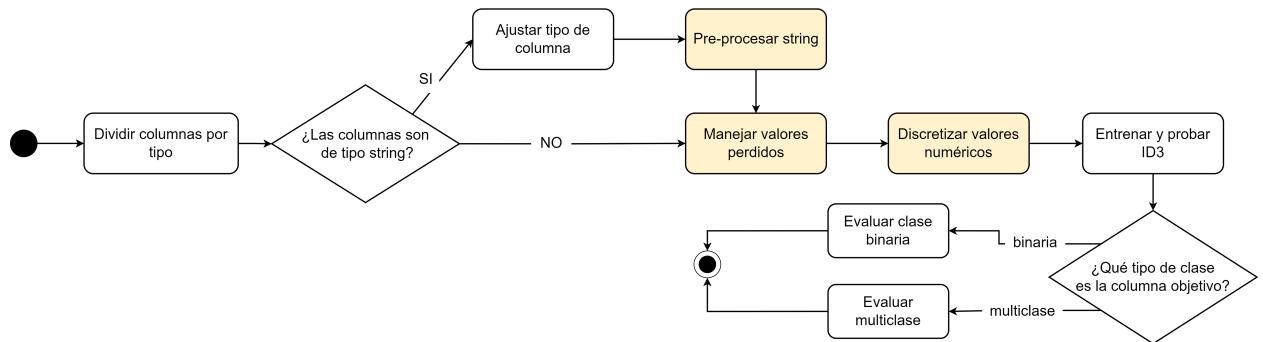


Figura 2.23: Diagrama de flujo del procesamiento del modelo ID3

blablabla

2.3.2. Procesado para C4.5 y CART

blablabla

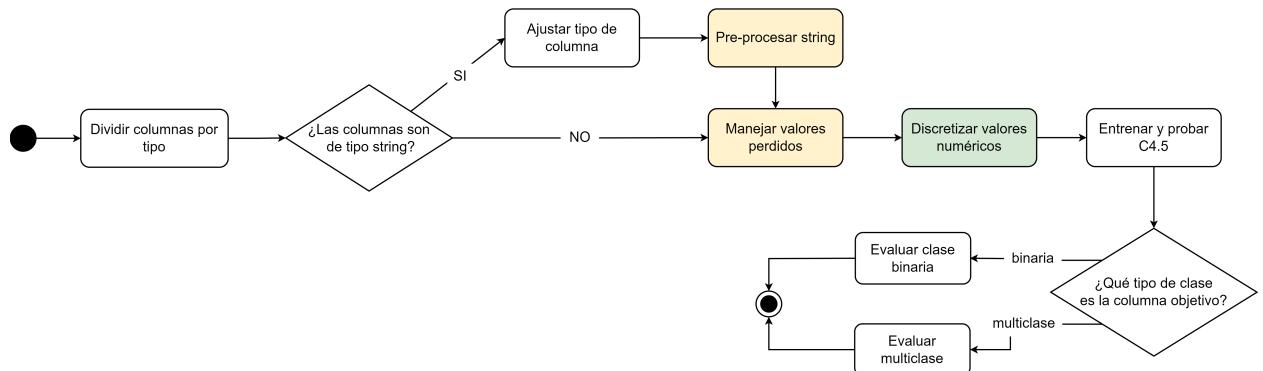


Figura 2.24: Diagrama de flujo del procesamiento del modelo C4.5

blablabla

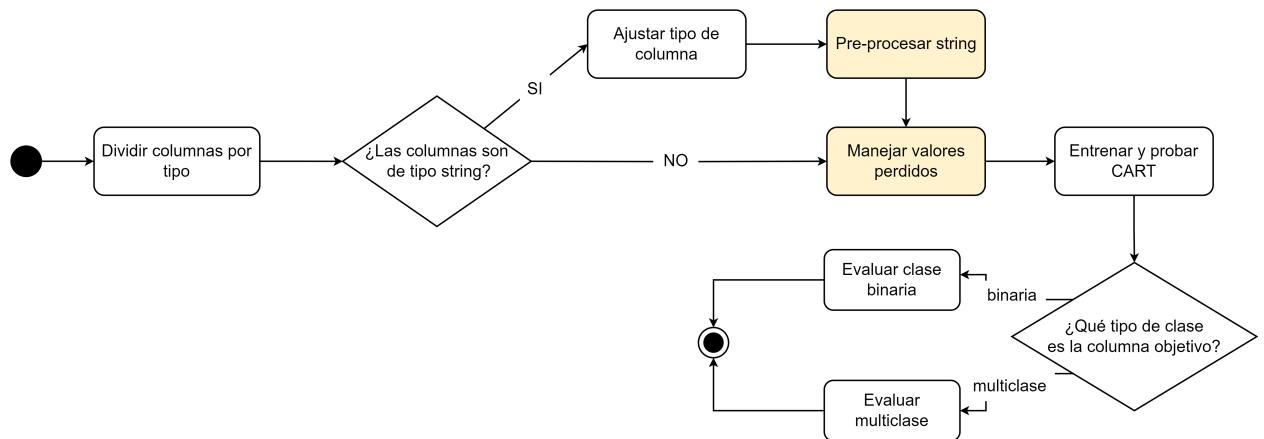


Figura 2.25: Diagrama de flujo del procesamiento de CART

blablabla

2.3.3. Procesamiento para Redes Neuronales por Retropropagación

blablabla

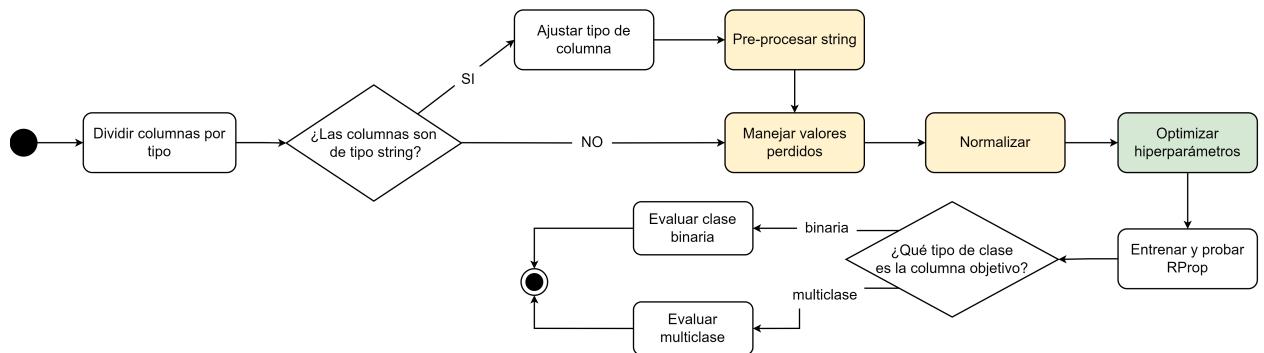


Figura 2.26: Diagrama de flujo del procesamiento de RProp

blablabla

2.3.4. Procesamiento para Redes Neuronales Probabilísticas

blablabla

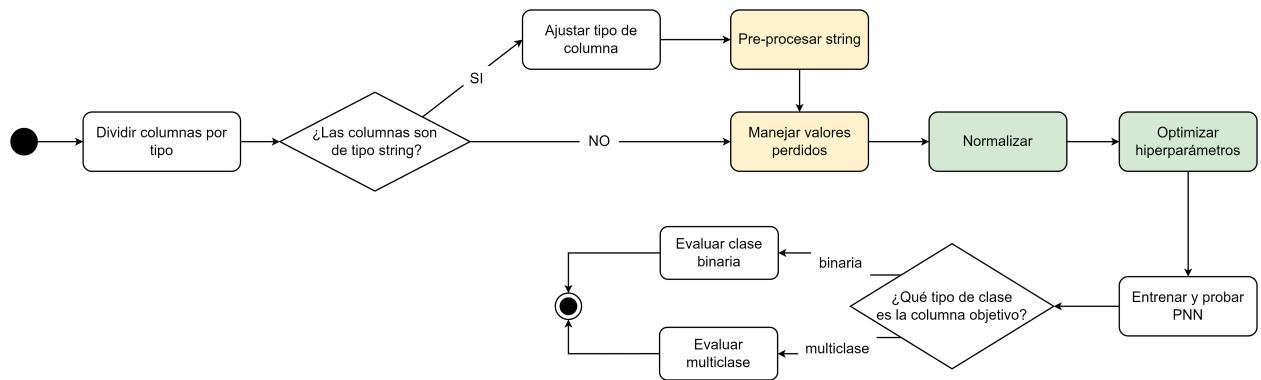


Figura 2.27: Diagrama de flujo para el procesamiento de PNN

blablabla

2.3.5. Procesamiento para SVM

blablabla

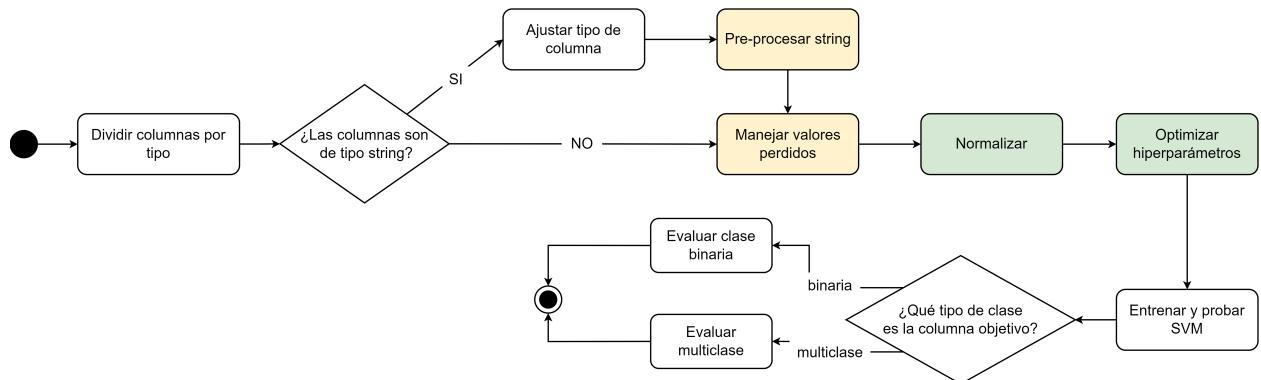


Figura 2.28: Diagrama de flujo del procesamiento de SVM

blablabla

2.3.6. Procesamiento para Random Forest

blablabla

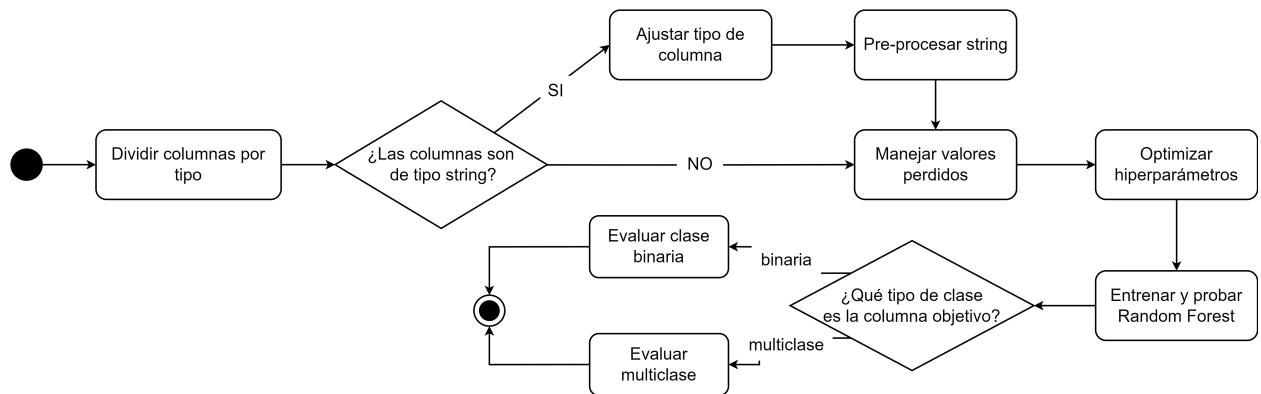


Figura 2.29: Diagrama de flujo del procesamiento de Random Forest

blablabla

2.4. Conclusiones parciales

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

Integración y validación de soluciones propuestas al componente de AutoML

En este capítulo, se presentan las pruebas necesarias para validar el correcto funcionamiento del subcomponente *Discretizer* y del *Componente AutoML Clasificación (Optimización de Hiperparámetros)*, así como las pruebas de sus integraciones al *Componente AutoML Clasificación (pre-procesado)*.

3.1. Análisis de las bases de datos de prueba

Las bases de datos a emplear fueron descargadas de los repositorios Kaggle Repository y UCI Machine Learning Repository. A continuación, se presenta una descripción de las mismas:

- Base de datos 1 (BD1, Figura 3.1): presenta 614 instancias y 11 atributos, 6 de ellos de tipo *string* y los restantes de tipo numérico. Esta base de datos se puede clasificar en función de dos columnas objetivo, donde una es multiclase y la otra binaria.

Row ID	S Gender	S Married	S Depend...	S Education	S Self_E...	I Aplica...	D Coopoli...	I Loan_A...	I Term	I Credit ...	S Area
Row0	Male	No	0	Graduate	No	584900	0	15000000	360	1	Urban
Row1	Male	Yes	1	Graduate	No	458300	150,800	12800000	360	1	Rural
Row2	Male	Yes	0	Graduate	Yes	300000	0	66000000	360	1	Urban
Row3	Male	Yes	0	Not Graduate	No	258300	235,800	12000000	360	1	Urban
Row4	Male	No	0	Graduate	No	600000	0	14100000	360	1	Urban
Row5	Male	Yes	2	Graduate	Yes	541700	419,600	26700000	360	1	Urban
Row6	Male	Yes	0	Not Graduate	No	233300	151,600	9500000	360	1	Urban
Row7	Male	Yes	3+	Graduate	No	303600	250,400	15800000	360	0	Semirurban
Row8	Male	Yes	2	Graduate	No	406600	152,600	16800000	360	1	Urban
Row9	Male	Yes	1	Graduate	No	1284100	1,066,800	34900000	360	1	Semirurban
Row10	Male	Yes	2	Graduate	No	320000	70,400	7000000	360	1	Urban
Row11	Male	Yes	2	Graduate	Yes	253000	84,400	10500000	360	1	Urban
Row12	Male	Yes	2	Graduate	No	307300	910,600	20000000	360	1	Urban
Row13	Male	No	0	Graduate	No	185300	284,000	11400000	360	1	Rural
Row14	Male	Yes	2	Graduate	No	129900	108,600	1700000	120	1	Urban
Row15	Male	No	0	Graduate	No	495000	0	1250000	360	1	Urban
Row16	Male	No	1	Not Graduate	No	359600	0	1000000	240	?	Urban
Row17	Female	No	0	Graduate	No	351000	0	7600000	360	0	Urban
Row18	Female	No	0	Not Graduate	No	480700	0	1300000	360	1	Rural
Row19	Male	Yes	0	Graduate	?	260000	359,000	11500000	360	1	Urban
Row20	Male	Yes	0	Not Graduate	No	766000	0	10400000	360	0	Urban
Row21	Male	Yes	1	Graduate	No	595500	562,500	3150000	360	1	Urban
Row22	Male	Yes	0	Not Graduate	No	260000	191,100	1160000	360	0	Semirurban
Row23	Male	Yes	2	Not Graduate	No	336300	191,700	1120000	360	0	Rural
Row24	Male	Yes	1	Graduate	?	371700	292,500	1510000	360	0	Urban
Row25	Male	Yes	0	Graduate	Yes	958000	0	1910000	360	1	Semirurban
Row26	Male	Yes	0	Graduate	No	279900	225,300	1220000	360	1	Semirurban
Row27	Male	Yes	2	Not Graduate	No	422600	104,000	1100000	360	1	Urban
Row28	Male	No	0	Not Graduate	No	144200	0	350000	360	1	Urban
Row29	Female	No	2	Graduate	?	375000	208,300	1200000	360	1	Semirurban
Row30	Male	Yes	1	Graduate	?	416000	336,900	301000	360	?	Urban
Row31	Male	No	0	Graduate	No	315700	0	740000	360	1	Urban
Row32	Male	No	1	Graduate	Yes	469200	0	1060000	360	1	Rural
Row33	Male	Yes	0	Graduate	No	350000	166,700	1140000	360	1	Semirurban
Row34	Male	No	3+	Graduate	No	1250000	300,000	3200000	360	1	Rural
Row35	Male	Yes	0	Graduate	No	227500	206,700	0	360	1	Urban
Row36	Male	Yes	0	Graduate	No	182800	133,000	1000000	360	0	Urban
Row37	Female	Yes	0	Graduate	No	366700	145,900	1440000	360	1	Semirurban

Figura 3.1: Vista preliminar de BD1

- Base de datos 2 (BD2, Figura 3.2): presenta 569 instancias y 31 atributos, uno de ellos de tipo *string* y los restantes de tipo numérico. La columna objetivo es de tipo binaria.

Row ID	S dispos	D revalue	D perm...	D w...	D month...	D compe...	D com...	D comm...	D Total...	D balanc...	D duration	D credit...	D addr...	D age...	D amount...	D compl...	D con...	D credit...	D month...	D revalue	D perm...	
Row0	M	16.37	15.77	123.9	126.8	6.065	6.076	0.667	9.47	9.457	5.542	3.704	2.366	74.04	9.365	0.013	0.019	0.014	24.49	25.41	16.8	
Row1	M	11.42	26.28	75.58	106.1	1.142	0.944	0.241	8.241	8.256	8.256	0.997	0.999	1.196	3.40	0.012	0.017	0.007	9.19	9.6	9.09	
Row2	M	12.48	18.7	80.57	109.1	0.991	0.978	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row3	M	13.71	26.42	80.5	102.8	0.991	0.978	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row4	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row5	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row6	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row7	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row8	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row9	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row10	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row11	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row12	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row13	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row14	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row15	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row16	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row17	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row18	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row19	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row20	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row21	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row22	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54
Row23	M	16.37	24.24	103.7	109.8	0.982	0.967	0.209	8.776	8.776	8.776	0.978	0.978	0.978	31.04	0.008	0.019	0.011	0.012	19.42	23.79	16.54

Row ID	s26	s Column27	s Column28	s Column29	s Column30	s Column31	s Column32	s Column33	s Column34	s Column35	s Column36
Row0	f	f	f	f	f	t	t	t	t	n	won
Row1	f	f	f	f	f	t	t	t	t	n	won
Row2	f	f	f	f	f	t	t	t	t	n	won
Row3	f	f	f	f	f	t	t	t	t	n	won
Row4	f	f	f	f	f	t	t	t	t	n	won
Row5	f	f	f	f	f	t	t	t	t	n	won
Row6	f	f	f	f	f	t	t	t	t	n	won
Row7	f	f	f	f	f	t	t				

- Base de datos 4 (BD4, Fig10) presenta 743 instancias y 12 atributos, 7 de ellos de tipo String, el restante de tipo numérico. La columna objetivo es de tipo múltiple.

Row ID	S Status	S Employed	D Loans	D Bank_Acc...	D Family	D Joint_S...	D Loan_Am...	D Term	S Gender	S Location	S Education	S Married
Row0	Y	n	1	0	0.07	0.214	0.744	M	Urban	Yes	No	
Row1	N	n	1	0.036	1	0.055	0.183	0.744	M	Rural	Yes	Yes
Row3	Y	n	1	0.057	0	0.03	0.171	0.744	M	Urban	No	Yes
Row4	Y	n	1	0	0	0.072	0.201	0.744	M	Urban	Yes	No
Row5	Y	y	1	0.101	2	0.03	0.31	0.744	M	Urban	Yes	Yes
Row6	Y	n	1	0.056	0	0.027	0.136	0.744	M	Urban	No	Yes
Row7	N	n	0	0.06	0	0.036	0.226	0.744	M	Semurban	Yes	No
Row8	Y	n	1	0.037	2	0.04	0.24	0.744	M	Urban	Yes	Yes
Row10	Y	n	1	0.017	2	0.038	0.1	0.744	M	Urban	Yes	Yes
Row11	Y	n	1	0.044	2	0.029	0.156	0.744	M	Urban	Yes	Yes
Row12	Y	n	1	0.195	2	0.059	0.286	0.744	M	Urban	Yes	Yes
Row13	N	n	1	0.068	0	0.03	0.13	0.744	M	Rural	Yes	No
Row14	Y	n	1	0.026	2	0.014	0.024	0.231	M	Urban	Yes	Yes
Row15	Y	n	1	0	0	0.059	0.179	0.744	M	Urban	Yes	No
Row18	N	n	1	0	0	0.059	0.19	0.744	M	Rural	No	Yes
Row20	N	n	0	0	0	0.093	0.149	0.744	M	Urban	No	Yes
Row22	N	n	0	0.046	0	0.03	0.166	0.744	M	Semurban	No	Yes
Row23	N	n	0	0.046	2	0.04	0.16	0.744	M	Rural	No	Yes
Row24	N	n	1	0.055	0	0.044	0.205	0.744	M	Semurban	Yes	Yes
Row26	N	n	1	0.054	0	0.033	0.174	0.744	M	Semurban	Yes	Yes
Row27	Y	n	1	0.025	2	0.05	0.157	0.744	M	Urban	No	Yes
Row28	N	n	1	0	0	0.018	0.05	0.744	M	Urban	No	No
Row29	Y	n	1	0.05	2	0.045	0.171	0.744	M	Semurban	Yes	No
Row32	N	y	1	0	1	0.056	0.151	0.744	M	Rural	Yes	No
Row35	Y	n	1	0.05	0	0.026	0	0.744	M	Urban	Yes	Yes
Row37	N	n	1	0.055	0	0.044	0.206	0.744	M	Semurban	Yes	Yes
Row39	Y	n	1	0.04	0	0.045	0.157	0.744	M	Semurban	No	No
Row40	N	n	1	0	0	0.043	0.114	0.744	M	Urban	Yes	No
Row41	Y	n	1	0.029	0	0.02	0.067	0.744	M	Urban	Yes	No
Row42	Y	n	1	0	0	0.028	0.107	0.744	M	Urban	Yes	Yes
Row43	Y	n	1	0.056	0	0.047	0.191	0.744	M	Semurban	Yes	Yes
Row44	Y	y	1	0	0	0.045	0.137	0.744	M	Urban	No	Yes
Row45	N	n	1	0	0	0.04	0.158	0.744	M	Urban	Yes	No
Row46	Y	n	1	0	1	0.068	0.063	0.744	M	Urban	Yes	Yes
Row49	Y	n	1	0.055	0	0.048	0.204	0.744	M	Semurban	Yes	No
Row51	Y	n	1	0	0	0.036	0.171	0.744	M	Semurban	Yes	No
Row53	N	n	1	0	2	0.055	0.191	0.744	M	Urban	Yes	Yes
Row54	N	n	y	0	0	0.14	0.409	0.744	M	Urban	Yes	Yes

Figura 3.4: Vista preliminar de la BD4

3.2. Casos de prueba al subcomponente *Discretizer*

En esta sección se presentan las pruebas al subcomponente *Discretizer*, una vez integrado al Componente AutoML Clasificación (*pre-procesado*). No se realizan pruebas individuales, dado que depende de variables de flujo configuradas en el componente AutoML.

Para la realización de las pruebas, se hace una copia del metanodo ID3, donde se realiza la integración del subcomponente *Discretizer* (figura 3.5) para comparar los resultados entre un conjunto de datos con una discretización predefinida, y uno con la discretización automatizada. Cabe destacar que el nodo *AutoBinner*, donde se encuentra la discretización previamente configurada, el método empleado es *equal-width* y la cantidad de bins es k = 5.

El método de comparación para los algoritmos es el valor obtenido por la precisión del método de clasificación (*accuracy*), el valor del Cohen's Kappa y el del área bajo la curva ROC (*AUC*).

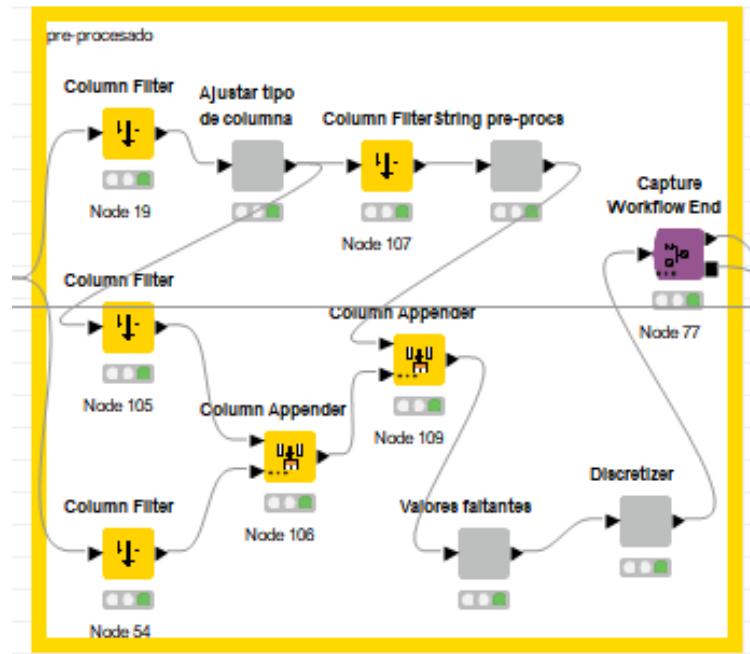


Figura 3.5: Integración del subcomponente *Discretizer*

Para la BD1, se ejecuta el componente, donde el método de discretización que presenta mejores resultados es *equal-width*, con *accuracy* de 0.727 y un Cohen's Kappa de 0.39. Los resultados de la comparación de los métodos de discretización se encuentran en la figura 3.6; así como la salida del subcomponente *Discretizer* en la figura 3.7.

⚠ Outport 1 - 4:2:0:108:46:0:2:0:47 - Compare Perfomance

Row ID	Accuracy	Cohen'...	rank
id3-width	0.727	0.39	1
id3-freq	0.707	0.382	2
id3-quantile	0.653	0.235	3
id3-caim	0.627	0.226	4

Figura 3.6: Comparación de métodos de discretización para la BD1

Al ejecutarse el flujo completo, se obtiene como resultado que el algoritmo con la discretización automatizada (ID3 #1) presenta mejores resultados, con un *accuracy* de 0.744 y un Cohen's Kappa de 0.456; a diferencia del algoritmo ID3 con discretización predefinida (ID3), con *accuracy* de 0.697 y Cohen's Kappa de 0.3372. Estos resultados se encuentran en la figura 3.8.

Capítulo 3. Integración y validación de soluciones propuestas al componente de AutoMIS1

Output 1 - 4:2:0:108:46:0:2 - Discretizer						
File Edit Hilite Navigation View						
Table "default" - Rows: 499 Spec - Columns: 12 Properties Flow Variables						
Row ID	\$ Applicant_Income [Binned]	\$ Coapplicant_Income [Binned]	\$ Loan_Amount [Binned]	\$ Term [Binned]	\$ Credit_History [Binned]	
Row0	[15,000,823,500]	[0,416,670]	(14,000,000,21,000,000)	(399,6,386,-4)	[0,9,-1]	
Row1	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row2	[15,000,823,500]	[0,416,670]	[0,7,000,000]	(399,6,386,-4)	[0,9,-1]	
Row3	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row4	[15,000,823,500]	[0,416,670]	(14,000,000,21,000,000)	(399,6,386,-4)	[0,9,-1]	
Row5	[15,000,823,500]	[416,670,833,340]	(21,000,000,28,000,000)	(399,6,386,-4)	[0,9,-1]	
Row6	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row7	[15,000,823,500]	[0,416,670]	(14,000,000,21,000,000)	(399,6,386,-4)	[0,9,-1]	
Row8	[15,000,823,500]	[0,416,670]	(14,000,000,21,000,000)	(399,6,386,-4)	[0,9,-1]	
Row9	[15,000,823,500]	[0,416,670,1,420,0,10]	(28,000,000,35,000,000)	(399,6,386,-4)	[0,9,-1]	
Row10	[15,000,823,500]	[0,416,670]	[0,7,000,000]	(399,6,386,-4)	[0,9,-1]	
Row12	[15,000,823,500]	[416,670,833,340]	(14,000,000,21,000,000)	(399,6,386,-4)	[0,9,-1]	
Row13	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row14	[15,000,823,500]	[0,416,670]	[0,7,000,000]	(105,6,152,-4)	[0,9,-1]	
Row15	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row17	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row18	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row20	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row21	[15,000,823,500]	[416,670,833,340]	(28,000,000,35,000,000)	(399,6,386,-4)	[0,9,-1]	
Row22	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row25	(823,500,1,632,0,00)	[0,416,670]	(14,000,000,21,000,000)	(399,6,386,-4)	[0,9,-1]	
Row26	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row27	[15,000,823,500]	[0,416,670]	(7,000,000,14,000,000)	(399,6,386,-4)	[0,9,-1]	
Row28	[15,000,823,500]	[0,416,670]	[0,7,000,000]	(399,6,386,-4)	[0,9,-1]	

Figura 3.7: Salida del sucomponente *Discretizer* para la BD1



Figura 3.8: Comparación de los resultados de ambos algoritmos para la BD1

Para la BD2, se ejecuta el componente, donde el método de discretización que presenta mejores resultados es CAIM, con un *accuracy* de 0.936 y un *Cohen's Kappa* de 0.865. Los resultados de la comparación de los métodos de discretización se encuentran en la figura 3.9; así como la salida del subcomponente *Discretizer* en la figura 3.10.

Outport 1 - 4:2:0:108:46:0:2:0:47 - Compare Perfomance

Table "default" - Rows: 4 Spec - Columns: 3 Properties Flow Variables			
Row ID	Accuracy	Cohen's kappa	rank
id3-caim	0.936	0.865	1
id3-quantile	0.924	0.841	2
id3-freq	0.906	0.799	3
id3-width	0.895	0.771	4

Figura 3.9: Comparación de métodos de discretización para la BD2

Row ID	S diagnosis	S radius_w...	S texture_w...	S perimeter...	S area_m...	S smooth...	S compact...	S concav...	S concav...	S symmet...	S...
Row0	m	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row1	m	Interval_1	Interval_0	Interval_1	Interval_1	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Inte
Row2	m	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row3	m	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row4	m	Interval_1	Interval_0	Interval_1	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_0	Inte
Row5	m	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row6	m	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_0	Inte
Row7	m	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row8	m	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row9	m	Interval_0	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row10	m	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Inte
Row11	m	Interval_1	Interval_0	Interval_1	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_0	Inte
Row12	m	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row13	m	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_0	Inte
Row14	m	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row15	m	Interval_0	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row16	m	Interval_0	Interval_1	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Inte
Row17	m	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Interval_1	Inte
Row18	m	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_0	Inte
Row19	b	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Inte
Row20	b	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Inte
Row21	b	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Interval_0	Inte
Row22	m	Interval_1	Interval_0	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_1	Inte
Row23	m	Interval_1	Interval_1	Interval_1	Interval_1	Interval_0	Interval_0	Interval_1	Interval_1	Interval_0	Inte

Figura 3.10: Salida del subcomponente Discretizer para la BD2

Al ejecutarse el flujo completo, una vez más el algoritmo con mejores resultados fue ID3 # 1, donde el *accuracy* alcanzó 0.9851, y el AUC un valor de 0.9797; a diferencia del algoritmo ID3 con 0.8818 y 0.9314, respectivamente. En la figura 3.11 se muestran los resultados de las comparaciones entre los dos algoritmos de clasificación.

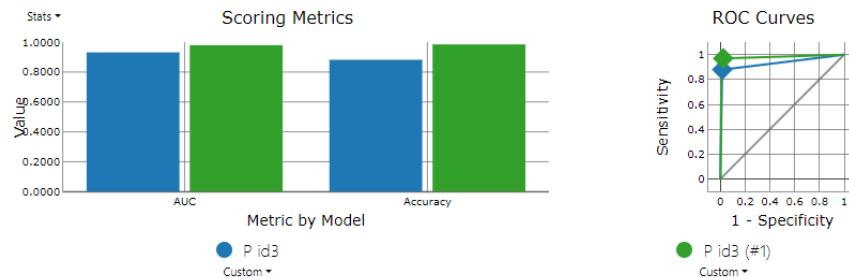


Figura 3.11: Comparación de resultados entre ambos algoritmos para la BD2

La última prueba realizada es con el objetivo de validar la entrada al componente, dado que el nodo *CAIM Binner* solamente admite valores numéricos. Por tanto, se emplea una base de datos que solamente contiene atributos de tipo *string*, para comprobar que funciona la validación realizada. En el anexo A.6 se puede observar que todas las columnas son nominales, así como en la figura 3.12 se comprueba la salida del subcomponente *Discretizer* de la misma base de datos sin ninguna modificación.

Row ID	S Column36	S Column0	S Column1	S Column2	S Column3	S Column4	S Column5	S Column6	S Column7	S Column8	S
Row0	won	f	f	f	f	f	f	f	f	f	f
Row1	won	f	f	f	t	f	f	f	f	f	f
Row2	won	f	f	f	t	f	t	f	f	f	f
Row3	won	f	f	f	f	f	f	f	f	f	f
Row4	won	f	f	f	f	f	f	f	f	f	f
Row5	won	f	f	f	f	f	f	f	f	f	f
Row6	won	f	f	f	f	f	f	f	f	f	f
Row7	won	f	f	f	t	f	f	f	f	f	f
Row8	won	f	f	f	f	f	f	f	f	f	f
Row9	won	f	f	f	f	f	t	f	f	f	f
Row10	won	f	f	f	t	f	t	f	f	f	f
Row11	won	f	f	f	f	f	f	f	t	f	f
Row12	won	f	f	f	f	f	f	f	t	f	f
Row13	won	f	f	f	f	f	f	f	t	f	f
Row14	won	f	f	f	t	f	f	f	t	f	f
Row15	won	f	f	f	t	f	f	f	f	f	f
Row16	won	f	f	f	f	f	f	f	f	f	f
Row17	won	f	f	f	t	f	t	f	f	f	f
Row18	won	f	f	f	f	f	f	f	f	t	f
Row19	won	f	f	f	f	f	f	f	f	t	f
Row20	won	f	f	f	f	f	f	f	f	t	f
Row21	won	f	f	f	t	f	f	f	f	t	f
Row22	won	f	f	f	f	f	f	f	f	f	f
Row23	won	f	f	f	f	f	f	f	f	f	f

Figura 3.12: Salida el subcomponente *Discretizer* para la BD3

3.3. Casos de prueba del Componente AutoML (*Optimización de Hiperparámetros*)

A continuación, se presenta el correcto funcionamiento del *Componente AutoML (Optimización de Hiperparámetros)*, al suministrarle las bases de datos expuestas previamente. En cada caso, se muestra una comparación entre la aplicación del algoritmo Red neuronal por retro-propagación (RProp), con y sin la utilización del componente propuesto. En la figura 3.13 se muestra el flujo creado para la comparación de los algoritmos.

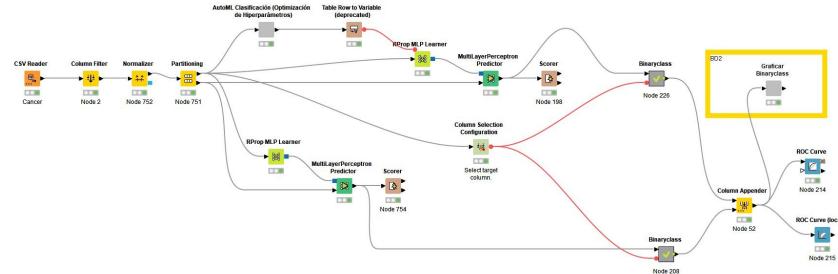


Figura 3.13: Flujo para la comparación de algoritmos

3.3.1. Pruebas individuales al Componente AutoML (*Optimización de Hiperparámetros*)

Para la BD1 se ejecuta correctamente el componente. En la figura 3.14, se muestra la evaluación del rendimiento del modelo en función de las clases positivas y negativas

mediante el área bajo la curva (AUC) y la efectividad de clasificación (accuracy). En la figura 3.15 se presenta una comparación de los modelos mediante una curva ROC.

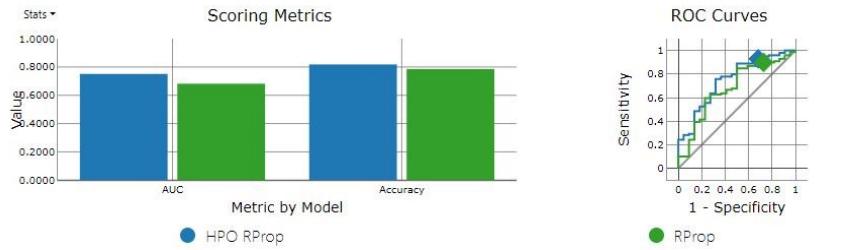


Figura 3.14: Evaluación del rendimiento del modelo con BD1

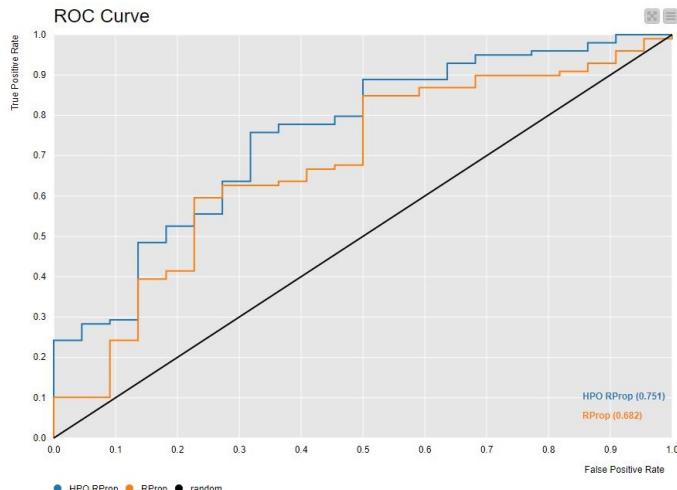


Figura 3.15: Curva ROC comparación de modelos con BD1

Como se evidencia en la figura 3.14, el algoritmo optimizado (HPO RProp) presenta un AUC 0.75 y un *accuracy* 0.82, mayor que el del algoritmo sin optimizar (RProp) con AUC 0.68 y *accuracy* 0.78; lo que se traduce a que el modelo HPO RProp puede distinguir con mayor precisión entre las clases positivas y negativas, presentando mejor capacidad para clasificar correctamente los datos, con respecto a RProp. En la figura 3.15, se observa que HPO RProp llega más rápido al óptimo global, logrando un mejor rendimiento en menos tiempo de entrenamiento, o iteraciones, que el modelo representado por RProp.

Para la BD2 se ejecuta correctamente el componente. En la figura 3.16, se muestra la evaluación del rendimiento del modelo en función de las clases positivas y negativas mediante el AUC y el *accuracy*. En la figura 3.17, se presenta una comparación de los modelos mediante una curva ROC.

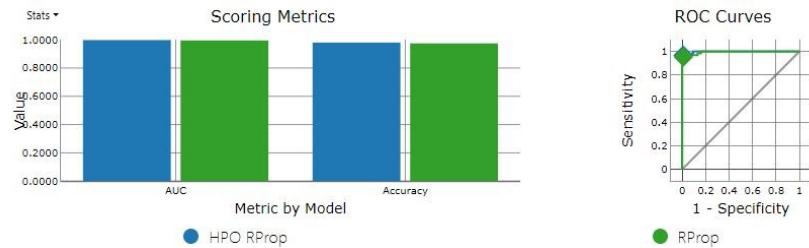


Figura 3.16: Evaluación del rendimiento del modelo para la BD1

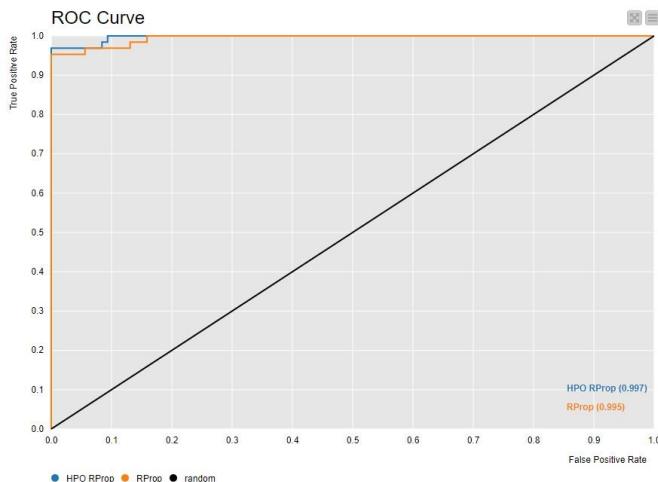


Figura 3.17: Curva ROC de comparación de modelos para la BD2

Como se evidencia en la figura 3.16, el algoritmo optimizado (HPO RProp) presenta un AUC 0.997 y un *accuracy* 0.979, mayor que el del algoritmo sin optimizar (RProp), con AUC 0.994 y *accuracy* 0.973. No obstante, la diferencia no es significativa, lo que se puede traducir a que el modelo HPO RProp y el modelo RProp tienen la misma capacidad para clasificar correctamente las muestras en términos generales. En la figura 3.17, se observa que HPO RProp llega más rápido al óptimo global, logrando un mejor rendimiento en menos tiempo de entrenamiento, o iteraciones, que el modelo representado por RProp.

3.3.2. Pruebas de integración al Componente AutoML Clasificación (*pre-procesado*)

En la figura 3.18 se presenta la correcta integración del *Componente AutoML Clasificación (Optimización de Hiperparámetros)* al *Componente AutoML Clasificación (pre-procesado)*. Se ejecuta una prueba con la BD4 (figura 3.19) para validar el funcionamiento correcto de la integración.

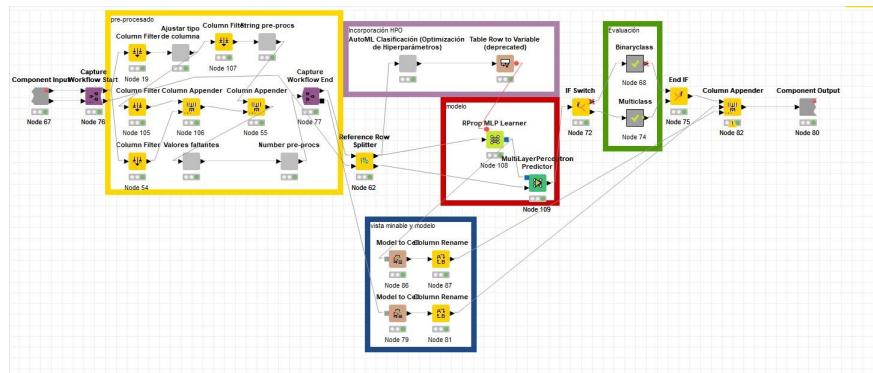


Figura 3.18: Integración de ambos componentes.

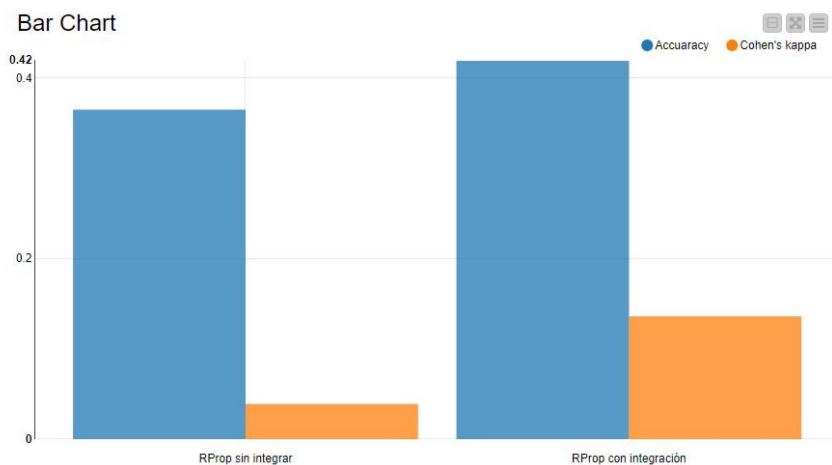


Figura 3.19: Evaluación de la integración con la BD4

En la figura 3.19 se aprecia que el componente integrado presenta un mayor *accuracy* (0.42) y *Cohen's Kappa* (0.14) con respecto al componente sin integrar, con un *accuracy* de 0.36 y *Cohen's Kappa* de 0.03. Por este motivo, se puede determinar que el componente con la integración presenta una mejor capacidad para clasificar correctamente los datos, así como una mayor consistencia en las respuestas de los evaluadores. En consecuencia, se puede tener una mayor confianza en los resultados.

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

- Abiodun, Oludare Isaac, Jantan, Aman, Omolara, Abiodun Esther, Dada, Kemi Victoria, Mohamed, Nachaat AbdElatif, & Arshad, Humaira. 2018. State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), e00938.
- Agrawal, Rakesh, Imielinski, Tomasz, & Swami, Arun. 1993. Mining Association Rules between Sets of Items in Large Databases.
- 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.*
- 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.
- Chandrashekhar, Girish, & Sahin, Ferat. 2014. A survey on feature selection methods. *Computers & Electrical Engineering*, **40**(1), 16–28.
- 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.
- 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.
- 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."
- Guenther, Nick, & Schonlau, Matthias. 2016. Support vector machines. *The Stata Journal*, **16**(4), 917–937.
- Gupta, Bhumika, Rawat, Aditya, Jain, Akshay, Arora, Arpit, & Dhami, Naresh. 2017. Analysis of various decision tree algorithms for classification in data mining. *International Journal of Computer Applications*, **163**(8), 15–19.
- Hastie, Trevor, Tibshirani, Robert, Friedman, Jerome H, & Friedman, Jerome H. 2009. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer.
- He, Xin, Zhao, Kaiyong, & Chu, Xiaowen. 2021. AutoML: A survey of the state-of-the-art. *Knowledge-Based Systems*, **212**, 106622.

- Hernández, Eduardo Sánchez-Jiménez Yasmín, & Ortiz-Hernández, Javier. 2017. Técnicas de Optimización de Hiperparámetros en Modelos de Aprendizaje Automático para Predicción de Enfermedades Cardiovasculares.
- Hernández Orallo, José, et al. 2004. *Introducción a la Minería de Datos*. Biblioteca Hernán Malo González.
- 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.
- Javed Mehedi Shamrat, FM, Ranjan, Rumesh, Hasib, Khan Md, Yadav, Amit, & Siddique, Abdul Hasib. 2022. Performance evaluation among ID3, C4.5, and CART Decision Tree Algorithm. *Pages 127–142 of: Pervasive Computing and Social Networking: Proceedings of ICPCSN 2021*. Springer.
- Jiawei Han, Micheline Kamber, Jian Pei. 2011. *Data Mining. Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. 3rd edn.
- Joachims, Thorsten. 2002. *Learning to classify text using support vector machines*. Vol. 668. Springer Science & Business Media.
- KNIME. 2023 (Mar.). *KNIME Official Site*.
- Kurgan, Lukasz A, & Cios, Krzysztof J. 2004. CAIM discretization algorithm. *IEEE transactions on Knowledge and Data Engineering*, **16**(2), 145–153.
- 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, Hussain, Farhad, Tan, Chew Lim, & Dash, Manoranjan. 2002. Discretization: An enabling technique. *Data mining and knowledge discovery*, **6**, 393–423.
- 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.
- 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.
- 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.
- 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.
- Waring, Jonathan, Lindvall, Charlotta, & Umeton, Renato. 2020. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial intelligence in medicine*, **104**, 101822.

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

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

Anexo A

Anexos

A.1. Flujo KNIME del componente AutoML Clasificacion (pre-procesado)

A.2. Flujo KNIME del procesamiento de RProp con HPO

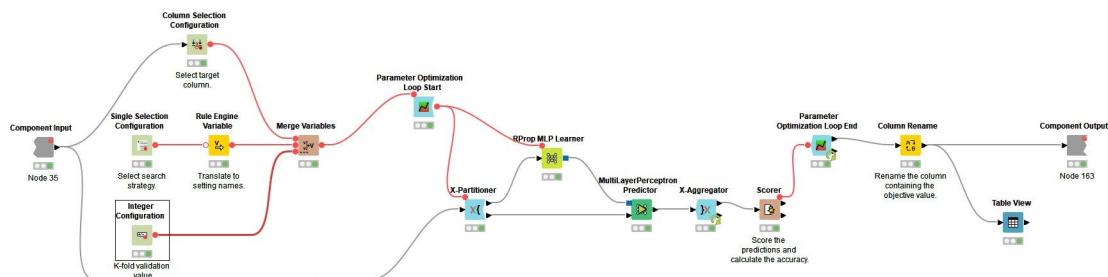


Figura A.2: Flujo KNIME del procesamiento de RProp con HPO

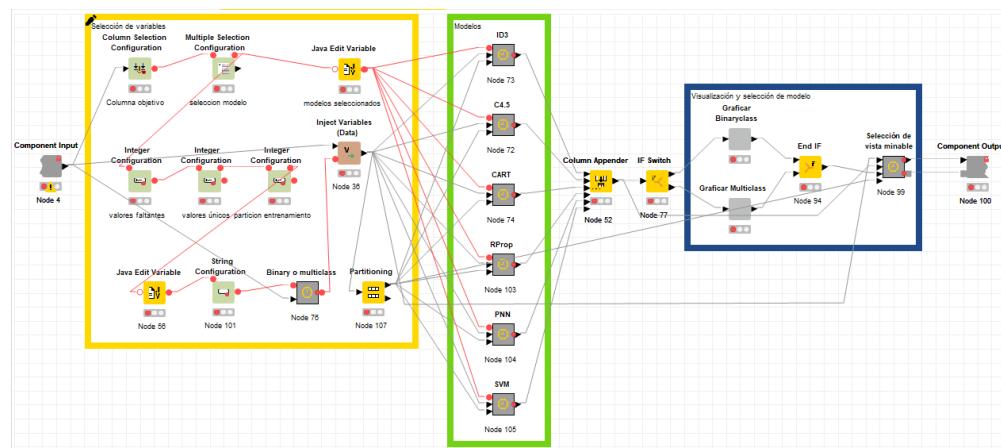


Figura A.1: Flujo KNIME del Componente AutoML Clasificación (pre-procesado) (Carraza-na Ruiz, 2022)

A.3. Flujo KNIME de subcomponente para la discretización para el algoritmo ID3

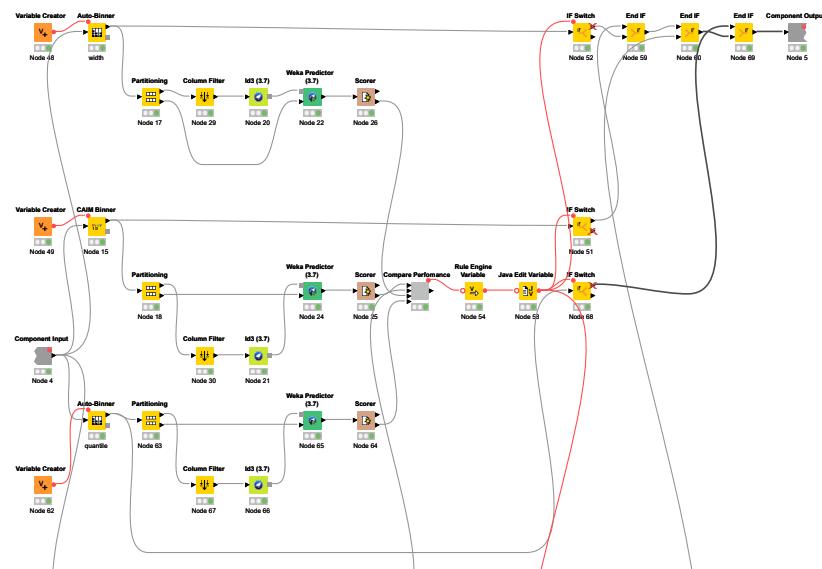


Figura A.3: Flujo KNIME de subcomponente para la discretización para el algoritmo ID3

A.4. Flujo KNIME de la ejecución del método CAIM para la discretización orientada al algoritmo ID3

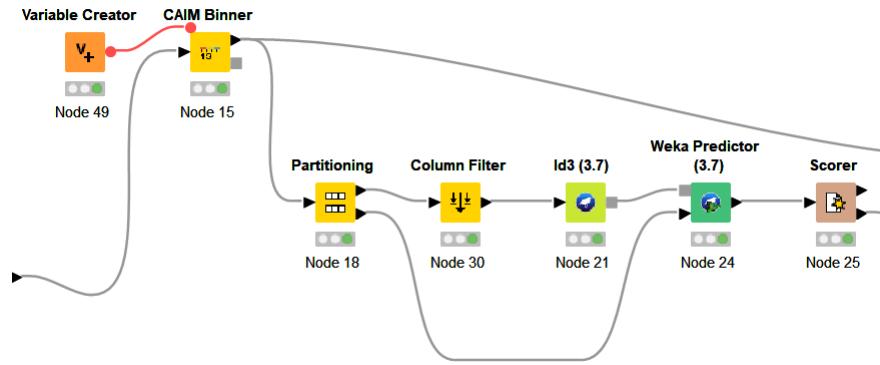


Figura A.4: Flujo KNIME de la ejecución del método CAIM para la discretización orientada al algoritmo ID3

A.5. Flujo KNIME para la comparación de métodos de discretización acorde al algoritmo ID3

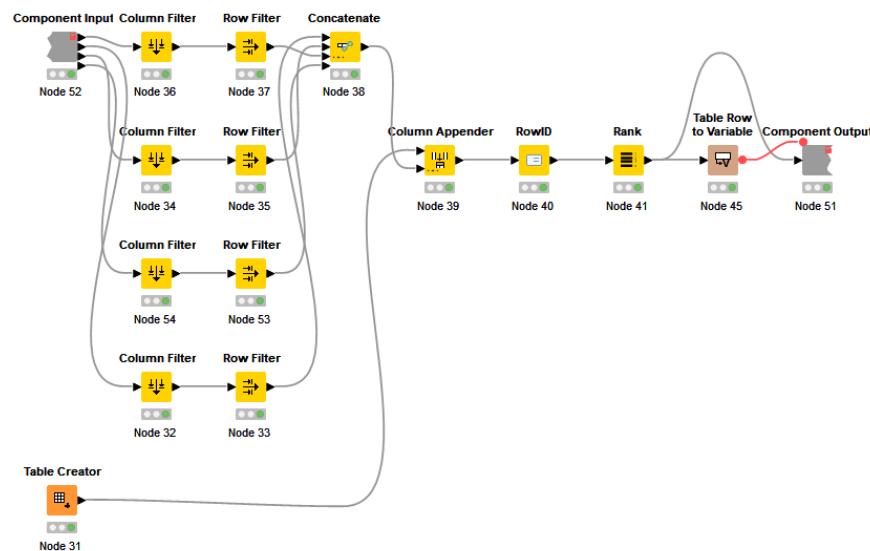


Figura A.5: Flujo KNIME para la comparación de métodos de discretización acorde al algoritmo ID3

A.6. Columnas de tipo String en BD3

File Table - 4:3 - File Reader

File

Table "default" - Rows: 3196 Spec - Col

Colum...	Column Type	Column Index
Column0	String	0
Column1	String	1
Column2	String	2
Column3	String	3
Column4	String	4
Column5	String	5
Column6	String	6
Column7	String	7
Column8	String	8
Column9	String	9
Column10	String	10
Column11	String	11
Column12	String	12
Column13	String	13
Column14	String	14
Column15	String	15
Column16	String	16
Column17	String	17
Column18	String	18
Column19	String	19
Column20	String	20
Column21	String	21
Column22	String	22
Column23	String	23
Column24	String	24
Column25	String	25
Column26	String	26
Column27	String	27
Column28	String	28
Column29	String	29
Column30	String	30
Column31	String	31
Column32	String	32
Column33	String	33
Column34	String	34
Column35	String	35
Column36	String	36

Figura A.6: Columnas de tipo String en BD3