La investigación reportada en esta tesis es parte de los programas de investigación del CICESE (Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California).
La investigación fue financiada por el CONAHCYT (Consejo Nacional de Humanidades, Ciencias y Tecnologías).

Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California

Maestría en Ciencias
en Ciencias de la Computación

Predicción de actividad antimicrobiana usando modelos de escala evolutiva a través de un flujo de trabajo en la plataforma KNIME

Tesis

para cubrir parcialmente los requisitos necesarios para obtener el grado de

Maestro en Ciencias

Presenta:

Karla Lorena Martínez Mauricio

Ensenada, Baja California, México

2023

Tesis defendida por

Karla Lorena Martínez Mauricio

y aprobada por el siguiente Comité

Dr. César Raúl García Jacas
Director de tesis

Dr. Carlos Alberto Brizuela Rodríguez

Dra. Johanna Bernáldez Sarabia

Dr. Israel Marck Martínez Pérez

Dr. Pedro Gilberto López Mariscal
Coordinador del Posgrado en Ciencias de la Computación

Dra. Ana Denise Re Araujo
Directora de Estudios de Posgrado

Resumen de la tesis que presenta Karla Lorena Martínez Mauricio como requisito parcial para la obtención del grado de Maestro en Ciencias en Ciencias de la Computación.

Predicción de actividad antimicrobiana usando modelos de escala evolutiva a través de un flujo de trabajo en la plataforma KNIME

Resumen aprobado por:

Dr. César Raúl García Jacas
Director de tesis

Dentro de las estrategias para combatir la resistencia antimicrobiana, se está llevando a cabo investigación para la creación de nuevos fármacos basados en péptidos antimicrobianos. En los últimos años, se han realizado esfuerzos para incorporar herramientas computacionales que ayuden a acelerar la identificación de péptidos con actividad antimicrobiana. Una de estas herramientas son los modelos QSAR basados en aprendizaje tradicional, que permiten predecir la actividad antimicrobiana en péptidos a partir de información basada en su secuencia. Un componente clave en este proceso es el tipo de características moleculares a utilizar. Recientemente, ha surgido una familia de modelos pre-entrenados llamados ESM-2, los cuales generan incrustaciones (características) que fueron aprendidas a partir de 65 millones de secuencias que abarcan diversidad evolutiva. En este trabajo de tesis, se analiza la contribución de las incrustaciones ESM-2 de diferentes dimensiones de forma individual y en conjunto en el desarrollo de modelos QSAR basados en aprendizaje tradicional para la clasificación de péptidos antimicrobianos, así como sus tipos funcionales, como antibacteriano, antifúngico y antiviral. A partir de este estudio se concluye que aumentar la capacidad de los modelos ESM-2 no implica una mejora en el rendimiento de los modelos para predecir péptidos antimicrobianos. Los modelos ESM-2_t30 y ESM-2_t33 son los más apropiados para extraer características y mejorar la exactitud en las predicciones de péptidos antimicrobianos. Además, fusionar características de diferentes incrustaciones ESM-2 es una estrategia efectiva para construir mejores modelos QSAR que el uso exclusivo de características derivadas de un modelo ESM-2 específico. Se construyeron modelos más simples con un rendimiento comparable o superior a los modelos basados en aprendizaje profundo reportados en la literatura. Para llevar a cabo este estudio se implementó un flujo de trabajo en KNIME que genera de forma automática hasta 1980 modelos de clasificación binaria basados en aprendizaje tradicional. Incorpora diversas técnicas de selección de características, algoritmos de clasificación, métricas de desempeño y una fase de limpieza de datos. Este flujo de trabajo se encuentra disponible en `https://github.com/cicese-biocom/classification-QSAR-bioKom`.

Palabras clave: péptidos antimicrobianos, QSAR, aprendizaje automático ESM-2, KNIME

Abstract of the thesis presented by Karla Lorena Martínez Mauricio as a partial requirement to obtain the Master of Science degree in Computer Science.

Prediction of antimicrobial activity using models of evolutionary scale through a workflow in the KNIME platform

Abstract approved by:

PhD César Raúl García Jacas
Thesis Director

Molecular features play an important role in different bio-chem-informatics tasks, such as the Quantitative Structure-Activity Relationships (QSAR) modeling. Several pre-trained models have been recently created to be used in downstream tasks either by fine-tuning a specific model or by extracting features to feed traditional classifiers. In this sense, a new family of Evolutionary Scale Modeling models (termed as ESM-2 models) has been recently introduced, demonstrating outstanding results in structure protein prediction benchmarks. Herein, we are devoted to assessing the usefulness of different-dimensional embeddings derived from ESM-2 models in the prediction of antimicrobial peptides, given the great deal of attention received because of their potential to become a plausible option to mainly fight multi-drug resistant bacteria. To this end, we created a KNIME workflow to guarantee using the same modeling methodology, and consequently, carrying out fair comparisons. As a result, it can be drawn that the 640- and 1,280-dimensional embeddings are the most appropriate to be used in modeling because statistically better results were achieved from them. We also combined features from different embeddings, and we can draw that the fusion of features of different embeddings contributes to getting better models than only using a specific model ESM-2. Comparisons regarding state-of-the-art deep learning models confirm that when performing methodologically principled studies in the prediction of AMPs, non-DL based models yield comparable-to-superior results to DL-based models. The implemented KNIME workflow is available-freely at `https://github.com/cicese-biocom/classification-QSAR-bioKom`. We consider that this workflow can be valuable to prevent unfair comparisons regarding new computational methods, as well as to propose new non-DL based models.

Keywords: antimicrobial peptides, QSAR, machine learning, ESM-2, KNIME

Dedicatoria

A mi mamá y mi papá, a quienes les debo todo lo que soy, y a quienes siempre dedicaré todos mis logros.

Agradecimientos

Al Centro de Investigación Científica y de Educación Superior de Ensenada, Baja California por brindarme la oportunidad y los medios para concluir este proyecto de maestría.

Al Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT) por brindarme el apoyo económico para realizar mis estudios de maestría.

A mi director de tesis, el Dr. César Raúl García Jacas, por su constante apoyo, por guiarme, confiar en mí y por ser un ejemplo fundamental en mi formación como investigadora.

A mi comité de tesis por sus comentarios y observaciones que ayudaron a mejorar mi trabajo.

A mi madre y mi padre por su amor y apoyo incondicional. Ustedes me enseñaron a nunca rendirme y sin ustedes, no sería quien soy hoy. Siempre estaré agradecida con ustedes por todo.

A mis hermanas por las risas, el apoyo y la compañía que me han brindado todos estos años. A mis sobrinos por traer alegría y llenar nuestros corazones con ternura, cariño y orgullo.

A Dann por ser mi compañero de vida, mi equipo, por amarme, apoyarme, escucharme y nunca juzgarme. Por ser sol e iluminar mi mundo desde el instante en que te conocí. A Coco por llenar nuestros días de ronroneos.

A todos mis amigos que han traído alegría a mi vida, a mis conocidos en Ensenada por crear bonitos recuerdos que siempre llevaré conmigo, a los Pelícanos Viajeros por mostrarme lo hermoso que es esta ciudad y a todas las personas que, aunque no menciono explícitamente, han contribuido significativamente a mi vida.

Tabla de contenido

Página

Lista de figuras

Figura Página

Lista de tablas

Capítulo 1. Introducción

Los medicamentos antimicrobianos se usan para tratar y prevenir infecciones. Estos engloban a los antibióticos, antivirales, antifúngicos y antiparasitarios, siendo un ejemplo de cada uno de ellos la amoxicilina, el oseltamivir, el fluconazol y la cloroquina, respectivamente. Sin embargo, algunos gérmenes como las bacterias, los virus, los hongos y los parásitos están desarrollando resistencia a los fármacos antimicrobianos diseñados actualmente para combatirlos. Este hecho se conoce como resistencia antimicrobiana (AMR, por sus siglas en inglés) (Centers for Disease Control and Prevention, 2022). Por mencionar un ejemplo, la bacteria *Escherichia coli* ha generado resistencia al antibiótico ciprofloxacina con una tasa que varía entre el 8.4 % y el 92.9 % (World Health Organization, 2021).

La AMR representa una amenaza para los avances que la humanidad ha logrado en el tratamiento y curación de enfermedades. En el Espacio Económico Europeo se estima que en el 2020 se produjeron más de 800,000 infecciones causadas por bacterias resistentes a los antibióticos, y que más de 35,000 personas murieron como consecuencia directa de estas infecciones (European Centre for Disease Prevention and Control and World Health Organization, 2023). La AMR hace que las infecciones sean más difíciles de tratar, aumentando el riesgo de propagación de enfermedades, prolongando las estancias hospitalarias e incluso causando la muerte (World Health Organization, 2021). Además, puede tener un impacto significativo en la economía (World Health Organization, 2015), afectar la seguridad alimentaria y la producción de alimentos (Centers for Disease Control and Prevention, 2022).

Aunque la AMR es un proceso natural a medida que los gérmenes mutan y se adaptan a su entorno, hay factores que aceleran su propagación como el uso inadecuado y excesivo de medicamentos antimicrobianos (World Health Organization, 2015). Se estima que los consultorios médicos y los departamentos de emergencia de los EE. UU. prescriben cada año alrededor de 47 millones de tratamientos con antibióticos para infecciones que no necesitan antibióticos. Eso es alrededor del 30 % de todos los antibióticos recetados en estos entornos médicos (Centers for Disease Control and Prevention, 2019).

Actualmente se están llevando a cabo múltiples esfuerzos para combatir la AMR desde diferentes áreas claves. Una de ellas consiste en concientizar, educar y capacitar sobre esta problemática, además de implementar medidas de saneamiento, higiene y prevención de infecciones. También se busca optimizar el uso de medicamentos antimicrobianos existentes, y se está invirtiendo en el desarrollo de nuevas herramientas de diagnóstico y fármacos (World Health Organization, 2015). En este último ámbito, se está llevando a cabo investigación para la creación de fármacos basados en péptidos antimicrobianos.

Los péptidos antimicrobianos (AMPs, por sus siglas en inglés) son moléculas compuestas por la unión

de aminoácidos mediante enlaces peptídicos (Sewald & Jakubke, 2015). Los AMPs forman parte de la respuesta inmune innata de los organismos vivos, y presentan un amplio rango de actividades antibacterianas, antivirales, antifúngicas, antiparasitarias, entre otras (Divyashree et al., 2020), razón por la cual constituyen una alternativa prometedora en el diseño/descubrimiento de nuevos fármacos para combatir la AMR. Se ha observado que estas actividades antimicrobianas se deben a algunas características y propiedades comunes en casi todos los AMPs, tales como, carga neta positiva que varía de $+2$ a $+9$, son anfipáticos y, típicamente, presentan una hidrofobicidad del 50 % (Kumar et al., 2018).

Los experimentos de laboratorio para identificar AMPs son lentos y costosos. Por esta razón, en los últimos años se han realizado esfuerzos para incorporar herramientas computacionales que apoyen esta tarea (Agüero-Chapin et al., 2023, 2022). En particular, se han creado modelos de clasificación basados en el aprendizaje automático con el fin de predecir la posible actividad antimicrobiana de un péptido, así como para determinar su función específica. Estos predictores siguen el enfoque de modelación QSAR (Quantitative Structure-Activity Relationships), ya que buscan relacionar matemáticamente características fisicoquímicas extraídas de los péptidos con su correspondiente actividad biológica (Beltran et al., 2020; Tropsha, 2010).

1.1. Antecedentes

Existen varios modelos QSAR reportados en la literatura para la predicción de AMPs. Estos modelos se pueden clasificar por el tipo de técnica de aprendizaje que utilizaron. Por un lado, está el aprendizaje tradicional que utiliza algoritmos como Random Forest, Support Vector Machine, k-Nearest Neighbors, entre otros. Por otro lado, se encuentra el aprendizaje profundo (deep learning) que utiliza Redes Neuronales Convolucionales, Recurrentes, entre otras. Sin embargo, García-Jacas et al. (2022b) realizaron estudios para comparar de manera justa los modelos de aprendizaje profundo con los modelos de aprendizaje tradicional para la clasificación de péptidos antimicrobianos, revelando que ambos tipos de modelos codifican información química similar, y que los modelos de aprendizaje tradicional pueden ser más adecuados debido a su simplicidad y rendimiento comparable o superior en las pruebas realizadas sobre los conjuntos de datos actualmente disponibles. Hasta la fecha, el conjunto de datos de entrenamiento más grande se propuso en (Pinacho-Castellanos et al., 2021), el cual contiene 19,548 secuencias de péptidos para modelar AMPs y, aun así, no es lo suficientemente grande como para proponer nuevas arquitecturas de aprendizaje profundo (García-Jacas et al., 2022b).

Un componente clave en la construcción de modelos QSAR basados en aprendizaje tradicional es el tipo de características moleculares a ser usadas. Hasta la fecha, se han introducido varios algoritmos para calcular estas características acorde a diferentes teorías (Todeschini & Consonni, 2002; Valdés-Martiní et al., 2017; Romero-Molina et al., 2019; García-Jacas et al., 2020). Sin embargo, dado que no existe un solo algoritmo capaz de extraer toda la información química para diferentes conjuntos de datos porque la relevancia de las características depende de las moléculas bajo estudio, es necesario utilizar varios algoritmos teóricamente diferentes para calcular un conjunto inicial y diverso de características (descriptores) moleculares a partir del cual se seleccionen las más útiles para la tarea de modelación subyacente (Bolón-Canedo & Alonso-Betanzos, 2019; Pes, 2019; García et al., 2019). Este enfoque conlleva a dos problemas. El primero surge cuando se calculan las características porque generalmente implicará tratar con conjuntos de alta dimensionalidad. El segundo inconveniente se relaciona con la complejidad de la selección de características, ya que el conjunto potencia (2^n) del total de características (n) es el espacio de posibles subcojuntos a analizar. Por lo tanto, obtener un espacio útil de características de menor dimensión a partir de un conjunto de alta dimensionalidad es una tarea demandante de esfuerzo y tiempo.

Ante este escenario, debido al gran éxito de los modelos de lenguaje pre-entrenados en diferentes tareas de procesamiento de lenguaje natural (Floridi & Chiriatti, 2020; Acheampong et al., 2021; Touvron et al., 2023a), ha surgido un creciente interés en el desarrollo de modelos pre-entrenados en el ámbito de la bioquímica-informática (Fabian et al., 2020; Irwin et al., 2021; Lin et al., 2023). Esto, con el fin de obtener características útiles a partir de grandes bases de datos públicamente disponibles que están compuestas por millones de moléculas de pequeño y mediano tamaño (Kim et al., 2018), así como por millones de secuencias de proteínas (Dogan, 2018; Mistry et al., 2020) abarcando diversidad evolutiva (Yanofsky et al., 1964; Altschuh et al., 1987, 1988; Hughes & Yeager, 1997). Para obtener características útiles a partir de esa gran cantidad de datos, modelos auto-supervisados basados en *Transformers* (Devlin et al., 2019; Radford & Narasimhan, 2018; Lewis et al., 2019) han demostrado ser los más adecuados. Tal es el caso de los modelos Evolutionary Scale Modeling (ESM) (Rives et al., 2019; Lin et al., 2023), los cuales generan representaciones (características) de menor dimensión conocidas como incrustaciones (embeddings) sin necesidad de realizar un proceso de selección de características. Recientemente se pre-entrenó una nueva familia de modelos ESM, denominada ESM-2 (Lin et al., 2023), a partir de aproximadamente 65 millones de secuencias polipeptídicas. ESM-2 fue pre-entrenado en escalas que varían desde 8 millones de parámetros hasta 15 mil millones de parámetros. Por lo tanto, se pueden derivar incrustaciones de diferentes dimensiones a partir de estos modelos.

1.2. Planteamiento del problema

Es bien conocido que aumentar la capacidad de los modelos pre-entrenados conduce a obtener mejores representaciones aprendidas (características) (Touvron et al., 2023b). En este sentido, se puede observar en la Tabla S1 del estudio de Lin et al. (2023) que el modelo ESM-2 de 8 millones de parámetros (incrustación de 320 dimensiones) es el que tiene el peor rendimiento en tareas de predicción de estructuras terciarias de proteínas, mientras que el mejor resultado corresponde al modelo ESM-2 de 15 mil millones de parámetros (incrustación de 5120 dimensiones). Sin embargo, en esa misma tabla se puede observar que los modelos ESM-2 de 3 mil millones de parámetros (incrustación de 2560 dimensiones) y de 15 mil millones de parámetros lograron desempeños bastante similares. Estos resultados sugieren estudiar si cuanto mayor sea la capacidad del modelo ESM-2 siempre se lograrán resultados notablemente mejores en tareas más especificas (downstream tasks) como el desarrollo de modelos QSAR para predecir AMPs. Esto es de vital importancia porque las tareas específicas donde se aplican modelos como ESM-2 siempre tienen menor número de datos en comparación al volumen de datos que se usó para pre-entrenar esos modelos. Por lo tanto, si se obtienen buenos resultados tanto desde las incrustaciones más pequeñas como desde las más grandes, entonces se evitan problemas relacionados con el curso de la dimensionalidad. Esto también sugiere analizar si las incrustaciones derivadas de los modelos ESM-2 de mayor capacidad contienen información codificada en las incrustaciones derivadas de los modelos de menor capacidad, o si por el contrario, la información codificada en las diferentes incrustaciones es complementaria y puede fusionarse para mejorar el rendimiento de los modelos QSAR para predecir AMPs.

Además de la calidad de las características de entrada, es necesario explorar varias técnicas de fusión de características y diferentes algoritmos de clasificación para construir buenos modelos QSAR basados en aprendizaje tradicional. La implementación de estas tareas depende en gran medida de la experiencia del modelador (QSAR practitioners). En en el grupo de investigación donde se desarrolló el presente trabajo se tiene una amplia experiencia en la generación de metodologías mediante la fusión de selectores de características y fusión de clasificadores (García-Jacas et al., 2022a,b; Pinacho-Castellanos et al., 2021; Beltran et al., 2020). Sin embargo, realizar todas estas tareas manualmente es un proceso repetitivo y engorroso debido a la cantidad de estrategias de modelación que pueden generarse en esta exploración. Existen herramientas basadas en flujos de trabajo en las que se pueden implementar diferentes estrategias de modelación y modificarlas sin necesidad de tener amplios conocimientos de programación. Una de estas herramientas es KNIME (Konstanz Information Miner, disponible en `https://www.knime.com/`), que es una plataforma de software de código abierto que proporciona una interfaz gráfica de usuario para

crear flujos de trabajo. Usando KNIME se puede crear un flujo de trabajo para el proceso de modelación QSAR, incorporando diferentes algoritmos y técnicas implementadas en varios lenguajes de programación. De esta manera, se evita realizar de manera manual todo el proceso cada vez que se va a modelar un *endpoint*. Además, permite tener un mismo flujo de modelación para poder comparar resultados de una forma justa y así evitar sesgos de modelación, lo cual constituye actualmente un problema en varios estudios comparativos.

1.3. Objetivos

Tomando en consideración todo lo previamente expuesto, se tiene el siguiente el objetivo general y los siguientes objetivos específicos del presente trabajo de tesis.

1.3.1. Objetivo general

Desarrollar un flujo de trabajo en la plataforma KNIME, mediante el uso de técnicas de aprendizaje tradicional, que permita evaluar el impacto de modelos ESM-2 de diferentes capacidades en la construcción de modelos QSAR para la predicción de AMPs.

1.3.2. Objetivos especificos

- Crear un flujo de trabajo en la plataforma Konstanz Information Miner (KNIME) para automatizar el proceso de ingeniería de características y construcción de clasificadores.
- Utilizar el flujo de trabajo desarrollado sobre conjuntos de características extraídas de modelos ESM-2 de diferentes capacidades con el fin de evaluar su contribución en la clasificación de péptidos antimicrobianos.
- Comparar el rendimiento de los modelos creados en este trabajo con respecto a modelos reportados en la literatura.

1.4. Aportes

El presente trabajo tiene como **aporte metodológico** un flujo de trabajo implementado en KNIME que puede ser utilizado no solo en la clasificación de péptidos antimicrobianos, sino también en cualquier tarea de clasificación binaria que se pueda desarrollar tanto en el Departamento de Ciencias de la Computación como en el CICESE en general, así como por la comunidad científica internacional. Además, este trabajo cuenta como **aporte práctico** la creación de modelos predictivos basados en incrustaciones ESM-2 y basados en la fusión de características obtenidas a partir de diferentes incrustaciones ESM-2, los cuales estarán libremente disponibles en un software creado en el departamento.

1.5. Organización de la tesis

La organización del presente trabajo de tesis es la siguiente: el Capítulo 2 aborda los aspectos teóricos y metodológicos del trabajo. En el Capítulo 3 se explica cómo se implementó el flujo de trabajo en la plataforma KNIME. En el Capítulo 4 se presentan los resultados de la aplicación del flujo de trabajo en la plataforma KNIME utilizando conjuntos de datos compuestos por características extraídas de los modelos ESM-2. Así como los resultados generados a partir de la fusión de características obtenidas a partir de diferentes incrustaciones ESM-2. Además se comparan estos resultados obtenidos con los de modelos basados en aprendizaje profundo reportados en la literatura. Por último, en el Capítulo 5, se exponen las conclusiones y el trabajo futuro.

Capítulo 2. Fundamento teórico

En este capítulo se presentan los fundamentos teóricos que sustentan el presente trabajo de tesis. Se abordan los conceptos básicos sobre aminoácidos y su conexión en la formación de péptidos y proteínas. Luego, se habla de la representación de los péptidos utilizada. Se explica qué es un modelo de Relación Cuantitativa Estructura-Actividad (QSAR), y se describen las diferentes etapas involucradas en su construcción, así como las herramientas empleadas en cada una de estas fases que se usaron para implementar el flujo de trabajo en la plataforma KNIME. Finalmente, se aborda la evaluación de estos modelos para determinar su capacidad de predecir con precisión la actividad de interés.

2.1. Péptidos antimicrobianos

2.1.1. Aminoácidos

Las células usan aminoácidos para construir péptidos y proteínas que se unen en una larga cadena que se pliega en una estructura tridimensional única para cada tipo de péptido y proteína (Alberts et al., 2015). Los aminoácidos son pequeñas moléculas compuestas por un grupo amino ($-NH_2$), un grupo carboxilo ($-COOH$), un átomo de carbono central (conocido como $carbono\ \alpha$), un átomo de hidrógeno (H), y una cadena lateral (R) que varía según el tipo de aminoácido (ver Figura 1 a). La identidad de esta cadena lateral es lo que distingue a un aminoácido de otro (Alberts et al., 2015). En la Tabla 1 se muestran los 20 aminoácidos más comunes y predominantes que pueden ser encontrados en péptidos y proteínas. Éstos aminoácidos se clasifican según la naturaleza química de la cadena lateral que puede ser ácido, básico, polar sin carga, o no polar (Alberts et al., 2015).

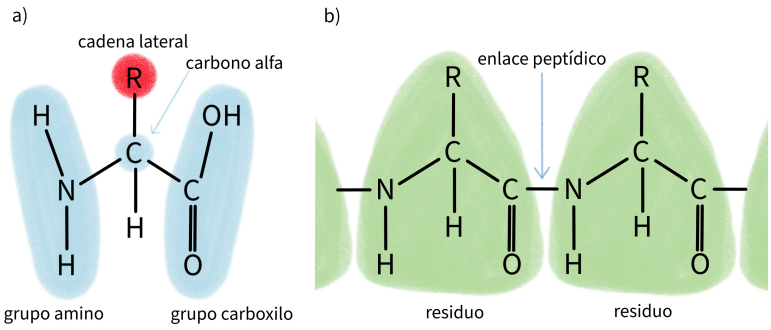

Figura 1. Formación de péptidos y proteínas por la unión de aminoácidos mediante enlaces peptídicos. a) Diagrama esquemático de un aminoácido. b) Enlace peptídico. Esta figura está basada en Figura 1.2 de Branden & Tooze (2012).

Los aminoácidos se unen por medio de enlaces peptídicos. Este enlace se forma cuando el grupo carboxilo de un aminoácido se condensa con el grupo amino del siguiente, en este proceso se elimina una molécula de agua (ver Figura 1 b) (Branden & Tooze, 2012). Una vez que los aminoácidos están unidos se denominan residuos de aminoácidos (Voet et al., 2016). La formación de una sucesión de enlaces peptídicos genera una cadena principal o backbone, desde la cual se proyectan las distintas cadenas laterales (Branden & Tooze, 2012).

Tabla 1. Aminoácidos estándar que pueden ser encontrados en péptidos y proteínas. Tabla basada en panel 2-5 de Alberts et al. (2015).

Aminoácido	Abreviatura	Letra	Tipo
Arginina	Arg	R	Básico
Lisina	Lys	K	
Histidina	His	H	
Ácido aspártico	Asp	D	Ácido
Ácido glutámico	Glu	E	
Asparagina	Asn	N	Polar no cargado
Glutamina	Gln	Q	
Serina	Ser	S	
Treonina	Thr	T	
Tirosina	Try	Y	
Alanina	Ala	A	No polar
Valina	Val	V	
Leucina	Leu	L	
Isoleucina	Ile	I	
Prolina	Pro	P	
Fenilalanina	Phe	F	
Metionina	Met	M	
Triptófano	Trp	W	
Glicina	Gly	G	
Cisteína	Cys	C	

2.1.2. Péptidos antimicrobianos

Los péptidos, como las proteínas, son macromoléculas compuestas por la unión de aminoácidos mediante enlaces peptídicos (Sewald & Jakubke, 2015). Un péptido tiene un menor número de aminoácidos que una proteína, sin embargo, el umbral de aminoácidos para distinguir uno de otro no está bien definido (Sewald & Jakubke, 2015). En el presente documento consideramos que la cantidad de aminoácidos de un péptido puede variar desde 2 hasta 100.

Los péptidos antimicrobianos (AMPs por sus siglas en inglés), también conocidos como péptidos de defensa del huesped o *host defense peptides*, son parte esencial de la inmunidad innata en la mayoría de los organismos vivos. Se encuentra prácticamente en todas las formas de vida, desde microorganismos hasta seres humanos (Divyashree et al., 2020). Los AMPs tienen una amplia gama de efectos inhibitorios contra bacterias, hongos, parásitos y virus (Huan et al., 2020), además de otras actividades biológicas como anticancerígenas, inmunomoduladoras, entre otras (Divyashree et al., 2020).

Zhang et al. (2021) divide a los AMPs en cinco subgrupos de acuerdo a la carga neta, la estructura y la fuente. El primer subgrupo son AMPs aniónicos que tienen carga neta de -1 a -8. La mayoría de los AMPs aniónicos son fragmentos peptídicos después de la proteólisis, y algunos AMP aniónicos son moléculas pequeñas codificadas por genes. Parecen utilizar iones metálicos y los componentes cargados negativamente de la membrana microbiana para formar puentes salinos, interactuando así con los microbios. El segundo subgrupo son AMP catiónicos con estructura α-hélice. Estos AMPs tiene una carga neta de $+2$ a $+9$. Además, estos AMP suelen contener más del 50 % de aminoácidos hidrofóbicos, lo que permite la formación de estructuras anfipáticas al interactuar con las células objetivo. El tercer subgrupo son AMPs catiónicos con estructura β-hoja. Los péptidos suelen contener de 2 a 8 residuos de cisteína que forman de 1 a 4 pares de enlaces disulfuro intramoleculares. El cuarto subgrupo son los AMPs catiónicos con estructura extendida que contienen aminoácidos específicos que incluyen arginina, prolina, triptófano, glicina e histidina, pero carecen de estructuras secundarias regulares. Sus estructuras se estabilizan únicamente mediante enlaces de hidrógeno y la fuerza de interacción de van der Waals con los lípidos de la membrana. Finalmente, el quinto subgrupo lo componen los fragmentos de proteínas antimicrobianas.

Los AMPs exhiben múltiples mecanismos de acción para proteger al huésped de patógenos invasores. El modo de acción se puede dividir en dos clases principales: muerte directa y modulación inmune. El mecanismo de acción de destrucción directa se puede dividir además en dirigido a la membrana y no dirigido a la membrana (Kumar et al., 2018). Como explica Zhang et al. (2021), el mecanismo directo de los AMPs dirigido a la membrana se realiza a través de la interacción con membranas cargadas negativamente, lo que da como resultado un aumento de la permeabilidad de la membrana, la lisis de la membrana celular y la liberación de contenidos intracelulares, lo que finalmente conduce a la muerte celular. Hay cuatro modelos principales de formación de poros de membrana: modelo duela de barril (barrel-stave), modelo de poro toroidal (toroidal-pore), modelo de alfombra (carpet) y modelo agregado (aggregate) (Zhang et al., 2021, ver Figura 1). Después de que los AMP penetran en la membrana, sus regiones hidrofóbicas se combinan con las regiones hidrofóbicas internas de la bicapa de fosfolípidos,

mientras que sus regiones hidrofílicas quedan expuestas al exterior. Además de la destrucción de la membrana, también se han informado otros modos de acción por ejemplo al actuar sobre proteínas de la membrana. El mecanismo directo de los AMPs no dirigido a la membrana se da cuando los AMPs penetran en el citoplasma e interactúan con sustancias intracelulares y pueden, por ejemplo, inhibir la síntesis de ADN, ARN y proteínas, inhibir el plegamiento de proteínas, inhibir la actividad enzimática y la síntesis de la pared celular o promover la liberación de liasas para destruir la pared celular (Zhang et al., 2021).

Según Zhang et al. (2021), la actividad de los AMPs está influenciada por varios factores, entre ellos la longitud del péptido, la carga neta, la hidrofobicidad y la estructura secundaria. La longitud del péptido afecta la actividad antimicrobiana porque los péptidos deben atravesar la bicapa lipídica para estabilizar el poro. Sin embargo, al cambiar la longitud del péptido, también se modifican la carga neta y la hidrofobicidad. Un aumento en la carga positiva de los AMPs resulta en una mayor unión del péptido a las membranas bacterianas aniónicas. No obstante, el efecto biológico de los péptidos altamente cargados se reduce significativamente en alta fuerza iónica y aumenta su actividad hemolítica (Kumar et al., 2018). En presencia de grupos hidrofóbicos, las cadenas de péptidos pueden formar polímeros en solución que les permiten insertarse en el núcleo hidrofóbico de la membrana. Los residuos hidrofóbicos también aumentan la capacidad de los AMPs para formar estructuras en α-hélice y mejorar su estabilidad. Sin embargo, los niveles excesivos de hidrofobicidad pueden conducir a la toxicidad de las células de los mamíferos (Kumar et al., 2018). Cuando los AMPs adoptan una estructura secundaria específica, muestran una clara anfipaticidad, la cual es una base estructural importante para su actividad antimicrobiana. Sin embargo, una alta anfipaticidad disminuye la actividad antibacteriana de los AMPs y aumenta su actividad hemolítica. Por estas razones, no existe una solución estándar para optimizar el diseño de los AMPs al coordinar simultáneamente estos diversos factores (Zhang et al., 2021).

2.2. Descriptores para péptidos y proteínas

El descriptor molecular es el resultado final de un procedimiento matemático que transforma la información química codificada dentro de una representación simbólica de una molécula en un número útil (Consonni & Todeschini, 2009). En este trabajo de tesis se utilizaron descriptores que extraen información de la secuencia polipeptídica y la convierten en un valor numérico. Estos descriptores son comúnmente utilizados en la generación de modelos matemáticos para predecir propiedades y funciones (Marrero-Ponce

et al., 2020). Existen dos tipos principales de descriptores (también conocidos como características). Por un lado, están las características calculadas (handcrafted) que se obtienen mediante algoritmos o fórmulas, mientras que, por otro lado, están las características aprendidas (non-handcrafted) que se determinan intrínsecamente a partir de modelos basados en arquitecturas de redes neuronales profundas (García-Jacas et al., 2022b).

En este trabajo de tesis se utilizaron los modelos preentrenados ESM-2. Estos modelos generan representaciones numéricas conocidas como incrustaciones (embeddings), las cuales capturan características de las secuencias de proteínas. Dichas características son aprendidas por los modelos sin intervención humana. Para crear esta familia de modelos utilizaron técnicas de aprendizaje automático auto-supervisado basadas en transformadores (*Transformers*). ESM-2 fue entrenado con aproximadamente 65 millones de secuencias sin etiqueta, abarcando una amplia diversidad evolutiva (Lin et al., 2023). La familia de modelos ESM-2 incluye seis modelos preentrenados con 6, 12, 30, 33, 36 y 48 capas, que escalan a 8 millones, 35 millones, 150 millones, 650 millones, 3 mil millones y 15 mil millones de parámetros, respectivamente. Por lo tanto, al utilizar la familia de modelos ESM-2 es posible obtener incrustaciones de 320, 480, 640, 1280, 2560, 5120 dimensiones, respectivamente (Lin et al., 2023). Para obtener los descriptores de un conjunto de secuencias de proteínas se utiliza el script proporcionado por los autores y que está disponible en `https://github.com/facebookresearch/esm`. Mediante la interfaz en línea de comandos (esm-extract) se puede extraer de manera eficiente incrustaciones para un archivo FASTA:

```
esm-extract model_location fasta_file output_dir
[-h] [--toks_per_batch TOKS_PER_BATCH]
[--repr_layers REPR_LAYERS [REPR_LAYERS ...]]
--include {mean,per_tok,bos,contacts} [{mean,per_tok,bos,contacts} ...]
[--truncation_seq_length TRUNCATION_SEQ_LENGTH]
```

Donde:

- `model_location`: Nombre del modelo preentrenado para descargar.
- `fasta_file`: Fichero FASTA sobre el que extraer representaciones.
- `output_dir`: Directorio de salida para representaciones extraídas.

Argumentos opcionales:

- `-h, --help`: Mostrar este mensaje de ayuda y salir.
- `--toks_per_batch TOKS_PER_BATCH`: Tamaño máximo de lote.
- `--repr_layers REPR_LAYERS [REPR_LAYERS ...]`: capas de los que extraer representaciones.
- `--include {mean,per_tok} [{mean,per_tok} ...]`: Especificar qué representaciones guardar, por ejemplo, `mean` incluye los incrustaciones promedio de la secuencia completa por capa y `per_tok` incluye la secuencia completa con una incrustación por aminoácido ($seq\_len * hidden\_dim$).
- `--truncation_seq_length TRUNCATION_SEQ_LENGTH`: Truncar secuencias más largas que cierto valor.

Por ejemplo, el siguiente comando lee un archivo FASTA y usa el modelo ESM-2 de 33 capas para extraer la incrustación promedio (`--include mean`) correspondiente a la incrustación por aminoácido de la capa final (`--repr_layers 33`). Esta incrustación promedio es la salida para cada péptido, que es un vector con dimensión $1x1280$ que contiene las características de la secuencia de un péptido.

```
esm-extract esm2_t33_650M_UR50D examples/data/some_proteins.fasta examples/data/
    ↪ some_proteins_emb_esm2 --repr_layers 33 --include mean
```

2.3. Modelación QSAR

Como explica Consonni & Todeschini (2009), los modelos QSAR (Relaciones Cuantitativas Estructura-Actividad, por sus siglas en inglés) son enfoques usados para predecir y comprender la relación entre propiedades moleculares de un compuesto y la diana farmacológica (o *endpoint*), que puede ser, su actividad biológica, físico-química o toxicológicas. En el proceso de desarrollo de un modelo QSAR se identifican patrones y correlaciones entre las características moleculares y el *endpoint*, haciendo uso de análisis estadístico, matemático o de técnicas de aprendizaje automático. Una vez que se ha desarrollado un modelo QSAR válido se predice el *endpoint* en compuestos desconocidos. Esto permite evaluar automáticamente una gran cantidad de compuestos ya sea para realizar inferencias, hipótesis y predicciones, o bien como apoyo antes de una evaluación biológica en el laboratorio (wet lab) (Beltran et al., 2020). En la presente sección se mencionan algunas técnicas y herramientas útiles en el proceso de desarrollo

de un modelo QSAR basados en aprendizaje automático para un problema de clasificación binaria, es decir, donde el *endpoint* es una variable que puede tener dos valores posibles, por ejemplo, "AMP" y "no-AMP".

2.3.1. Limpieza de datos

Una vez que se tiene un conjunto de datos, es decir, una lista de instancias representadas con su vector de características (descriptores moleculares) y su etiqueta correspondiente al *endpoint*, la limpieza de datos es un preprocesamiento crucial en el desarrollo de modelos de aprendizaje automático ya que son muy sensibles a la calidad de los datos de entrada, lo que puede afectar negativamente el rendimiento y la precisión del modelo (Ilyas & Chu, 2019). En esta etapa se eliminan datos irrelevantes, redundantes o incompletos. A continuación se muestran las técnicas utilizadas en este trabajo de tesis:

2.3.1.1. Imputación de datos

Cuando hay valores faltantes (missing values) en el conjunto de datos existen varias técnicas para tratarlos. Puede ser eliminando directamente las filas (instancias) o columnas (características) con valores faltantes, hay casos en los que esto implica una reducción significativa de el conjunto de datos, por lo que en lugar de eliminarlos se pueden reemplazar los valores faltantes con valores fijos o estimados utilizando técnicas como el promedio, la mediana, regresión, por vecinos más cercanos, entre otros. Es común en el área de bioinformática reemplazar los valores faltantes con cero (García-Jacas et al., 2022b; Pinacho-Castellanos et al., 2021).

2.3.1.2. Eliminar características cercanas a constantes

Las características cercanas a constantes son aquellas columnas en el conjunto de datos que tienen el mismo valor para todas o casi todas las instancias, lo que significa que no aportan información útil para crear un modelo. Eliminarlas puede ser beneficioso, por ejemplo, para reducir el costo computacional,

mejorar el rendimiento y simplificar el modelo resultante. Uno de los análisis que se pueden hacer para identificar estas características es usando la curtosis, una medida estadística que cuantifica en qué medida la distribución de probabilidad de una variable se parece a la forma de una distribución normal. La curtosis de una característica x se calcula de la siguiente manera (Kokoska & Zwillinger, 2000):

$$\text{Curtosis(x)} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^4/n}{\sigma^4} \quad (1)$$

Donde x_i es el valor de la característica x para la i-ésima instancia; $\bar{x}$ y σ son el promedio y la desviación estándar de los valores de la característica x para todas las instancias, respectivamente; y n es el número de instancias.

Comúnmente se utiliza el exceso de curtosis, que es la curtosis menos tres, para identificar el tipo de distribución en las categorías: Leptocúrtica cuando el exceso de curtosis es mayor a cero (la distribución tiene colas más pesadas y es más puntiaguda que una distribución normal), Mesocúrtica cuando el exceso de curtosis es igual a cero (la distribución es similar a la distribución normal), y Platicúrtica cuando el exceso de curtosis es menor a cero (la distribución tiene colas más ligeras y es más plana que una distribución normal). El exceso de curtosis toma valores entre -4 e infinito. Una variable con un alto valor de exceso de curtosis es constante o casi constante. Suele eliminarse las características con valores de exceso de curtosis mayores a 5. En el presente trabajo de tesis usamos un umbral de 11. A este tipo de filtros se les conoce como filtro no supervisado ya que en ningún momento se considera la variable *endpoint*.

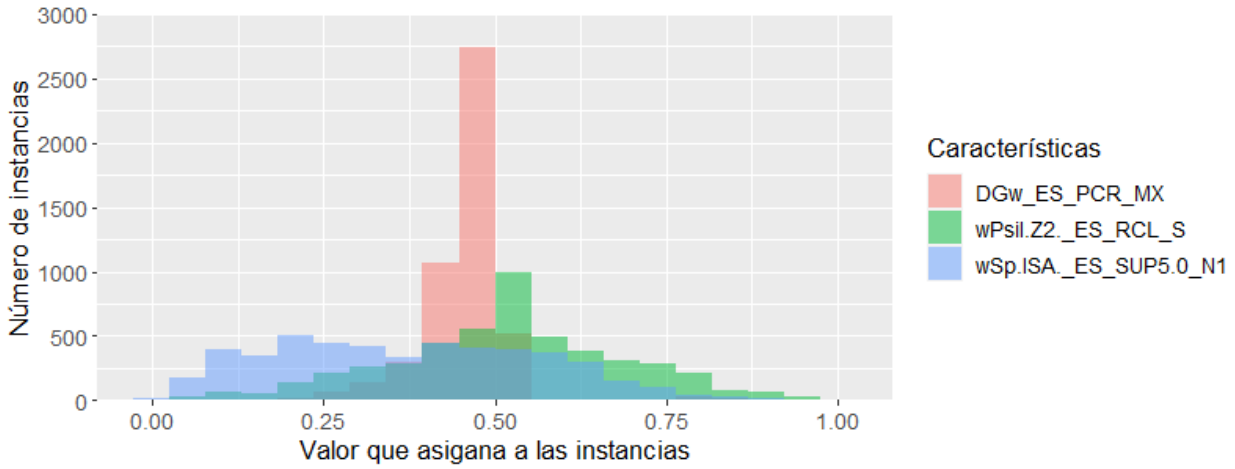

Figura 2. Ejemplo de la distribución de valores normalizados que las siguientes características le asignan a un conjunto de 4908 péptidos: DGw_ES_PCR_MX mide el efecto hidrofóbico originado por el aumento de la entropía de las moléculas de agua cuando una proteína se pliega. wPsiI(Z2)_ES_RCL_S mide la clasificación de un residuo como tipo irregular según su ángulo de torsión Psi. wSp(ISA)_ES_SUP5.0_N1 mide la accesibilidad de los residuos según un límite en el área superficial accesible establecido. Estas características se calcularon con la herramienta ProtDCal (Romero-Molina et al., 2019).

En la Figura 2 se compara la distribución de los valores que tres características le asignan a un conjunto de 4908 péptidos. DGw_ES_PCR_MX tiene un exceso de curtosis de 13.4 por lo que tiene una distribución leptocúrtica. wPsiI(Z2)_ES_RCL_S tiene un exceso de curtosis de 0.003 por lo que se puede decir que tiene una distribución mesocúrtica. wSp(ISA)_ES_SUP5.0_N1 tiene un exceso de curtosis de -0.73 por lo que tiene una distribución platicúrtica.

2.3.1.3. Eliminar características irrelevantes

Otro filtro no supervisado para eliminar características irrelevantes es mediante la Entropía de Shannon (SE por sus siglas en inglés) que representa la cantidad de información codificada en una variable. Si cada péptido se ubica en un intervalo según el valor asignado por una característica, entonces la entropía será mayor en la medida en que un mayor número de intervalos sean poblados. Por el contrario, si todos los péptidos son ubicados en pocos intervalos, entonces la entropía será menor.

Para calcular la SE de una característica desde un enfoque no supervisado, se utilizan técnicas de discretización (binning) (Urias et al., 2015) para agrupar los valores en intervalos específicos siguiendo los siguientes pasos que se encuentran implementamos en Fragmento 1 en los anexos.

1. Se ordenan los valores de la característica de menor a mayor.
2. Se calcula el valor mínimo (min) y el valor máximo (max).
3. Se calcula el ancho de cada intervalo (bin_width) dividiendo la diferencia entre max y min por el total de instancias (n). Esto permite crear n intervalos aumentando desde min en bin_width hasta max.
4. Se obtienen las frecuencias de cada intervalo.
5. Se calculan la probabilidad de ocurrencia para cada intervalo dividiendo su frecuencia por n.
6. Utilizando estas probabilidades, se calcula la Entropía de la siguiente manera (Shannon, 1948):

$$SE = -\sum_{i=1}^{n} p_i \log_2 p_i \tag{2}$$

Una vez calculada la Entropía de Shannon a cada característica, el filtro consiste en conservar las características con valores de SE superiores al 25 % de la SE máxima que una característica puede

alcanzar. El esquema de discretización utilizado es igual al número de instancias en el conjunto de datos. Por lo tanto, el SE máximo es igual a $\log_2 n$. En la Figura 2 se observa que la característica DGw_ES_PCR_MX asigna a la mayoría de los péptidos en pocos intervalos y tiene un valor de entropía de 8.23. En contraste, las características wPsiI(Z2)_ES_RCL_S y wSp(ISA)_ES_SUP5.0_N1 tienen un valor de entropía de 9.08 y 11.15, respectivamente. Dado que la entropía máxima es $\log_2 4908 = 12.26$, estas tres características pasan el filtro del 25 % de la entropía máxima $(12.26 * 0.25 = 3.06)$.

2.3.1.4. Eliminar características redundantes

En el ámbito del aprendizaje automático, las características redundantes son aquellas que no agregan información adicional o única al modelo. Estas características pueden introducir ruido innecesario y aumentar la complejidad del modelo sin mejorar su capacidad predictiva. Una de las métricas para cuantificar la redundancia entre cada par de características es el coeficiente de correlación de Spearman. Este coeficiente mide la correlación entre los rangos de las características, en lugar de utilizar sus valores reales (Kokoska & Zwillinger, 2000). En otras palabras, para el par de características X_1 y X_2, se utilizan sus respectivos rangos $R(X_1)$ y $R(X_2)$ para calcular el coeficiente de correlación de Spearman utilizando la siguiente fórmula:

$$\rho_S = \frac{COV(R(X_1), R(X_2))}{\sigma_{R(X_1)} * \sigma_{R(X_2)}} = \frac{n \sum_{i=1}^{n} r_{1i} r_{2i} - \left(\sum_{i=1}^{n} r_{1i}\right)\left(\sum_{i=1}^{n} r_{2i}\right)}{\sqrt{\left[n \sum_{i=1}^{n} r_{1i}^2 - \left(\sum_{i=1}^{n} r_{1i}\right)^2\right]\left[n \sum_{i=1}^{n} r_{2i}^2 - \left(\sum_{i=1}^{n} r_{2i}\right)^2\right]}} \quad (3)$$

El coeficiente de correlación de Spearman oscila entre -1 y $+1$, indicando asociaciones negativas o positivas, respectivamente. Un valor de cero significa que no hay correlación entre las características. Cuando un par de características tienen un alto coeficiente de correlación de Spearman superior a un umbral establecido, entonces una de las características será eliminada. El umbral utilizado es 0.95.

2.3.1.5. Normalización

La normalización de características, también conocida como escalamiento, es un paso útlil en el preprocesamiento de datos en el campo del aprendizaje automático, ya que, puede ayudar a evitar sesgos en

algunos algoritmos de aprendizaje, acelerar la convergencia o facilitar la interpretación de los pesos o coeficientes del modelo final. La normalización consiste en transformar los valores de las características para que se encuentren dentro de un intervalo específico. Una de las técnicas utilizadas es mediante el "escalamiento min-max" que lleva los valores de una característica a un intervalo entre 0 y 1. Su fórmula es:

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{4}$$

Donde X es el valor original de la característica; X_{min} y X_{max} son los valores mínimos y máximos de la característica, respectivamente; y X_{norm} es el valor normalizado resultante.

2.3.2. Selección de características

Cuando se tiene una gran cantidad de características, los modelos tienden a sobreajustarse, lo que puede causar una degradación del rendimiento en instancias nuevas. Además, aumenta los requisitos de almacenamiento de memoria y los costos computacionales (Li et al., 2017). La selección de características es una manera de reducir la dimensionalidad de los datos (es decir, reducir la cantidad de características) y, si se realiza correctamente, puede solucionar los problemas mencionados anteriormente.

En el caso de problemas de clasificación, se suele realizar una selección de características supervisada. Su objetivo es identificar y seleccionar las características más relevantes y útiles que puedan discriminar instancias de diferentes clases. Por ejemplo, en la Figura 3 se muestran tres características que se calcularon para 4908 péptidos que tienen una etiqueta de clase "AVP" o "no-AVP". Se puede observar que las características DGw_ES_PCR_MX y wPsiI(Z2)_ES_RCL_S tienden a asignarle los mismos valores tanto a instancias de la clase "AVP" como a instancias de la clase "no-AVP". Esto no ayuda a distinguir o predecir la clase de una instancia según su valor en la característica. En cambio, la característica wSp(ISA)_ES_SUP5.0_N1 puede discriminar mejor las dos clases. Por ejemplo, si una nueva instancia tiene un valor en esta característica entre 0 y 0.45, podríamos predecir que su clase es "AVP", mientras que si tiene un valor entre 0.45 y 1, podríamos predecir que su clase es "no-AVP".

Como explican Bolón-Canedo & Alonso-Betanzos (2019), los selectores de características se pueden clasificar en tres categorías principales: métodos de filtro (tipo ranking), métodos de envoltura (wrapper) y métodos integrados (embedded). Los métodos de filtro asignan un orden a las características según su relevancia con respecto al *endpoint* (la clase). En este caso, es necesario establecer un umbral para

poder decidir la cantidad de características a conservar. Los métodos de envoltura, en cambio, utilizan el rendimiento de un clasificador para evaluar qué tan bueno es un subconjunto de características. Por último, los métodos integrados realizan el proceso de selección de características durante el entrenamiento de un clasificador dado.

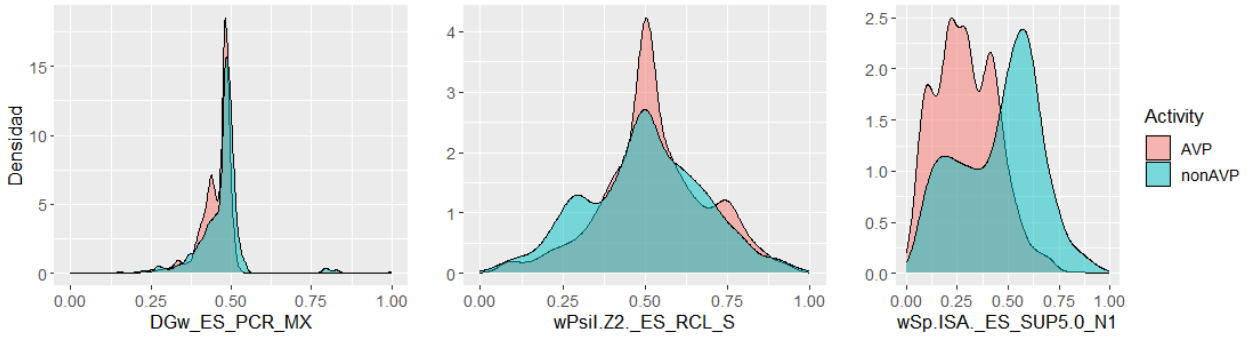

Figura 3. Ejemplo de densidades de los valores normalizados que las características DGw_ES_PCR_MX, wPsiI(Z2)_ES_RCL_S y wSp(ISA)_ES_SUP5.0_N1 le asignan a un conjunto de 4908 péptidos distinguiéndolos de acuerdo a su etiqueta de clase "AVP" o "no-AVP".

2.3.2.1. Métodos de filtro

Los métodos de filtro (o de tipo ranking) asignan un orden a las características según su relevancia con respecto al *endpoint* (la clase). Existen muchos criterios para medir esta relevancia, lo que da lugar a diferentes métodos de filtro. A continuación se describen los usados en este trabajo.

- **Relief-F**

  El Relief-F es un método basado en similitud que evalúa qué tan similares son los valores de la característica X para instancias cercanas (vecinas) que pertenecen a la misma clase y, al mismo tiempo, qué tan distintos son los valores para instancias cercanas (vecinas) que pertenecen a clases diferentes. Se basa en la suposición de que una característica "buena" debe tener el mismo valor para instancias de la misma clase y valores diferentes para instancias de diferentes clases (Pes, 2019). La implementación de ReliefF del repositorio scikit-feature en Python (Li et al., 2017), obtiene la puntuación de la característica X partiendo de una extracción aleatoria de l instancias de entre todas las n instancias, para así calcular:

$$\text{ReliefF}(X) = \frac{1}{c}\sum_{i=1}^{l}\left(-\frac{1}{m_i}\sum_{x_r \in NH(i)} d(x_i, x_r) + \sum_{y \neq y_i}\frac{1}{h_{iy}}\frac{p(y)}{1-p(y)}\sum_{x_r \in NM(i,y)} d(x_i, x_r)\right) \quad (5)$$

Donde $NH(i)$ y $NM(i,y)$ son las instancias más cercanas a x_i en la misma clase y en la clase y, respectivamente. Cuyos tamaños son m_i y h_{iy}, respectivamente. $d(.,.)$ es una función de distancia. $p(y)$ es la proporción de instancias en la clase y. Entre mayor sea el valor de reliefF, mejor característica es.

- **T-Score**

  El T-score es un método basado en estadística usado en problemas de clasificación binaria. Se basa en la idea de que una "buena" característica tiene las medias de dos clases alejadas y poca varianza dentro de cada clase. La implementación del t-score del repositorio scikit-feature en Python (Li et al., 2017), obtiene el t-score para la característica X con la siguiente fórmula:

$$Tscore(X) = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1}{n_1} + \frac{\sigma_2}{n_2}}} \tag{6}$$

  Donde μ_1 y μ_2 es la media de las características de las instancias de la clase 1 y 2, respectivamente; σ_1 y σ_2 son las desviaciones estándar correspondientes; y n_1 y n_2 indican el número de instancias de estas dos clases. Cuanto mayor sea el valor del t-score, más importante es la característica.

- **Índice de Gini**

  El índice de Gini (Gini-index) es un método basado en estadística. En su fórmula aparecen dos términos que miden la impureza de dos conjuntos de datos separados en función de los valores de una característica. Si esta impureza es alta, indica que las clases están mezcladas en el conjunto, mientras que si es baja indica que el conjunto está compuesto principalmente por una única clase. Para la clasificación binaria, el índice de Gini puede tomar un valor máximo de 0,5. Cuanto más bajo es el valor del índice de Gini, más relevante es la característica en la clasificación, ya que indica que la división según esa característica resulta en conjuntos más puros y distintos en términos de las clases que contienen.

  La implementación de Gini-index del repositorio scikit-feature en Python (Li et al., 2017) obtiene la puntuación del índice de Gini para la característica X de la siguiente manera: si X tiene r valores de característica diferentes, para cada i-ésimo valor de característica se separa el conjunto de datos en W_i y $\bar{W}_i$ que denotan el conjunto de instancias con el valor menor o igual al i-ésimo valor de característica y mayor que el i-ésimo valor de característica, respectivamente. Luego, se utiliza la siguiente fórmula:

$$gini\_index(X) = \min_i \left(p(W_i)(1 - \sum_{s=1}^{c} p(C_s|W_i)^2) + p(\bar{W}_i)(1 - \sum_{s=1}^{c} p(C_s|\bar{W}_i)^2) \right) \tag{7}$$

Donde $p(W_i)$ y $p(\bar{W}_i)$ representan las probabilidades de los conjuntos W_i y $\bar{W}_i$, respectivamente; $p(C_s|W_i)$ y $p(C_s|\bar{W}_i)$ es la probabilidad condicional de la clase s dado W_i y $\bar{W}_i$, respectivamente.

- **ANOVA F**

  El método ANOVA F calcula el el valor F entre cada característica y la clase (*endpoint*). El análisis de varianza se basa en la descomposición de la variabilidad total de los datos en dos componentes: La variabilidad entre clases, es decir, distancia entre las medias de las distribuciones dada la clase. Y la variabilidad dentro de las clases, es decir, la varianza de cada clase individual. Un valor F alto indica mayor variabilidad entre grupos en comparación con la variabilidad dentro de los grupos. Lo cual es deseable en una característica, por lo tanto, entre mayor sea el valor F, más relevante es.

  Para el caso de una característica X con valores continuos y una variable objetivo que determina 2 clases (por ejemplo, "AVP" y "No-AVP"), el estadístico F calcula la relación entre estas dos fuentes de variabilidad de la siguiente manera (Kokoska & Zwillinger, 2000):

$$F = \frac{\sum_{i=1}^{2} n_i(\bar{X}_i - \bar{X})^2}{\frac{1}{N-2}\sum_{i=1}^{2}\sum_{j=1}^{n_i}(X_{ij} - \bar{X}_i)^2} \tag{8}$$

  Donde $\bar{X}_i$ es la media de los valores de la característica X que pertenecen a la clase i; $\bar{X}$ es la media de todos los valores de la característica X; n_i es la cantidad de instancias que pertenecen a la clase i; y X_{ij} es el j-ésimo valor de la característica X que pertenecen a la clase i. Una de las implementaciones de este método es la función `f_classif` que se encuentra en la biblioteca scikit-learn en Python (Buitinck et al., 2013).

- **Chi-Cuadrado**

  El test chi-cuadrado (Liu & Setiono, 1995) se usa en estadística para probar la independencia de dos eventos. Mide cómo se desvían entre sí el conteo esperado y el conteo observado. Cuando la característica es independiente de la clase, el conteo observado está cerca del conteo esperado, por lo que el valor de chi-cuadrado será más pequeño. Cuanto mayor sea el valor de chi-cuadrado, la característica depende más de la clase, por lo que es más relevante para el entrenamiento del modelo (Pes, 2019). Dada una característica particular X con r valores diferentes, la puntuación de Chi-cuadrado de esa característica se puede calcular como:

$$\chi^2 = \sum_{i=1}^{r}\sum_{j=1}^{c}\frac{(n_{ij} - \mu_{ij})^2}{\mu_{ij}} \tag{9}$$

  Donde n_{ij} es el número de instancias con el j-ésimo valor de característica de X; $\mu_{ij} = \frac{n_{*j}n_{i*}}{n}$ donde n_{i*} es el número de instancias con el i-ésimo valor de característica de X y n_{*j} es el número

de instancias en la clase s. Una de las implementaciones de este método es la función `chi2` que se encuentra en la biblioteca scikit-learn en Python (Buitinck et al., 2013).

- **Información Mutua**

  El método de la información mutua (MI) mide el contenido de información compartido entre una característica y la clase (García et al., 2019). Se define como la reducción en la entropía de una variable debido al conocimiento del valor de la otra, es decir, $MI(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$. Cuando MI es igual a cero, las variables son independientes, y entre mayor sea su valor, mayor dependencia y por lo tanto, más relevante será la característica con respecto a la clase.

  Una de las implementaciones de este método es la función `mutual_info_classif` que se encuentra en la biblioteca scikit-learn en Python (Buitinck et al., 2013). Esta función se basa en la estimación de la entropía a partir de las distancias de k vecinos más cercanos, como se describe en Ross (2014): para cada instancia i, se calcula un número Ii basado en sus k vecinos más cercanos según su valor en la característica X. Primero, se encuentra el k-ésimo vecino más cercano al punto i entre las N_{yi} instancias cuya clase es yi usando alguna métrica de distancia. Se obtiene d como la distancia a este k-ésimo vecino. Luego se cuenta el número de vecinos m_i en el conjunto de datos completo que se encuentran dentro de la distancia d al punto i (incluido el propio k-ésimo vecino). Con base en N_{yi} y m_i se calcula:

$$I_i = \psi(N) - \psi(N_{yi}) + \psi(k) - \psi(m_i) \tag{10}$$

  Donde $\psi(.)$ es la función digamma y k es un valor fijo establecido por el usuario, indica la cantidad de vecinos a considerar. Finalmente, para estimar la Información Mutua de X y Y se promedian los I_i.

- **Selector de Subconjuntos de Características Basado en Correlación**

  El selector de características basado en correlación (CFS, por sus siglas en inglés), es un método que busca un subconjunto de características que estén altamente correlacionadas con la clase y poco correlacionadas entre sí. Para ello utiliza un algoritmo que evalúa subconjuntos de características con la siguiente función heurística (Hall & Smith, 1998):

$$M(S) = \frac{k\bar{r_{cf}}}{\sqrt{k + k(k-1)\bar{r_{ff}}}} \tag{11}$$

  A esta función se le conoce como el mérito del subconjunto de características S que contiene k

características. $\bar{r_{cf}}$ es el promedio de las correlaciones entre la clase y las características. $\bar{r_{ff}}$ es el promedio de las correlaciones entre características.

Para evitar una búsqueda exhaustiva de todos los posibles subconjuntos de características, CFS utiliza estrategias de búsqueda heurística como selección hacia adelante (forward), eliminación hacia atrás (backward) y mejor primero (best first) (Hall & Smith, 1998). Con mejor primero se puede comenzar sin características o con todas las características. En el primer caso, la búsqueda avanza hacia adelante a través del espacio de búsqueda agregando características una a una. Para evitar que explore todo el espacio de búsqueda de subconjuntos de características, se impone un criterio de detención, por ejemplo, la búsqueda se detendrá si cinco subconjuntos expandidos consecutivos no muestran ninguna mejora respecto al mejor subconjunto actual. Una implementación del método CFS se encuentra en Weka (Waikato Environment for Knowledge Analysis) (Witten et al., 1999), el cual se utilizó desde Python a través de la librería python-weka-wrapper3 v0.2.12.

2.3.2.2. Métodos de fusión de características (Ensemble)

Existen varios de métodos de selección de características. Sin embargo, no hay una guía para elegir el método más adecuado para un problema en específico. La selección de características mediante ensembles, que combina la salida de varios selectores de características, puede formar un subconjunto de características más confiable, mejorar el rendimiento y el usuario se libera de tener que elegir un solo método (Bolón-Canedo & Alonso-Betanzos, 2019). La salida de los selectores de características básicos puede ser un subconjunto de características o bien, todas las características con un peso de relevancia asignado. En el primer caso se pueden combinar los subconjuntos seleccionados por diferentes métodos, por ejemplo, mediante la intersección, la unión u otros técnicas más sofisticadas que implican entrenar y evaluar clasificadores. La unión consiste en combinar todas las características que han sido seleccionadas por al menos uno de los selectores de características (Bolón-Canedo & Alonso-Betanzos, 2019). En el segundo caso, la combinación de los resultados se realiza mediante métodos de agregación de los pesos, por ejemplo, asignar el mínimo, la mediana, el promedio (Pes, 2019), entre otros. Por ejemplo, el promedio de los pesos obtenidos por varios selectores de características nos da como resultado un nuevo orden por relevancia (nuevos pesos) para todas las características.

2.3.2.3. Métodos de envoltura (wrapper)

Los métodos de envoltura (wrapper) buscan un subconjunto de características que maximice el rendimiento de un clasificador en particular (Pes, 2019). Este tipo de métodos implica elegir un clasificador que se utilizará para evaluar a los subconjuntos de características candidatos, la métrica de evaluación (por ejemplo, MCC, precisión, exactitud, entre otros), y la estrategia de búsqueda para reducir la cantidad de subconjuntos a evaluar y evitar así una búsqueda exhaustiva. Es importante mencionar que las estrategias de búsqueda no garantizan la selección del subconjunto óptimo de características, pero ayudan a reducir el coste computacional ya que los métodos de tipo *wrapper* implican entrenar y evaluar clasificadores para diferentes subconjuntos de características.

Una de las estrategias de búsqueda son los algoritmos genéticos. Un algoritmo genético es un algoritmo de optimización basado en la selección natural y en la genética, buscan la solución óptima o una buena aproximación a ella. Parten de una población inicial de individuos, donde cada individuo representa una solución candidata. Los individuos suelen ser codificados utilizando una cadena de bits, también llamada cromosoma o genotipo. Cada individuo en la población es evaluado con la función de aptitud (por ejemplo la precisión) que mide qué tan bueno es ese individuo en términos de la solución al problema. Los individuos con una mejor aptitud tienen más probabilidades de ser seleccionados para reproducirse y formar la próxima generación. Los individuos seleccionados se cruzan entre sí para crear descendencia. El cruzamiento (reproducción) implica intercambiar partes de los cromosomas de los padres para crear nuevos individuos. A esta nueva descendencia se aplican cambios aleatorios (mutación) a los cromosomas para introducir variabilidad en la población y evita que la búsqueda se estanque en una región específica del espacio de soluciones. Al reemplazar la población original por la descendencia generada puede implicar la eliminación de algunos individuos menos aptos para dar paso a los nuevos individuos. El proceso de selección, reproducción, mutación y reemplazo se repite durante varias generaciones hasta que se alcanza un criterio de terminación predefinido, como un número máximo de generaciones o una mejora aceptable en la aptitud de la población (Mitchell, 1998).

2.3.3. Algoritmos de clasificación

El objetivo en un modelo de clasificación es tomar una instancia representada por un vector que contiene sus características y asignarlo a una de las clases (Bishop & Nasrabadi, 2006). Un algoritmo de aprendizaje

es el que se encarga de aprender patrones en los datos para poder crear un modelo. En esta sección se muestran los algoritmos de aprendizaje para tareas de clasificación que se utilizaron en este trabajo de tesis.

2.3.3.1. Clasificadores individuales

- **K-Vecinos más Cercanos**

  Los algoritmos de aprendizaje "k-vecinos más cercanos" (KNN, por sus siglas en inglés) solo almacena las instancias de entrenamiento y no realizan ningún trabajo real hasta el momento de la clasificación (Witten et al., 2011). En este tipo de clasificadores se supone que, instancias similares tienen clasificaciones similares (Aha & Kibler, 1991). Se pueden crear diferentes clasificadores de este tipo, por ejemplo, la implementación IB1 para predecir la clase de una nueva instancia usando la distancia euclidiana para buscar instancia de entrenamiento más cercana (vecino) y asignarla a la misma clase. Se puede usar otra métrica para calcular la distancia, así como diferentes algoritmos de búsqueda para acelerar la búsqueda del vecino más cercano (Witten et al., 2011). También se puede variar la cantidad de vecinos a considerar, lo que da lugar a los algoritmos IBk que buscan los k vecinos más cercanos para clasificar la nueva instancia, cada vecino vota por su clase, y la clase con más votos se toma como la predicción para la nueva instancia. Existe una variante donde a los votos de los vecinos se les asigna un peso para dar más importancia a los más cercanos a la nueva instancia función de su proximidad por ejemplo, $1/distancia$ o $1 - distancia$. Así el voto de cada vecino se multiplica por su peso de distancia correspondiente. Luego, se suman los votos ponderados de todos los vecinos para cada clase.

- **Máquina de Soporte Vectorial**

  Los algoritmos de aprendizaje "Máquina de Soporte Vectorial" (SVM, por sus siglas en inglés), se pueden escribir como ecuaciones matemáticas de una manera razonablemente natural. Viendo las instancias como vectores de dimensión m (porque hay características m), los modelos lineales separan los vectores con un hiperplano de dimensión $m + 1$ de tal manera que de un lado queden instancias de una misma clase y del otro lado las instancias de la otra clase. Maximizando la distancia de él con el punto de más cercano en cada lado. Los SVM crean límites no lineales transformando el espacio de las instancias en un nuevo espacio de mayor dimensión, luego se usan modelos lineales para encontrar un hiperplano que separe los datos en este nuevo espacio (Witten

et al., 2011). El kernel es una función matemática que representa la similitud o cercanía de dos vectores en el nuevo espacio y necesario para la construcción del modelo lineal (Üstün et al., 2006). Por ejemplo, está el kernel polinomial que calcula $K(x_1, x_2) =< X, Y >^p$, el Kernel Universal Pearson VII, PUK por sus siglas en inglés $K(x_1, x_2) = \left[1 + \left(\frac{2\sqrt{||x_1 - x_2||^2}\sqrt{2^{1/\omega} - 1}}{\sigma}\right)^2\right]^{-\omega}$.

- **Red Bayesiana**

  Una Red Bayesiana produce estimaciones de probabilidad en lugar de clasificaciones estrictas. Para cada clase, estima la probabilidad de que una determinada instancia pertenezca a esa clase (Witten et al., 2011). Una red bayesiana consta de dos componentes: un grafo acíclico dirigido y un conjunto de tablas de probabilidad condicional. Los nodos del grafo representan las variables de interés (caracteríticas), mientras que las conexiones entre ellos tienen una interpretación formal en términos de independencia probabilística. Las tablas de probabilidad condicional cuantifican la relación entre una variable y sus nodos padre en la red, por lo que cada variable tiene una tabla asociada (Darwiche, 2010). La red bayesiana debe satisfacer la condición de Markov: "cada nodo debe ser independiente de los otros nodos de la red (salvo sus descendientes) dados sus padres". Lo cual induce una distribución conjunta para las n variables X_k en el grafo (Cruz et al., 2021):

$$P(X_1 = x_1, \ldots, X_n = x_n) = \prod_{k=1}^{n} P(X_k = x_k \mid pa(X_k) = \pi_k), \tag{12}$$

  Donde $pa(X_k)$ denota el conjunto de padres de X_k y π_k denota la proyección de $\{x_1, \ldots, x_n\}$ en $pa(X_k)$. Si una variable X_k no tiene padres, entonces tomamos su término correspondiente como $P(X_k = x_k)$. Con esta fórmula se predice la probabilidad de pertenecer a cada clase una instancia dada.

- **Árboles de Decisión**

  Los árboles de decisión son un tipo de representación del aprendizaje donde cada nodo representa una prueba de una característica y cada rama que sale de él es un valor posible de la característica. Los nodos hoja proporcionan una clasificación que se aplica a todas las instancias que lleguen a ese nodo hoja (Witten et al., 2011). Existen varios algoritmos para crear árboles de decisión, por ejemplo, J48, REPTree, Random Tree, entre otros.

  El algoritmo **J48** es una implementación del algoritmo de árbol de decisión C4.5, desarrollado por Quinlan (1993). Se basa en el paradigma divide y vencerás para producir árboles de decisión. Comenzando con el conjunto de datos de entrenamiento completo en la raíz del árbol, para cada nodo, se selecciona la mejor característica para dividir el conjunto de datos en subconjuntos más

pequeños. La mejor característica es aquella que maximiza la ganancia de información, que mide la reducción de la entropía del conjunto de datos después de dividirlo según una característica. El proceso de selección continúa recursivamente hasta que se satisfaga un cierto criterio de detención, como la profundidad máxima del árbol o el número mínimo de ejemplos en un nodo (Witten et al., 2011).

Una variante del algoritmo de árbol de decisión estándar es **RandomTree**, que introduce aleatoriedad en el proceso de construcción del árbol. A diferencia de J48 que seleccionan la mejor característica para dividir los nodos en cada paso, RandomTree en cada nodo elige una característica de k características extraídas aleatoriamente (Witten et al., 2011).

El algoritmo **REPTree** o "Árbol con Poda de Error Reducido" construye un árbol de decisión y luego realiza una poda de error reducido para reducir el error de clasificación (Witten et al., 2011). Verifica cada nodo interno, desde abajo hacia arriba, y si reemplazar un nodo con la clase más frecuente no reduce la precisión del árbol entonces el nodo se poda. El procedimiento continúa hasta que cualquier poda adicional disminuya la precisión. Para estimar la precisión, se retienen algunos de los datos proporcionados para entrenamiento y se utilizan como un conjunto de prueba independiente (Maimon & Rokach, 2005).

2.3.3.2. Clasificadores basados en consenso

Los clasificadores basados en consenso son creados a partir de combinar diferentes modelos. Como explican Witten et al. (2011), es como tener un grupo de especialistas, donde cada uno sobresale en un dominio limitado. A pesar de que individualmente puedan equivocarse, como conjunto tomarán mejores decisiones. Hay varias maneras de crear clasificadores basados en consenso, en esta sección se presentan algunos de ellos.

- **Bagging**

  El enfoque que sigue bagging para combinar las decisiones de diversos modelos en una sola predicción en el caso de modelos de clasificación es mediante una votación. De manera general, lo que hace el algoritmo de bagging es crear varios subconjuntos de datos de entrenamiento del mismo tamaño mediante un muestreo con reemplazo. Luego, para cada uno de los subconjuntos, se crea un modelo con un algoritmo de aprendizaje en particular y se guarda. Así, a la hora de predecir la

clase de una instancia, la predicción final es la clase que fue predicha por la mayoría de los modelos (Witten et al., 2011). **Random Forest** es un ejemplo de un clasificador creado con el algoritmo bagging, con la particularidad de que crea árboles de decisión como modelos individuales con el algoritmo RandomTree (Breiman, 2001).

- **Boosting**

  A diferencia de bagging, boosting crea diferentes modelos de manera iterativa. De tal manera que cada nuevo modelo está influenciado por el rendimiento de los modelos construidos previamente. Alentando a los nuevos modelos a especializarse en instancias que fueron manejadas incorrectamente por los modelos anteriores. Además, para la clasificación final, boosting combina las salidas de los diferentes modelos con una votación ponderada, donde los modelos que mostraron un mejor rendimiento tienen una mayor influencia (Witten et al., 2011). **AdaBoost** es un ejemplo de boosting. Según Witten et al. (2011), el algoritmo comienza asignando el mismo peso a todas las instancias en los datos de entrenamiento. Primeramente, se crea un clasificador con estos datos, el cual es usado para predecir la clase de las instancias y se vuelve a ponderar cada instancia de tal manera que el peso de las instancias correctamente clasificadas se reduce mientras que aumenta el de las mal clasificadas. En cada iteración se crea un nuevo clasificador con los datos ponderados, que en consecuencia se enfoca en clasificar correctamente las instancias con pesos más altos. Luego, los pesos de las instancias aumentan o disminuyen de acuerdo con la salida de este nuevo clasificador. Finalmente, para formar una predicción, combinan los resultados de los fiferentes modelos mediante un voto ponderado. Para ver más detalles sobre el cálculo del error de los clasificadores, cómo se actualizan los pesos y cómo se pondera el voto de cada clasificador, ver Freund & Schapire (1996). **LogitBoost** es otro ejemplo de boosting. Las diferencias entre LogitBoost y AdaBoost es que LogitBoost utiliza una regresión logística aditiva como algoritmo de aprendizaje base. Luego, LogitBoost utiliza una función de pérdida logística para medir el error de los clasificadores en cada iteración. Además, en cómo actualizan los pesos de las instancias. Ver Friedman et al. (2000) para más detalle.

- **Random Commite**

  El enfoque Random Committee es bastante sencillo, consiste en crear varios clasificadores y luego la predicción final de la clase se da por mayoría de votos. Cada uno de estos clasificadores se construye utilizando los mismos datos, pero se diferencian entre sí en las semillas de números aleatorios ya que cada uno usará una semilla diferente. Esto solo resulta útil si el algoritmo de aprendizaje base es aleatorio; de lo contrario, todos los clasificadores serían idénticos y no aportarían diversidad en sus predicciones (Witten et al., 2011).

2.4. Evaluación de modelos

En el campo del aprendizaje automático, los modelos de clasificación deben mantener un equilibrio entre el error de sesgo (bias) y el error de varianza. Estos conceptos están estrechamente relacionados con los fenómenos de sobreajuste (overfitting) y subajuste (underfitting). El primero se da cuando un modelo presenta alta varianza y bajo sesgo, ya que tiende a sobreajustarse a los datos de entrenamiento. Es decir, el modelo se adapta en exceso a los detalles y al ruido presentes en los datos de entrenamiento, llegando incluso a capturar irregularidades y excepciones. En este caso, el modelo se especializa tanto en los datos de entrenamiento que pierde la capacidad de realizar predicciones precisas en datos desconocidos. El segundo concepto se da cuando un modelo presenta alto sesgo y baja varianza, porque tiende a subajustarse a los datos de entrenamiento. En este caso, el modelo es demasiado simple para capturar la complejidad de los datos y no puede ajustarse lo suficiente, incluso a los patrones evidentes en los datos de entrenamiento. Como resultado, el modelo tendrá un rendimiento pobre tanto en los datos de entrenamiento como en los datos nuevos. Para evaluar el sesgo y la varianza en un modelo, se pueden aplicar diversos enfoques. Por ejemplo, se pueden comparar los errores de entrenamiento y prueba utilizando métricas de desempeño, así como emplear técnicas como la validación cruzada.

En problemas de clasificación binaria, donde se tiene una variable objetivo que puede tener dos valores posibles, los siguientes términos son utilizados para describir los resultados de las predicciones realizadas por el modelo en comparación con los valores reales de los datos:

- Verdaderos Positivos o TP (True Positives): Representan los casos en los que el modelo ha predicho correctamente una clase positiva. Es decir, el modelo predice que una instancia pertenece a la clase positiva y la instancia realmente pertenece a esa clase.
- Verdaderos Negativos o TN (True Negatives): Representan los casos en los que el modelo ha predicho correctamente una clase negativa. Es decir, el modelo predice que una instancia pertenece a la clase negativa y la instancia realmente pertenece a esa clase.
- Falsos Positivos o FP (False Positives): Representan los casos en los que el modelo ha predicho incorrectamente una clase positiva. Esto ocurre cuando el modelo predice que una instancia pertenece a la clase positiva, pero en realidad la instancia no pertenece a esa clase.
- Falsos Negativos FN (False Negatives): Representan los casos en los que el modelo ha predicho incorrectamente una clase negativa. Esto ocurre cuando el modelo predice que una instancia pertenece a la clase negativa, pero en realidad la instancia no pertenece a esa clase.

En la siguiente lista se muestran las métricas de desempeño que se utilizaron en este trabajo de tesis para evaluar y comparar el rendimiento de modelos de clasificación binaria (Chicco & Jurman, 2020). Estas métricas proporcionan una medida cuantitativa de qué tan bien se está desempeñando un modelo en la tarea que se le ha asignado.

- El coeficiente de correlación de Matthews o MCC por sus siglas en inglés, varía entre -1 y $+1$. Un valor de $+1$ indica una clasificación perfecta, 0 indica una clasificación aleatoria y -1 indica una clasificación inversa. El MCC es una métrica útil cuando hay un desequilibrio en las clases o cuando se desea tener en cuenta tanto los verdaderos como los falsos positivos y negativos en la evaluación del modelo.

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP) \times (TP + FN) \times (TN + FP) \times (TN + FN)}} \tag{13}$$

- La exactitud o Accuracy, indica la proporción de predicciones correctas realizadas en comparación con el total de predicciones realizadas.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{14}$$

- La precisión es la proporción de predicciones positivas correctas sobre el total de predicciones positivas realizadas por el modelo.

$$\text{Precisión} = \frac{TP}{TP + FP} \tag{15}$$

- La sensibilidad es la proporción de predicciones positivas correctas sobre el total de ejemplos positivos reales.

$$\text{Sensibilidad} = \frac{TP}{TP + FN} \tag{16}$$

- La especificidad es la proporción de predicciones negativas correctas sobre el total de ejemplos negativos reales.

$$\text{Especificidad} = \frac{TN}{TN + FP} \tag{17}$$

- La métrica F1 o F1-Score es la media armónica entre la precisión y la sensibilidad. Proporciona una medida equilibrada del desempeño del modelo, es útil cuando hay un desequilibrio entre las clases en los datos.

$$\text{Valor-F} = \frac{2 \times Precision \times Sensibilidad}{Precision + Sensibilidad} \tag{18}$$

- El coeficiente Kappa de Cohen o Cohen's Kappa, métrica utilizada para evaluar la concordancia. Es una medida estadística que ajusta el efecto del azar en la proporción de la concordancia observada. Este coeficiente varía entre -1 y 1. Un valor de 1 indica una concordancia perfecta, 0 indica una concordancia aleatoria y -1 indica una discordancia total.

$$\text{Kappa} = \frac{Po - Pe}{1 - Pe} \tag{19}$$

Donde:

$$Po = \frac{TP + TN}{TP + TN + FP + FN} \tag{20}$$

$$Pe = \frac{(TP + FP)(TP + FN) + (TN + FP)(TN + FN)}{(TP + TN + FP + FN)^2} \tag{21}$$

2.5. Conclusiones parciales

En este capítulo se ha establecido una base sólida para comprender los conceptos clave relacionados con los péptidos antimicrobianos y la modelación QSAR que serán abordados a lo largo del documento. En primer lugar, se ha analizado la composición de los péptidos, destacando los aminoácidos como sus componentes fundamentales, y se ha introducido la función de los péptidos antimicrobianos como moléculas con propiedades defensivas contra patógenos. Como vimos, los péptidos antimicrobianos presentan diferencias entre sí en cuestiones topológicas (secuencia), estructurales y modo de acción. Sin embargo, se han logrado identificar características comunes en la mayoría de ellos, como por ejemplo, la carga neta positiva entre $+2$ y $+9$, y su tendencia a contener más del 50 % de aminoácidos hidrofóbicos.

Se abordó también la importancia de los descriptores en la caracterización de péptidos y proteínas, haciendo énfasis en los descriptores aprendidos (non-handcrafted) a partir de secuencias polipeptídicas. De esta manera, cada péptido es representado mediante un vector de características y una etiqueta de clase binaria (*endpoint*) para la construcción de modelos QSAR. Se han expuesto los pasos fundamentales en el proceso de modelación QSAR, incluyendo técnicas de limpieza de datos, selección de características y algoritmos de clasificación. Dado que no existe un método único que funcione bien para todos los problemas de clasificación, se ha buscado abarcar una amplia gama de técnicas y herramientas en cada uno de estos pasos. Finalmente, se han presentado diversas métricas de desempeño que permiten una evaluación rigurosa y objetiva de la calidad y eficacia de los modelos desarrollados.

Capítulo 3. Metodología

En este capítulo, se presenta la metodología empleada, ofreciendo una comprensión detallada de cómo se llevó a cabo el estudio que se realizó en este trabajo de tesis y permitiendo contextualizar los resultados y conclusiones presentados en capítulos subsiguientes. La metodología abarca desde los conjuntos de péptidos antimicrobianos utilizados en esta investigación, la extracción de características, y la implementación y aplicación de un flujo de trabajo en KNIME para la generación automática de clasificadores.

3.1. Conjuntos de datos de péptidos antimicrobianos

En esta sección se describen cuatro conjuntos de datos de secuencias de péptidos del estado del arte que fueron desarrollados por Sharma et al. (2021a,b,c,d) para construir los modelos basados en aprendizaje profundo AniAMPpred, Deep-ABPpred, Deep-AFPpred y Deep-AVPpred. Estos modelos tienen como finalidad predecir péptidos antimicrobianos en general (general-AMPs), péptidos antibacterianos (ABPs), péptidos antifúngicos (AFPs) y péptidos antivirales (AVPs), respectivamente. Como se explica a continuación, cada una de estos cuatro conjuntos de datos se dividieron en conjuntos de entrenamiento y prueba por los propios autores (consultar la Tabla 2). Estas mismas divisiones se utilizaron en este trabajo de tesis con el fin de garantizar comparabilidad de resultados.

Tabla 2. Conjuntos de entrenamiento, validación y prueba utilizados en este trabajo.

	Conjunto de Entrenamiento			**Conjunto de Validación**			**Conjunto de Prueba**		
Conjunto de datos	Total	Positivos	Negativos	Total	Positivos	Negativos	Total	Positivos	Negativos
AMP (Sharma et al., 2021a)	13,430	6,657	6,773	-	-	-	7,179	3,530	3,649
ABP (Sharma et al., 2021b)	3,120	1,635	1,485	-	-	-	9,816	4,017	5,799
AFP (Sharma et al., 2021c)	4,124	2,062	2,062	-	-	-	2,758	1,379	1,379
AVP (Sharma et al., 2021d)	4,908	2,454	2,454	1,636	818	818	1,636	818	818

3.1.1. Conjunto de datos de péptidos antimicrobianos generales

Sharma et al. (2021a) obtuvieron las secuencias de péptidos General-AMPs de las bases NCBI (Schoch et al., 2020) y StarPep (Aguilera-Mendoza et al., 2019). Dado que no existe un repositorio que contenga secuencias de péptidos con evidencia experimental relacionada con la ausencia completa de actividad antimicrobiana (es decir, no-AMP), los autores generaron los ejemplos negativos a partir de secuencias de proteínas revisadas y anotadas manualmente que están disponibles en la base de datos “Universal

Protein Resource" (UniProt) (Dogan, 2018). Ellos excluyeron las secuencias que contenían las siguientes palabras claves: antimicrobial, antibacterial, anti-Gram positive, anti-Gram negative, anti-cancer, antitumor, anti-TB, antitoxin, antiviral, antimalarial, anti-HIV, antifungal, antiparasitic, antidiabetic, insecticidal, antioxidant, antibiotic, antiprotozoal, antibiofilm, anti-MRSA, anti-endotoxin, chemotactic, anti-inflammatory, anti-protist. De las secuencias recuperadas, eliminaron las secuencias duplicadas, aquellas con aminoácidos no estándar, y las que tenían una longitud menor a 10 o mayor a 200. Posteriormente, excluyeron las secuencias no-AMP que compartían más del 80 % de similitud con las secuencias AMP. Finalmente, seleccionaron aproximadamente el 65 % de las secuencias positivas y negativas para formar el conjunto de entrenamiento, y el resto lo utilizaron para el conjunto de prueba (ver Tabla 2).

3.1.2. Conjunto de datos de péptidos antibacterianos

Sharma et al. (2021b) obtuvieron las secuencias de péptidos ABP de las bases de datos "Antimicrobial Peptide Database" (APD) (Wang et al., 2015), "Data Repository of Antimicrobial Peptides" (DRAMP) (yue Kang et al., 2019) y "Milk Antimicrobial Peptides Database" (MilkAMP) (Théolier et al., 2014). Eliminaron las secuencias que tenían una carga neta menor a $+2$. Los autores generaron las secuencias no-ABP a partir de UniProt, excluyendo aquellas que contenían las siguientes palabras claves: antimicrobial, antibacterial, anti-Gram positive, anti-Gram negative, anti-cancer, antitumor, anti-TB, antitoxin, antiviral, antimalarial, anti-HIV, antifungal, antiparasitic, antidiabetic, insecticidal, antioxidant, antibiotic, antiprotozoal, antibiofilm, anti-MRSA, anti-endotoxin, chemotactic, anti-inflammatory, anti-protist, antiparasitical, spermicidal, secreted, excreted y effector. Adicionalmente, recuperaron 31 secuencias no activas verificadas experimentalmente de MilkAMP.

De las secuencias recuperadas, eliminaron las secuencias duplicadas, aquellas con aminoácidos no estándar, y las secuencias con longitud menor a 4 o mayor a 30. Finalmente, eliminaron las secuencias que estaban presentes tanto en el conjunto de ABP como en el conjunto no-ABP. Todas estas secuencias recuperadas las utilizaron para conformar el conjunto de entrenamiento. Mientras que, para formar el conjunto de prueba, extrajeron secuencias ABP de la base StarPep, así como secuencias no-ABP de Gabere & Noble (2017) y de UniProt siguiendo el mismo proceso explicado previamente. Finalmente, eliminaron las secuencias que ya estaban presentes en el conjunto de entrenamiento (consultar Tabla 2).

3.1.3. Conjunto de datos de péptidos antifúngicos

Sharma et al. (2021c) obtuvieron secuencias de péptidos AFP de las bases de datos CAMP (Waghu et al., 2015), DRAMP y StarPep. Las secuencias no-AFP fueron generadas por los autores a partir de la base de datos UniProt, excluyendo aquellas que contenían las palabras clave: antimicrobial, antibacterial, anti-Gram positive, anti-Gram negative, anti-cancer, antitumor, anti-TB, antitoxin, antiviral, antimalarial, anti-HIV, antifungal, antiparasitic, antidiabetic, insecticidal, antioxidant, antibiotic, antiprotozoal, antibiofilm, anti-MRSA, anti-endotoxin, anti-inflammatory, anti-protist, secreted, excreted, effector, antiendotoxin, cytokine, bacteriocin. De las secuencias recuperadas, eliminaron las secuencias duplicadas, aquellas con aminoácidos no estándar, y las secuencias con longitud menor a 5 o mayor a 30. Del conjunto de secuencias AFP y no-AFP extrajeron aproximadamente el 60 % de cada grupo para formar el conjunto de entrenamiento, mientras que el resto lo utilizaron para el conjunto de prueba (ver Tabla 2).

3.1.4. Conjunto de datos de péptidos antivirales

Sharma et al. (2021d) obtuvieron secuencias de péptidos AVP de las bases de datos DRAMP, StarPep, AVPpred (Thakur et al., 2012), DBAASP (Pirtskhalava et al., 2015) y SATPDB (Singh et al., 2015). Las secuencias no-AVP fueron generadas por los autores a partir de la base de datos UniProt, excluyendo aquellas que contenían las palabras clave: antimicrobial, antibacterial, anti-Gram positive, anti-Gram negative, anti-cancer, antitumor, anti-TB, antitoxin, antiviral, antimalarial, anti-HIV, antifungal, antiparasitic, antidiabetic, insecticidal, antioxidant, antibiotic, antiprotozoal, antibiofilm, anti-MRSA, anti-endotoxin, anti-inflammatory, anti-protist, secreted, excreted, effector, defensin, cytokine, bacteriocin. De las secuencias recuperadas, eliminaron las secuencias duplicadas, aquellas con aminoácidos no estándar, y las secuencias con longitud menor a 5 o mayor a 50. Posteriormente, eliminaron las secuencias que estaban presentes tanto en el conjunto AVP como en el conjunto no-AVP. También excluyeron las secuencias no-AVP que compartían más del 70 % de similitud con las secuencias AVP. Además, identificaron motivos de cuatro longitudes que estaban presentes en al menos 20 secuencias AVP, y eliminaron las secuencias no-AVP que incluían estos motivos. Del conjunto de secuencias AVP y no-AVP, extrajeron aproximadamente el 60 % de cada grupo para formar el conjunto de entrenamiento. El resto lo dividieron en dos partes para formar conjuntos de validación y prueba. Sin embargo, en este trabajo de tesis, el conjunto de validación no se utilizó para ningún propósito (consultar Tabla 2).

3.1.5. Caracterización de los conjuntos de datos de péptidos antimicrobianos

En esta sección se presenta una caracterización de los conjuntos de péptidos antimicrobianos utilizados en el presente trabajo. En este sentido, la Figura 4 muestra la frecuencia en la que aparece cada aminoácido en cada uno de los conjuntos de datos, distinguiendo entre el conjunto de entrenamiento y el conjunto de prueba. Se observa que en cada conjunto de datos se preserva aproximadamente la misma distribución de frecuencias entre los respectivos conjuntos de entrenamiento y prueba. Sin embargo, se puede observar en la gráfica correspondiente a ABP que todos los aminoácidos en el conjunto de prueba tiene más frecuencia que en el conjunto de entrenamiento. Esto se debe a que el conjunto de prueba tiene más instancias que el de entrenamiento, ya que, los autores crearon el conjunto de prueba desde StarPep (Aguilera-Mendoza et al., 2019), la cual no usaron para crear el conjunto de entrenamiento a pesar de que StarPep es la base de datos con el mayor número de péptidos antimicrobianos reportados hasta la fecha.

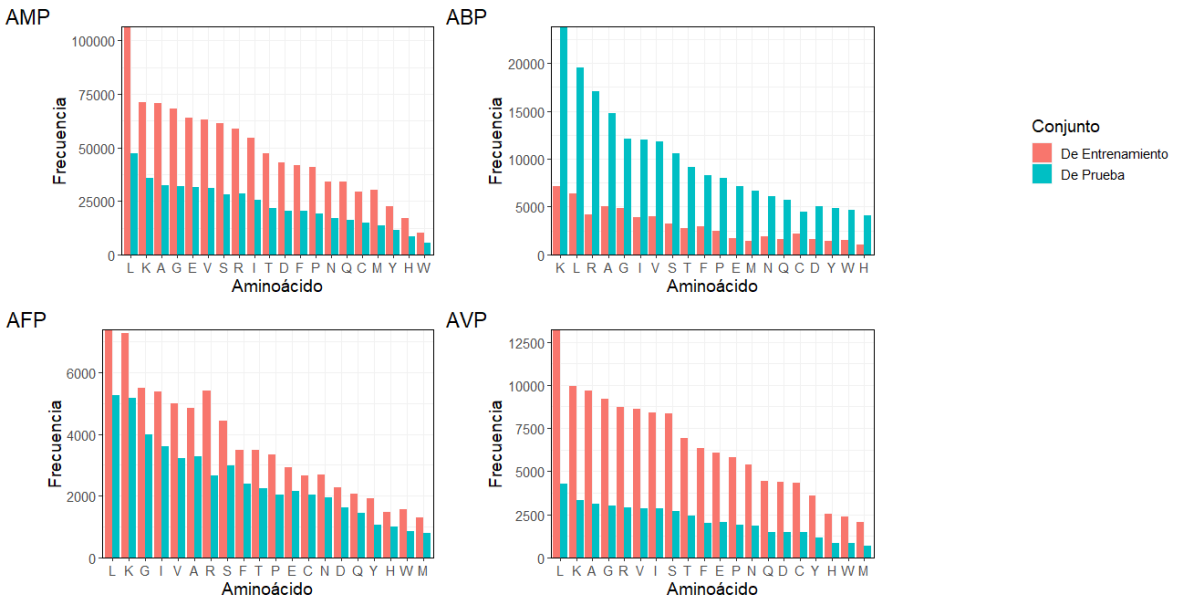

Figura 4. Frecuencia de aminoácidos presentes en cada conjunto de datos por conjunto de entrenamiento y prueba.

Por otro lado, en la Figura 5 se observa que las frecuencias de las longitudes de las secuencias presentan un comportamiento similar entre los conjuntos de entrenamiento y prueba. Por ejemplo, en el conjunto de datos de ABP, las longitudes de 4 a 13 son las menos frecuentes tanto en el conjunto de entrenamiento como en el conjunto de prueba. Sin embargo, hay una excepción en el conjunto de datos de AMP en la que no es tan evidente esta similitud en la forma de la distribución de las frecuencias entre el conjunto

de entrenamiento y prueba. Por ejemplo, la cantidad de secuencias con una longitud dentro del intervalo de 100 a 200 y del intervalo de 90 a 100 es mucho mayor en el conjunto de entrenamiento que en el de prueba, en comparación con lo que sucede en otros intervalos, como en el de secuencias con longitud entre 50 y 60, donde ambos conjuntos tienen una cantidad similar.

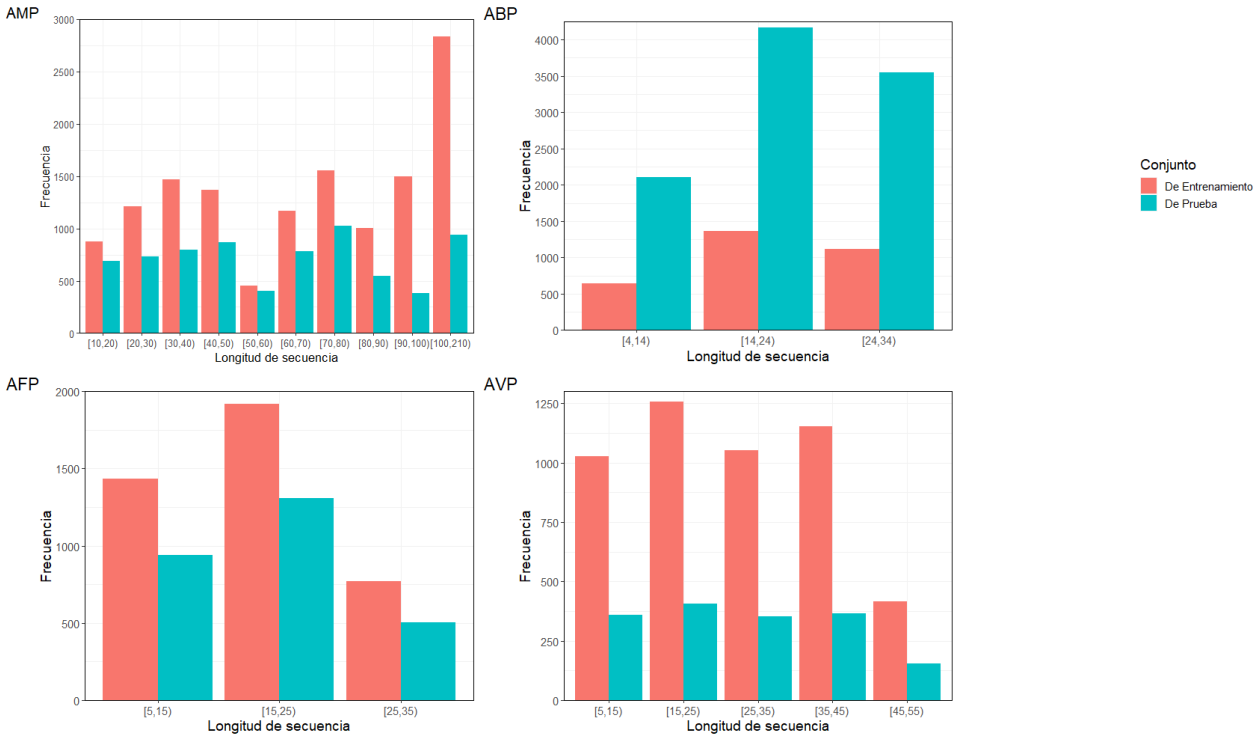

Figura 5. Frecuencia de las longitudes de las secuencias presentes en cada conjunto de datos por conjunto de entrenamiento y prueba.

Finalmente, se muestra un análisis de diversidad en los conjuntos de entrenamiento y prueba. El índice de diversidad es una manera de estimar qué tan diverso es un conjunto de datos. El cual se obtiene aplicando una técnica de agrupamiento (clustering) en la que el conjunto de datos se organiza de tal manera que las secuencias dentro de cada grupo sean similares entre sí de acuerdo con un umbral de similitud establecido, mientras que las secuencias entre grupos diferentes se consideran distintas. El índice de diversidad se calcula como el número de grupos formados dividido entre el total de secuencias. Una implementación para calcularlo se encuentra en el software Dover Analyzer desarrollado por Aguilera-Mendoza et al. (2015).

Usando Dover Analyzer, se generaron las gráficas que se presentan en la Figura 6. Estas gráficas muestran el índice de diversidad en función del umbral de identidad para cada uno de los conjuntos de datos, distinguiendo entre los conjuntos de entrenamiento y prueba. Se observa que en los conjuntos de datos

AMP, AFP y AVP, los conjuntos de prueba presentan niveles de diversidad comparables o incluso superiores a los de los conjuntos de entrenamiento. Este comportamiento puede ser adecuado para evaluar la capacidad de generalización de los modelos construidos, ya que si demuestran un buen poder predictivo, indica que generalizan bien en conjuntos de datos más diversos.

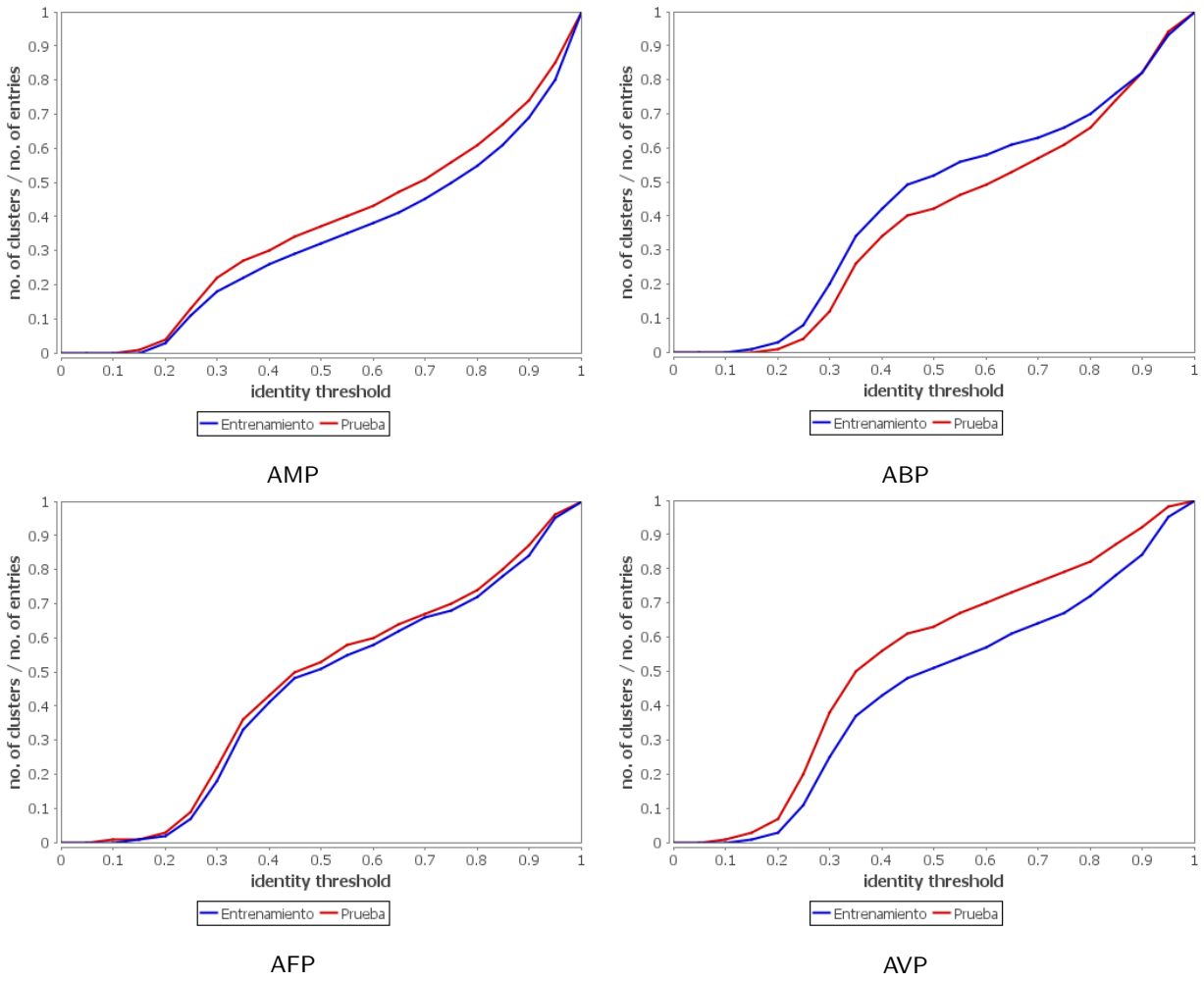

Figura 6. Para cada conjunto de datos (AMP, ABP, AFP y AVP), se muestra su gráfica correspondiente que representa el radio de diversidad (el número de grupos de similitud entre el total de datos) en función del umbral de similitud, diferenciando entre el conjunto de entrenamiento y el conjunto de prueba.

3.2. Extracción de características

A partir de las secuencias de los péptidos presentes en los cuatro conjuntos de datos, se calcularon características (descriptores) aprendidas mediante la extracción de incrustaciones (embeddings) de dimensiones 320, 480, 640, 1280 y 2560 en cada conjunto de datos descrito en la Tabla 2. Estas incrusta-

ciones se obtuvieron de modelos con 6, 12, 30, 33 y 36 capas, respectivamente, de la familia de modelos ESM-2 como se explica en la sección 2.2. Las incrustaciones se calcularon promediando la incrustación por aminoácido de la capa final obtenida para cada secuencia de péptidos. Estas incrustaciones se usaron como conjunto de características iniciales en el flujo de trabajo de KNIME, el cual se detalla en la siguiente sección. Debido a limitaciones de hardware (NVIDIA RTX A5500 24 GB), no se pudo utilizar el modelo ESM-2 de 48 capas para extraer incrustaciones de 5120 dimensiones, ya que requiere más memoria de GPU que la especificada anteriormente.

3.3. Flujo de trabajo en KNIME

El flujo de trabajo fue construido utilizando el software Konstanz Information Miner (KNIME) v4.7.2, el cual está disponible en `https://www.knime.com/`. KNIME es una plataforma de código abierto que proporciona una interfaz gráfica de usuario para la creación de flujos de trabajo. Esta plataforma permite usar nodos tanto de la Plataforma Analítica de KNIME como de sus Extensiones e Integraciones (por ejemplo, scripts de Python, y clasificadores del framework Weka v7.0).

El flujo de trabajo se diseñó de manera genérica, lo que significa que se creó para construir automáticamente modelos de clasificación binaria a partir de cualquier conjunto de datos tabular, y no solo para la clasificación de AMPs y sus tipos funcionales. En la Figura 13 se muestra una vista principal del flujo de trabajo, el cual está compuesto por cinco meta-nodos representando la configuración de inicio, seguido de las etapas de limpieza de datos, entrenamiento, evaluación, y la configuración final, respectivamente. En esta sección, se explica la implementación de cada una de estas etapas, los requisitos para instalar el flujo de trabajo, y cómo utilizarlo.

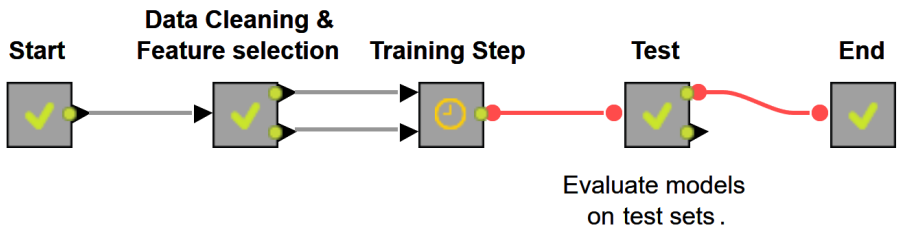

Figura 7. Vista principal del flujo de trabajo de KNIME, donde cada meta-nodo contiene la implementación de las secciones correspondientes al desarrollo de modelos de aprendizaje automático.

3.3.1. Implementación

En esta subsección se explican los detalles más relevantes de cada una de las etapas que integran el flujo de trabajo.

3.3.1.1. Limpieza de datos

En la etapa de preprocesamiento se realiza una limpieza de datos en el archivo CSV de entrenamiento (ver Figura 8). El proceso comienza con la imputación de valores perdidos (representados por -9999), los cuales son reemplazados por cero. Posteriormente, se aplica un filtro basado en el estadístico curtosis para eliminar características constantes o cercanas a constante. La curtosis es una medida que indica la forma de la distribución de los datos; un valor alto indica una distribución con picos y colas pesadas. En este caso, se utiliza un umbral de 11 para eliminar características con una curtosis elevada.

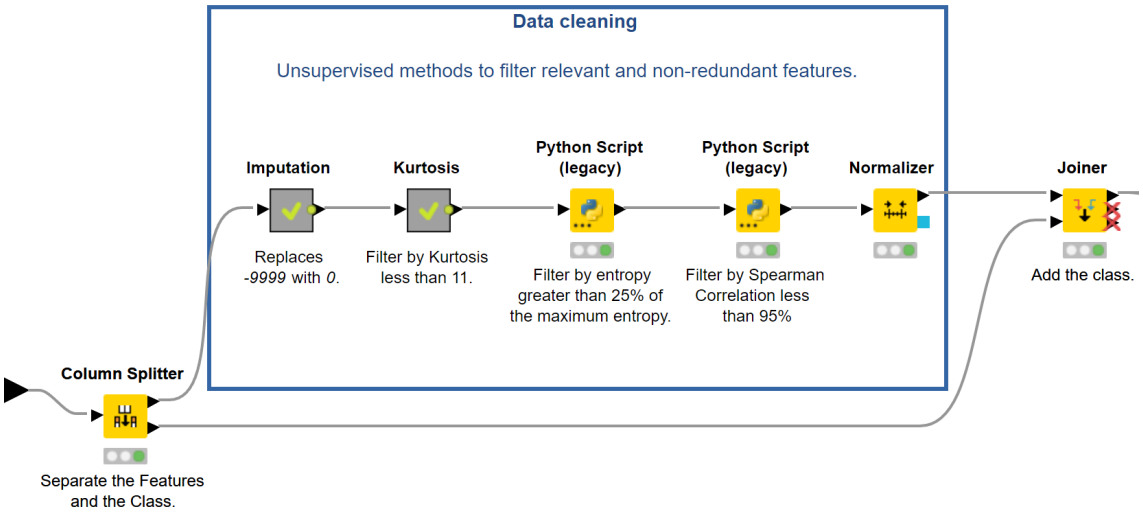

Figura 8. Limpieza de datos en el meta-nodo "Data Cleaning & Feature selection" del flujo de trabajo de KNIME.

Luego, se aplica un filtro de relevancia no supervisado basado en la Entropía de Shannon (SE) para retener aquellas características con valores de SE mayores al 25 % del valor máximo posible de SE que una característica puede alcanzar (ver Fragmento 1 en Anexos). La Entropía de Shannon es una medida de la información contenida en una variable, y valores altos indican una mejor capacidad de discriminación entre

diferentes instancias (en este caso, secuencias de péptidos). El esquema de discretización utilizado genera una cantidad de intervalos igual al número de instancias en el conjunto de entrenamiento considerado. Por lo tanto, el máximo valor de SE de cada característica en los conjuntos de datos generales de AMP, ABP, AFP y AVP es igual a 13.71, 11.61, 12.01 y 12.26 bits, respectivamente.

Después, se eliminan características redundantes mediante un filtro basado en el coeficiente de correlación de Spearman, con un umbral de 0.95 (ver Fragmento 2 en Anexos). Finalmente, esta etapa concluye con una normalización utilizando la estrategia Min-Max, que ajusta los valores de las características dentro del rango entre 0 y 1. Para implementar los filtros de relevancia y redundancia basados en Entropía de Shannon y el coeficiente de correlación de Spearman, respectivamente, se utilizaron scripts de Python en KNIME. Estos scripts emplearon las bibliotecas SciPy v1.10.1 y joblib v1.2.0 para la implementación paralela del filtro de relevancia. Para más detalles sobre los métodos utilizados, consultar la sección 2.3.1.

3.3.1.2. Selección de características

Tras finalizar el preprocesamiento, se lleva a cabo una etapa de selección de características supervisadas utilizando siete métodos diferentes (ver Figura 9). Uno de ellos consiste en la aplicación del algoritmo CFS (Fragmento 3 en Anexos), mientras que los demás son selectores de tipo filtro (ranking) que utilizan los enfoques Relief-F, F de ANOVA, Chi-cuadrado, índice de Gini, t-Score e información mutua (Fragmento 4 en Anexos). La cantidad de características a elegir con los selectores de tipo filtro es igual al tamaño del mejor subconjunto obtenido mediante el algoritmo CFS. Para más detalles sobre cada uno de estos métodos, consultar la sección 2.3.2. Se emplearon scripts de Python en KNIME para aplicar los selectores de tipo filtro disponibles en las bibliotecas scikit-learn (Buitinck et al., 2013) y scikit-feature (Li et al., 2017), mientras que para implementar el método CFS se usó la biblioteca python-weka-wrapper3 v0.2.12 (Witten et al., 1999).

Además, se crean adicionalmente 23 subconjuntos de características mediante métodos de fusión de la siguiente manera: (1) uniendo las características seleccionadas por al menos uno de los selectores mencionados anteriormente; (2) uniendo las características por cada par de subconjuntos de características conformados por los selectores mencionados anteriormente ($C(7,2) = 21$); y (3) seleccionando las mejores características según el ranking promedio de los rankings generados con cada método de tipo filtro aplicado. Cuanto menor sea el ranking promedio, más relevante es la característica.

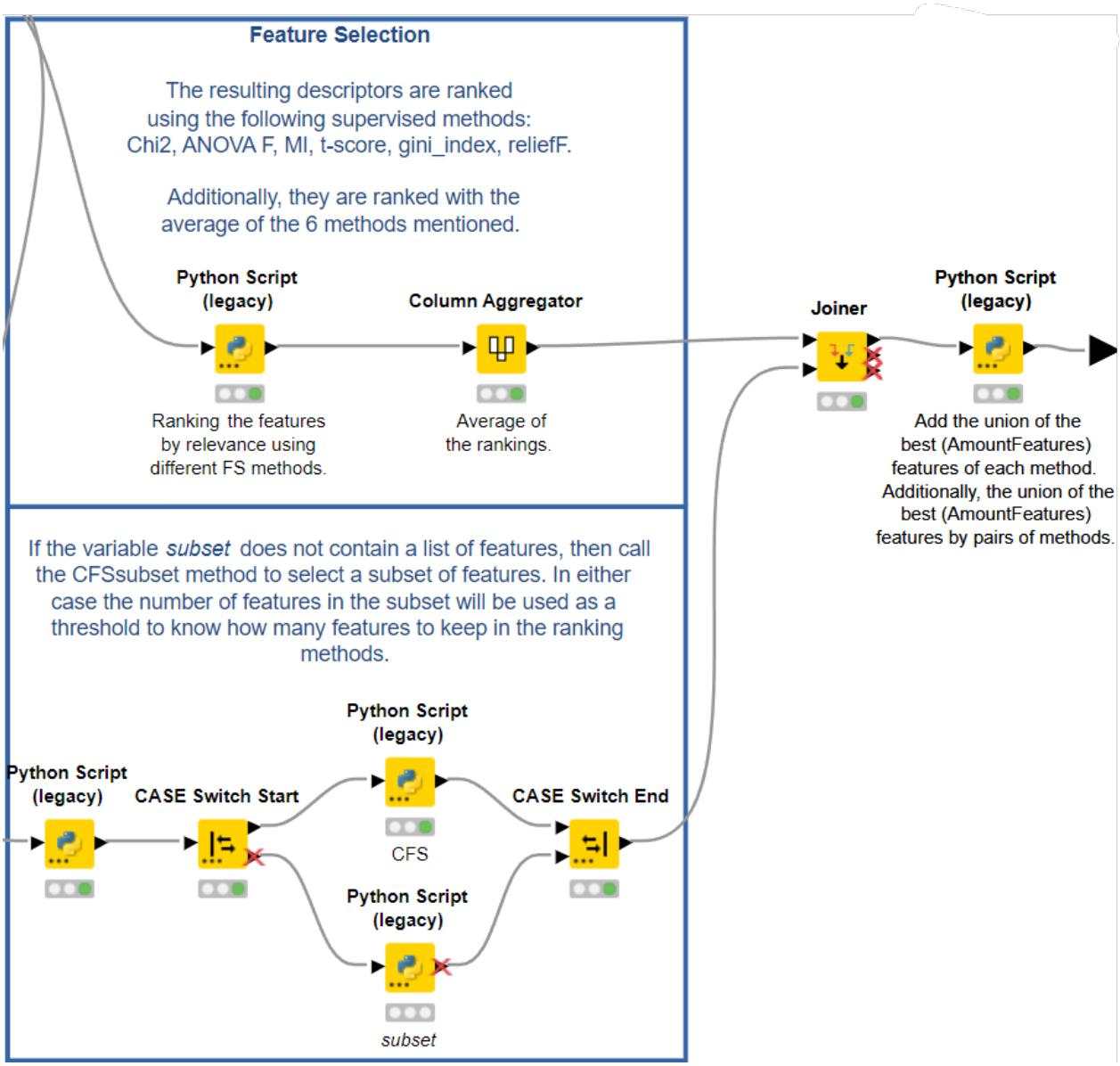

Figura 9. Selección de características en el meta-nodo "Data Cleaning & Feature selection" del flujo de trabajo de KNIME.

3.3.1.3. Algoritmos de aprendizaje

Los algoritmos de aprendizaje utilizados son los siguientes: Random Forest (RF), J48, Reduce Error Pruning Tree (REPTree), Random Tree, Bayes Nets, k-NN y SVM. Para obtener una breve explicación de cada uno de estos algoritmos, consultar la sección 2.3.3. Se utilizó la implementación estos algoritmos presentes en nodos de KNIME Weka. Los algoritmos RF, J48, REPTree, Random Tree y Bayes Nets se aplicaron utilizando sus configuraciones por defecto, ver Tabla 3. En el caso del algoritmo k-NN,

se utilizaron los tres esquemas de ponderación de distancias disponibles en el nodo KNIME. El valor óptimo de k se determina mediante una validación cruzada. Por otro lado, el clasificador SVM se aplica utilizando el kernel polinomial (PolyKernel) y el kernel universal basado en la función Pearson VII (Puk). Esto nos da un total de 10 algoritmos de aprendizaje para la construcción de los modelos. Es importante señalar que no se realizó optimización de hiperparámetros durante la construcción de los modelos en el flujo de trabajo KNIME, a excepción de el valor óptimo de k del algoritmo k-NN que se determina mediante la téctnica "validación cruzada dejando uno fuera" para seleccionar el mejor valor de k entre 1 y 30.

Tabla 3. Configuración por defecto de los algoritmos de clasificación de nodos de KNIME Weka.

Clasificador	Configuración por defecto
Random Forest	Genera 100 árboles con profundidad máxima ilimitada. El número de características seleccionadas al azar es igual a log2(total_de_características) + 1. La semilla para generar números aleatorios es igual a 1.
J48	Elimina las partes que no reduzcan el error de entrenamiento. Utiliza un factor de confianza de 0.25 para la poda. El número mínimo de instancias por hoja es 2. Divide 3 pliegues, con uno destinado a la poda y los otros dos para el crecimiento del árbol. Aplica el procedimiento de poda C4.5 y realiza la operación de elevación de subárbol al podar. Además, utiliza la corrección de MDL (Minimum Description Length) al encontrar divisiones en atributos numéricos.
REPTree	Recuento inicial de valores de clase igual a 0. Profundidad máxima de los árboles ilimitada. Peso total mínimo de las instancias en una hoja igual a 2. Proporción mínima de la varianza en todos los datos que debe estar presente en un nodo para que se realice la división en árboles de regresión igual a 0.001. Número de pliegues igual a 3, de los cuales uno se utiliza para la poda y el resto para hacer crecer el árbol. La semilla utilizada para la aleatorización de los datos igual a 1.
Random Tree	Número de características seleccionados al azar igual a log_2(total_de_características) + 1. Profundidad máxima de los árboles ilimitada. El peso total mínimo de las instancias en una hoja igual a 1. Sin backfitting. La semilla del número aleatorio utilizada para seleccionar características igual a 1.
Bayes Nets	Utiliza el algoritmo "SimpleEstimator" para encontrar las tablas de probabilidad condicional de la Red Bayesiana. Utiliza el algoritmo K2 para buscar estructuras de red.

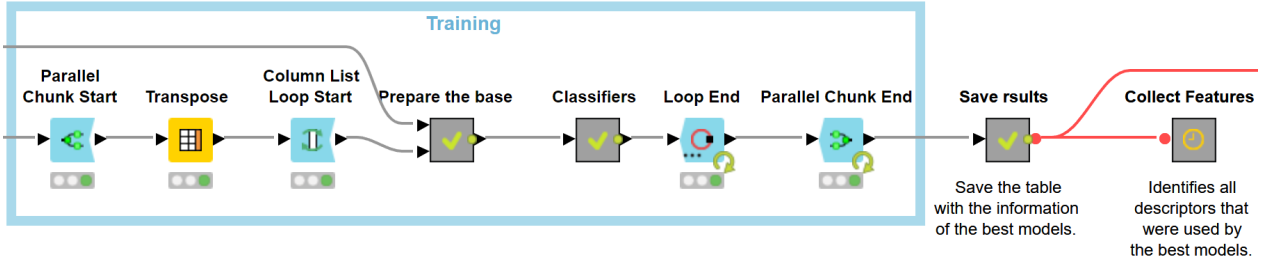

Figura 10. Vista del meta-nodo "Training Step" del flujo de trabajo de KNIME, donde se implementó la sección correspondiente a la etapa de entrenamiento.

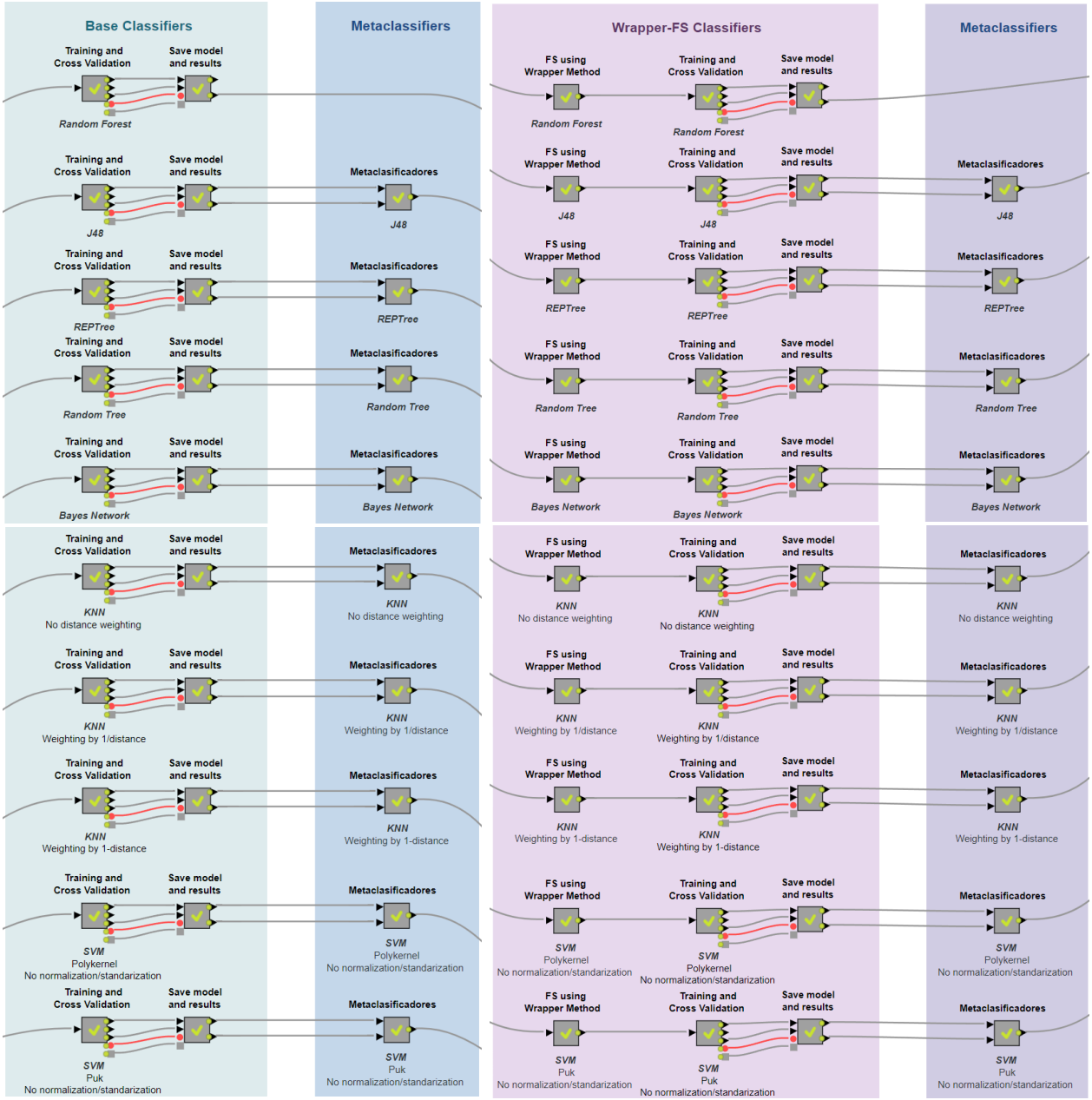

Figura 11. Vista del meta-nodo "Classifiers" dentro del meta-nodo "Training Step" del flujo de trabajo de KNIME. a) Clasificadores entrenados sin selección de características tipo *wrapper*. b) Clasificadores entrenados con selección de características tipo *wrapper*.

3.3.1.4. Entrenamiento

En la etapa de entrenamiento (ver Figuras 10 y 11), se procede a generar un total de 20 modelos a partir de cada subconjunto de características construido. Esta generación se divide en dos enfoques. El primero involucra el entrenamiento de 10 modelos utilizando los algoritmos de aprendizaje previamente

mencionados, mientras que el segundo, antes de proceder con el entrenamiento de los 10 modelos, se realiza una selección de características para el algoritmo de aprendizaje correspondiente. Esta selección se realiza a través del método *wrapper*. En este método, se utiliza un Algoritmo Genético (GA) como estrategia de búsqueda, utilizando la exactitud (accuracy) como medida de ajuste, y el algoritmo de aprendizaje que corresponda. La implementación de estos *wrapper* se lleva a cabo mediante nodos de KNIME y KNIME Weka. La metaheurística GA se aplica utilizando sus configuraciones predeterminadas: tamaño de población igual a 20, número máximo de generaciones 10, y una semilla para generar números aleatorios igual a 1.

Todos los modelos creados son evaluados mediante un procedimiento de validación cruzada de 10 pliegues. Para llevar a cabo esta evaluación, se utiliza el flujo de trabajo proporcionado por la plataforma KNIME, el cual calcula la tabla de contingencia. Después, se calculan siete métricas de rendimiento para cada modelo (consultar sección 2.4), entre ellas el MCC (coeficiente de correlación de Matthews). Los modelos con un valor de MCC de entrenamiento (MCCtrainin) mayor que el umbral $MCC\_Threshold$ (especificado como argumento de entrada) son seleccionados para construir clasificadores basados en consenso y para ser validados en los conjuntos de prueba. Para la construcción de modelos basados en consenso se usaron los enfoques Bagging, AdaBoost, Random Committee y LogitBoost. La Tabla 4 muestra qué enfoque se utiliza para cada clasificador. Para cada uno de estos modelos basados en consenso, también se les calcula el valor de MCC como se explicó anteriormente. En total, el flujo de trabajo en KNIME incorpora 33 algoritmos de clasificación. Si el argumento de entrada $MCC\_Threshold$ se establece en cero, el flujo generará un total de 1980 modelos. Esto se debe a que se generan 30 subconjuntos de características, seguidos de dos procesos distintos: uno que utiliza un enfoque *wrapper* y otro que no lo hace. Finalmente, se aplican los 33 algoritmos de clasificación disponibles.

Tabla 4. Enfoques utilizados para crear modelos basados en consenso. x indica que se utilizó el enfoque indicado.

Algoritmo	**Enfoque de clasificador basado en consenso**			
	Random Committee	Bagging	AdaBoost	LogitBoost
Random Forest				
J48			x	
REPTree	x	x	x	x
Random Tree	x	x	x	
Bayes Network		x	x	
KNN no distance weighting		x	x	x
KNN weighting 1/distance		x	x	x
KNN weighting 1-distance		x	x	x
SVM kernel polynomial		x	x	
SVM kernel puk		x	x	

3.3.1.5. Evaluación

A todos los modelos con $MCCtraining$ mayor que el $MCC\_Threshold$ pasan a la última etapa del flujo en la que se evalúan sobre los conjuntos de prueba (ver Figura 12). Además de la métrica MCC, el rendimiento de los modelos también se mide utilizando las métricas de sensibilidad, especificidad, valor-F, precisión y exactitud. Es importante destacar que los archivos CSV de prueba también se normalizan utilizando la estrategia de Min-Max. La salida generada por el flujo de trabajo es una carpeta que contiene todos los subconjuntos de características y modelos construidos. Además, se crean archivos CSV que resumen los resultados de entrenamiento y prueba. Asimismo, se crea un archivo de texto que proporciona una visión general del tiempo de ejecución.

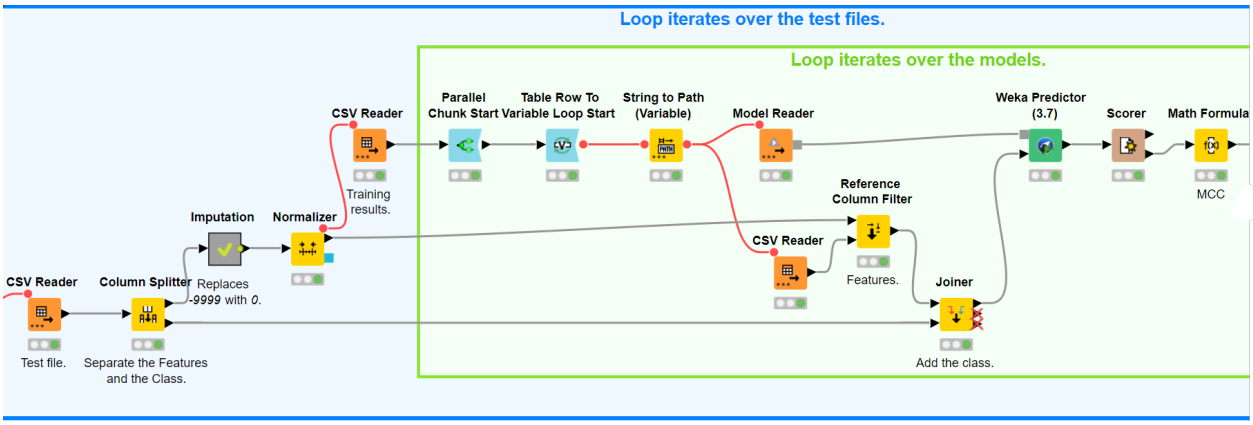

Figura 12. Vista del meta-nodo "Test" del flujo de trabajo de KNIME, donde se implementó la sección correspondiente a la etapa de evaluación.

3.3.2. Requisitos para instalar el flujo de trabajo

Para instalar el flujo de trabajo, es necesario realizar las siguientes instalaciones: KNIME v4.7.2, Java 1.8 u351 (o posterior), Compilador de C++ y Anaconda. Una vez instalados estos programas, se debe crear un entorno en Conda con el nombre "KNIME" utilizando Python 3.9. En el entorno creado, es necesario instalar los siguientes paquetes:

```
pip install numpy
pip install pillow
pip install python-javabridge
```

```
pip install python-weka-wrapper3
conda install -c anaconda joblib
conda install -c anaconda scipy
conda install -c anaconda pandas
conda install -c anaconda scikit-learn
conda install -c anaconda git
conda install -c conda-forge matplotlib
conda install -c conda-forge pyarrow
conda install -c conda-forge py4j
pip install git + https://github.com/jundongl/scikit-feature.git
```

Después, se debe abrir el programa KNIME e instalar las extensiones Weka 3.7, "KNIME python integration" y "KNIME Parallel Chunk Loop Nodes". Luego, se debe configurar en KNIME el entorno conda que será usado para ejecutar el flujo de trabajo: Menú – Preferencias – KNIME – Python - seleccionar el entorno creado anteriormente con el nombre de "KNIME". Por último, se debe importar el flujo de trabajo.

3.3.3. Ejecución desde línea de comandos

El flujo de trabajo construido recibe los siguientes argumentos de entrada. Primero se debe especificar las variables `PathTrainingFolder` y `PathTestFolder`, las cuales contienen las rutas de las carpetas donde se encuentran los archivos en formato CSV (Comma-Separated Values) con los conjuntos de entrenamiento y prueba, respectivamente. También, se debe especificar la variable `MCC_Threshold` con un valor entre 0 y 1, el cual indica el valor del coeficiente de correlación de Matthew (MCC) que se usará como umbral para seleccionar los clasificadores base en los que se aplicarán métodos de consenso. Adicionalmente, se debe especificar la variable `ClassName`, la cual contiene el nombre de la variable *endpoint* y, que debe ser la misma tanto en el archivo CSV de entrenamiento como en los archivos CSV de prueba. Finalmente, se puede especificar la variable `Subset` (opcional). En esta variable se puede proporcionar una lista separada por comas con el nombre de las características a utilizar como subconjunto inicial. Si no se especifica esta lista, se aplicará el selector de características CFS sobre el archivo CSV de entrenamiento. Para ejecutar el flujo de trabajo por línea de comando, se debe utilizar la sentencia que se muestra a continuación:

```
knime.exe -consoleLog -nosplash -nosave -reset
-application org.knime.product.KNIME_BATCH_APPLICATION
-workflowDir="workspace/Knime_project"
-workflow.variable=PathTrainingFolder,path/to/directory/csv/training/file,String
-workflow.variable=PathTestFolder,path/to/directory/csv/test/files,String
-workflow.variable=MCC_Threshold,value,double
-workflow.variable=ClassName,Target_Feature_Name,String
```

3.3.4. Ejecución desde la interfaz gráfica de usuario

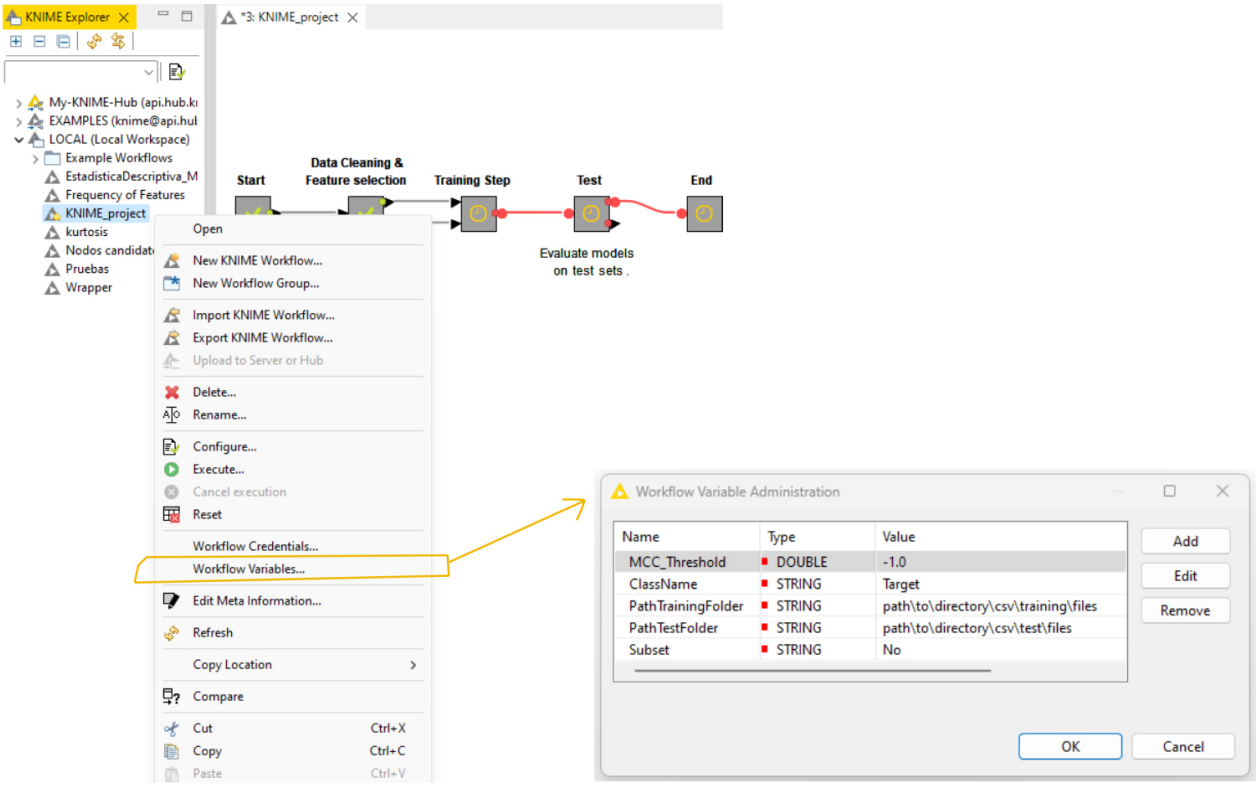

Figura 13. Pasos para modificar los valores de los argumentos de entrada del flujo de trabajo desde la GUI.

Desde la interfaz gráfica de usuario es posible ajustar los argumentos de entrada mencionados anteriormente antes de ejecutar el flujo. Para ello, basta con hacer clic derecho en el flujo "KNIME_project" desde el "KNIME Explorer", seleccionar "Workflow Variables...", y luego modificar los valores de las

variables en la ventana que se abre (ver Figura 13 b). Una vez ajustados los valores, el flujo completo puede ejecutarse simplemente con el botón "Execute all executable nodes".

3.4. Conclusiones parciales

En este capítulo se describen los cuatro conjuntos de datos de secuencias polipeptídicas que se utilizaron en este trabajo de tesis. Se presenta una caracterización de cada uno de estos, analizando los conjuntos de entrenamiento y prueba en cuanto de la cantidad de secuencias, la frecuencia de aminoácidos, la frecuencia de longitudes de las secuencias, y la diversidad. En primer lugar, se observa que los cuatro conjuntos están equilibrados (ver Tabla 2), es decir, tienen aproximadamente la misma cantidad de instancias positivas como de negativas, lo cual es apropiado para evitar que los modelos no se sesguen hacia una clase particular. En las gráficas de frecuencias de aminoácidos (Figura 4) se puede observar que hay representatividad entre el conjunto de entrenamiento y el de prueba en cuanto a la presencia de cada tipo de aminoácidos. Asimismo, al examinar las frecuencias de longitudes de las secuencias (Figura 5), se observa representatividad en general, con la excepción del conjunto de AMPs, donde no se aprecia claramente la similitud entre las distribuciones de ambos subconjuntos. En cuanto al análisis de diversidad (Figura 6), se puede notar que para AMP, AFP y AVP, el conjunto de prueba presenta una mayor diversidad que el conjunto de entrenamiento, lo cual puede ser adecuado para evaluar la capacidad de generalización de los modelos construidos. Esto significa que si los modelos demuestran un buen poder predictivo, indicaría que son capaces de generalizar bien en conjuntos de datos más diversos. Por otro lado, en el caso de ABP, el conjunto de entrenamiento es más diverso que el conjunto de prueba, lo cual podría compensar el hecho de que el conjunto de entrenamiento sea más pequeño.

Además, se abordó la implementación del flujo de trabajo en KNIME (sección 3.3). Dicho flujo de trabajo genera de manera automática hasta 1980 modelos de clasificación binaria. Incluye una etapa de limpieza de datos en la que se eliminan las características irrelevantes y redundantes. En la etapa de entrenamiento, se puede crear hasta 66 modelos al aplicar y no aplicar una selección de características con métodos basados en *wrapper*, siete algoritmos de aprendizaje automático base, y cuatro algoritmos de aprendizaje automático por consenso. En la etapa de evaluación se calculan siete métricas de desempeño sobre el conjunto de prueba para cada modelo creado. Finalmente se detallan los requisitos para instalar el flujo de trabajo (subsección 3.3.2) y se explica cómo ejecutarlo tanto desde la línea de comandos (subsección 3.3.3) como desde la interfaz gráfica de usuario (subsección 3.3.4). Este flujo de trabajo se

ha desarrollado para cumplir con los objetivos de esta tesis y para que pueda ser utilizado por especialistas en ciencias de la computación y la comunidad científica en general, incluso aquellos sin experiencia en la teoría de aprendizaje automático aquí implementada. Este flujo es útil para llevar a cabo experimentos de manera automática, asegurando la comparabilidad de los resultados y eliminando sesgos de modelado.

Capítulo 4. Resultados y discusión

En este capítulo se presentan y analizan los resultados de la investigación que se llevó a cabo con el objetivo de responder a las preguntas de investigación planteadas en esta tesis. Primero se analiza la contribución de cada incrustación ESM-2 en el desarrollo de modelos con respecto a la cantidad de modelos producidos y la cantidad de características utilizadas para crearlos. Se analiza también la contribución de cada incrustación ESM-2 en la capacidad de generalización de los modelos construidos, así como la fusión de características de diferentes incrustaciones para generar modelos con mejor capacidad de generalización. Después se compara la mayor capacidad de generalización lograda por los modelos construidos en este trabajo con la capacidad de generalización lograda por los modelos basados en aprendizaje profundo reportados en la literatura. Finalmente, se analiza la contribución de las estrategias de selección de características y clasificadores aplicados en el flujo te trabajo con respecto a la cantidad de modelos producidos.

4.1. Generación de modelos

Se ejecutó el flujo de trabajo implementado sobre las incrustaciones ESM-2 de 320, 480, 640, 1280 y 2560 dimensiones que se extrajeron sobre los conjuntos AMPs generales, ABP, AFP, AVP. Por lo tanto, el flujo de trabajo se ejecutó 20 veces, estableciendo el valor de la variable `MCC_Threshold` diferente para cada *endpoint*: 0.9 para AMP generales y ABP; 0.8 para AFP y 0.75 para AVP. Estos valores de `MCC_Threshold` se establecieron de esta manera para estudiar solo los modelos derivados de las incrustaciones ESM-2 con valores de MCC comparables o superior a los obtenidos por los modelos AniAMPpred (Sharma et al., 2021a), Deep-ABPpred (Sharma et al., 2021b), Deep-AFPpred (Sharma et al., 2021c), y Deep-AVPpred (Sharma et al., 2021d), respectivamente, es decir, con respecto a aquellos desarrollados donde se propusieron los conjuntos de datos utilizados en este trabajo.

4.2. Contribución de cada incrustación ESM-2 en el desarrollo de modelos

Primero se analizó la contribución de cada uno de las incrustaciones ESM-2 con respecto a la cantidad de modelos producidos y la cantidad de características utilizadas para crearlos. En la Figura 14 se muestran por *endpoint* la cantidad total de modelos creados a partir de las incrustaciones ESM-2. En

primer lugar, se puede observar que la cantidad de modelos con valores de MCCtraining superiores a los umbrales utilizados oscila entre 400 y 1150 modelos, excepto para el *endpoint* ABP, donde 170 y 197 son las dos cantidades más bajas de modelos construidos. Esto demuestra que el flujo de trabajo de KNIME implementado puede producir automáticamente una gran cantidad de modelos con buena varianza a partir de las incrustaciones ESM-2 utilizadas. Además, se puede observar en la Figura 14 que la mayor cantidad de modelos se obtuvo principalmente de las incrustaciones de 640, 1280 y 2560 dimensiones, que se calculan con los modelos de mayor capacidad compuestos por 30, 33 y 36 capas y que denotamos como ESM-2_t30, ESM-2_t33 y ESM-2_t36, respectivamente. Para el *endpoint* AVP, la menor cantidad de modelos se derivó de la incrustación de 2560 dimensiones, mientras que para los otros tres *endpoints*, la menor cantidad de modelos se derivó de las incrustaciones de 320 y 480 dimensiones, las cuales se calculan con los modelos de menor capacidad ESM-2_t6 (6 capas) y ESM-2_t12 (12 capas), respectivamente. Nótese que, independientemente del *endpoint*, las incrustaciones de 1280 dimensiones tuvieron el mejor comportamiento según el número de modelos obtenidos.

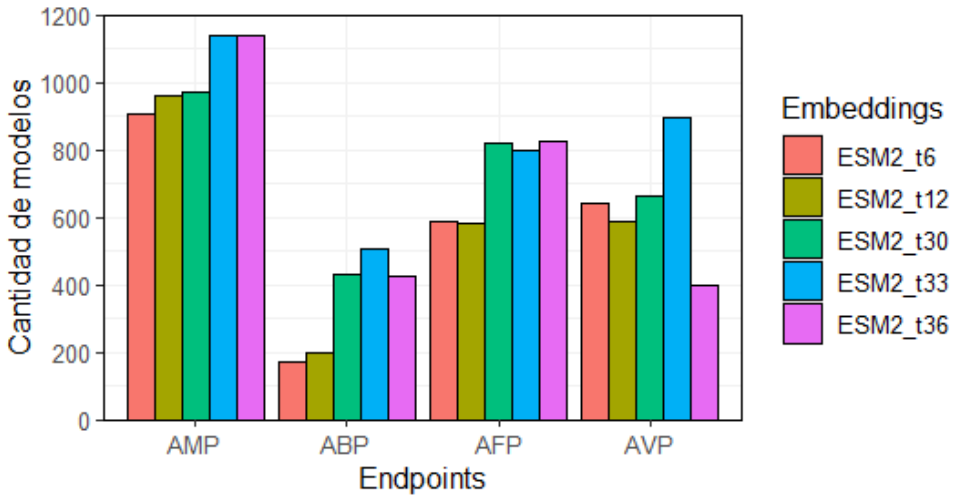

Figura 14. Número total de modelos creados a partir de cada una de las incrustaciones ESM-2 con MCCtraining mayor o igual al umbral utilizado para predecir secuencias AMP generales, ABP, AFP y AVP.

Por otro lado, la Figura 15 presenta la distribución de la cantidad de características utilizadas en los modelos construidos por *endpoint* e incrustación ESM-2. Una incrustación es un vector de números donde cada posición representa una característica específica. Teniendo esto en cuenta, se puede señalar que cuanto mayor es la dimensión de las incrustaciones utilizadas como conjunto inicial de características, mayor es el número de características utilizadas en los modelos. Esto implica que, si bien se pueden construir más y mejores modelos a partir de las incrustaciones ESM-2 de mayor dimensión, dichos modelos

son más complejos en cuanto a la cantidad de características que los creados a partir de las incrustaciones ESM-2 más pequeñas. Los modelos creados a partir de las incrustaciones de 2560 dimensiones son los únicos que utilizan más de 200 características. Con respecto al número de características, también se concluyó que el 90.4 % del total de modelos utilizan menos de 150 características, mientras que el 5.68 % del total de modelos utilizan entre 150 y 200 características. Esto indica que solo una parte de las incrustaciones fue valiosa para modelar los *endpoints* antimicrobianos considerados en este trabajo.

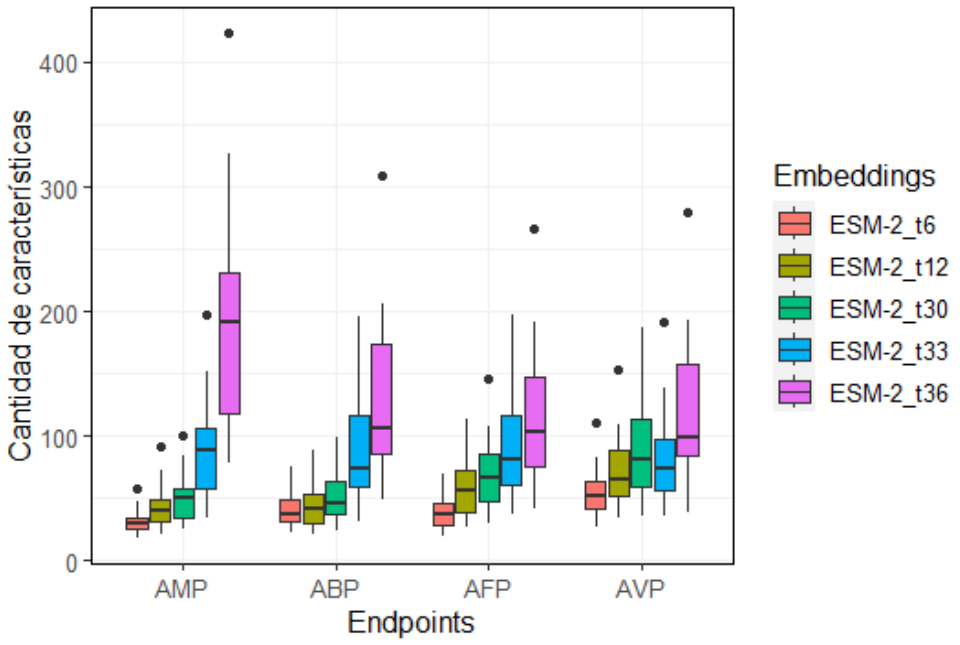

Figura 15. Diagrama de caja de la cantidad de características utilizadas en los modelos construidos.

En ese sentido, por un lado, la Figura 16 muestra la frecuencia de uso de las características pertenecientes a las incrustaciones ESM-2 en los modelos desarrollados. Estas frecuencias se calcularon a partir de los subconjuntos de características creados para entrenar los diferentes clasificadores. Así, un mismo subconjunto usado varias veces para entrenar diferentes clasificadores se consideró una vez. Finalmente, se sumó la frecuencia de uso de cada característica para todos los *endpoints*. Como un resultado, se puede observar en la Figura 16 que existen varios espacios vacíos (gaps), lo que demuestra que varias características en las incrustaciones ESM-2 nunca se seleccionaron para construir un modelo, ya sea porque no tenían información relevante, o porque su información fue redundante con respecto a las características seleccionadas para predecir los *endpoints* considerados. De hecho, al analizar todas las incrustaciones ESM-2 juntas, se puede concluir que 3156 (59.8 %) de un total de 5280 características nunca se usaron, 1014 (19.2 %) características se usaron entre 1 y 100 veces, 596 (11.3 %) características

se usaron entre 100 y 200 veces, y 514 (9.73 %) características se usaron más de (o igual a) 200 veces.

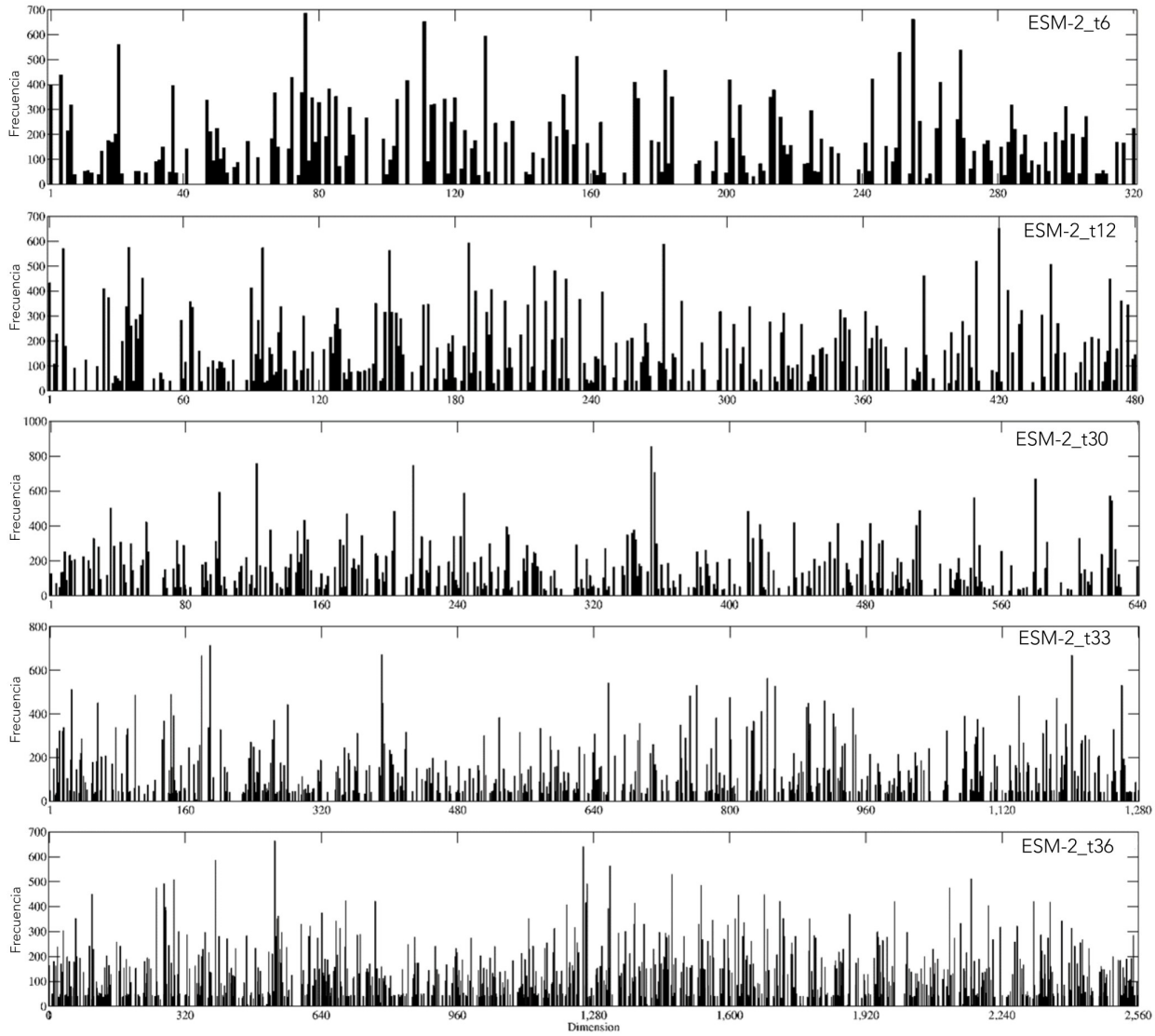

Figura 16. Frecuencia de uso de las características pertenecientes a las incrustaciones ESM-2 en los modelos desarrollados.

Por otro lado, la Tabla 5 muestra por *endpoint* y modelo ESM-2 el número total de características utilizadas al menos una vez para modelar cada *endpoint*. Como se puede observar, el número total de características utilizadas oscila entre 59 y 428 características, y el porcentaje con respecto a la dimensión de incrustación correspondiente varia entre un 10.47 % y un 34.38 %. También se puede observar que siempre se usó una cantidad similar de características de la incrustación de 1280 dimensiones para modelar los cuatro *endpoints*, mientras que hubo diferencias mayores en la cantidad de características utilizadas desde las otras incrustaciones ESM-2. Finalmente, al analizar el número total de características utilizadas considerando todos los *endpoints*, se puede notar que más del 49 % de las características pertenecientes

a las tres incrustaciones más pequeñas fueron usadas en la modelación, mientras que se usaron menos del 40 % de las características pertenecientes a las dos incrustaciones más grandes. Específicamente, se puede notar que el 43.44 %, 47.5 %, 50.47 %, 60.94 % y 65.86 % de la información química contenida en las incrustaciones ESM-2 de 320, 480, 640, 1280 y 2560 dimensiones, respectivamente, resultaron inútiles (o redundantes) para las tareas de modelación realizadas.

En sentido general, se puede decir que las incrustaciones de 1280 dimensiones calculadas con el modelo ESM-2_t33 son aquellas a partir de las cuales se produce un mayor número de modelos utilizando un número similar de características para todos los *endpoints*, seguidos por las incrustaciones de 2560 dimensiones calculadas con el modelo ESM-2_t36. Además de estos resultados, se puede sugerir que solo una parte de las incrustaciones parecen ser útiles para tareas downstream, lo cual puede evitar problemas de dimensionalidad cuando se utilizan pequeños conjuntos de datos químicos. Pero esta hipótesis necesita estudios más profundos, que están fuera del alcance de este trabajo.

Tabla 5. Número de características pertenecientes a cada incrustación ESM-2 que se usaron al menos una vez para construir los modelos para predecir AMP generales, ABP, AFP y AVP. [a]: entre paréntesis se muestra el porcentaje respecto a la dimensión de la incrustación correspondiente.

Incrustaciones (Dimensión)	*Endpoints*				**Total**[a]
	AMP[a]	**ABP**[a]	**AFP**[a]	**AVP**[a]	
ESM-2_t6 (320)	59 (18.43 %)	76 (23.75 %)	69 (21.56 %)	110 (34.38 %)	181 (56.56 %)
ESM-2_t12 (480)	91 (18.96 %)	89 (18.54 %)	113 (23.54 %)	154 (32.83 %)	252 (52.50 %)
ESM-2_t30 (640)	101 (15.78 %)	99 (15.47 %)	146 (22.81 %)	188 (29.38 %)	317 (49.53 %)
ESM-2_t33 (1,280)	201 (15.70 %)	198 (15.47 %)	200 (15.63 %)	193 (15.08 %)	500 (39.06 %)
ESM-2_t36 (2,560)	428 (16.72 %)	314 (12.27 %)	268 (10.47 %)	292 (11.41 %)	874 (34.14 %)

4.3. Contribución de cada incrustación ESM-2 en la capacidad de generalización de los modelos construidos

En esta sección, se analiza la capacidad de generalización en los conjuntos de prueba de los modelos creados automáticamente por el flujo de trabajo implementado en KNIME. Este análisis se lleva a cabo por incrustaciones ESM-2 para evaluar cuál de ellas generó los modelos con la mejor capacidad de generalización. La Tabla 6 muestra cuántos modelos por incrustación ESM-2 presentaron un valor de MCCtest en un intervalo específico. Como se puede observar, casi todos los modelos en la predicción de AMPs generales lograron valores de MCCtest superiores a 0.8, siendo los modelos derivados de las incrustaciones más grandes los que en general obtuvieron valores de MCCtest superiores a 0.9. En la predicción de ABP y AFP, los modelos obtuvieron principalmente valores de MCCtest entre 0.7 y 0.9.

Para estos dos *endpoints*, la mayor cantidad de modelos con valores de MCCtest superiores a 0.8 se obtuvo a partir de las incrustaciones de 2560 dimensiones. Para el *endpoint* AFP, la menor cantidad de modelos con valores de MCCtest superiores a 0.8 se construyeron a partir de las incrustaciones de 1280 dimensiones en comparación con la cantidad de modelos obtenidos a partir de las incrustaciones de 640 y 2560 dimensiones. Por último, se puede observar que casi todos los modelos en la predicción de AVPs alcanzaron valores de MCCtest entre 0.7 y 0.8. Teniendo en cuenta estos resultados obtenidos en los conjuntos de prueba y que todos los modelos analizados alcanzaron valores de MCCtraining superiores a 0.9, 0.9, 0.8 y 0.75 para predecir AMPs generales, ABPs, AFPs y AVPs, respectivamente, se puede concluir que los modelos analizados no presentan sobreajuste ni subajuste. Por lo tanto, el flujo de trabajo es capaz de producir automáticamente modelos con valores de varianza y bias adecuados.

Tabla 6. Análisis por intervalos de valores MCCtest y por incrustación ESM-2 del rendimiento alcanzado en los conjuntos de prueba por los mejores modelos retenidos según sus valores MCCtraining. Ningún modelo presentó un valor de MCCtest en el intervalo $0.5 \leq$ MCC <0.6.

Incrustaciones	**Total de Modelos**	**Intervalos de MCCtest**				
		$0.4 \leq$ MCC <0.5	$0.6 \leq$ MCC <0.7	$0.7 \leq$ MCC <0.8	$0.8 \leq$ MCC <0.9	$0.9 \leq$ MCC ≤ 1
AMP						
ESM-2_t6	908	0	0	23	842	43
ESM-2_t12	960	0	0	0	731	229
ESM-2_t30	972	0	0	0	112	860
ESM-2_t33	1140	0	0	0	113	1027
ESM-2_t36	1142	0	0	0	99	1043
ABP						
ESM-2_t6	170	0	9	161	0	0
ESM-2_t12	197	0	2	130	65	0
ESM-2_t30	432	0	0	196	236	0
ESM-2_t33	508	0	2	145	361	0
ESM-2_t36	427	1	1	40	385	0
AFP						
ESM-2_t6	587	0	31	449	107	0
ESM-2_t12	582	0	22	506	54	0
ESM-2_t30	820	0	0	431	389	0
ESM-2_t33	797	2	5	678	112	0
ESM-2_t36	825	0	0	264	561	0
AVP						
ESM-2_t6	643	0	55	588	0	0
ESM-2_t12	586	0	37	549	0	0
ESM-2_t30	664	0	1	663	0	0
ESM-2_t33	899	0	4	895	0	0
ESM-2_t36	402	0	0	402	0	0

Debido a que se construyeron diferentes cantidades de modelos a partir de cada incrustación ESM-2, se seleccionaron los mejores 900, 150, 500 y 400 modelos de acuerdo con los valores de MCCtest logrados en la predicción de AMP generales, ABP, AFP, y AVP, respectivamente. En la Figura 17 se resume la distribución de los valores de MCCtest logrados por los mejores modelos creados por *endpoint* e incrustación ESM-2. Como un resultado, se puede notar que los valores de MCCtest alcanzados por los

modelos creados a partir de las incrustaciones de 640, 1280 y 2560 dimensiones presentan las mejores distribuciones ya que el valor de su mediana está muy por encima de la mediana de las distribuciones de valores de MCCtest correspondientes a los modelos construidos a partir de las dos incrustaciones más pequeñas. Sin embargo, este comportamiento no está presente en la predicción de AVP porque la distribución de los valores de MCCtest correspondientes a las incrustaciones de 640 dimensiones presenta un comportamiento similar a las dos incrustaciones más pequeñas.

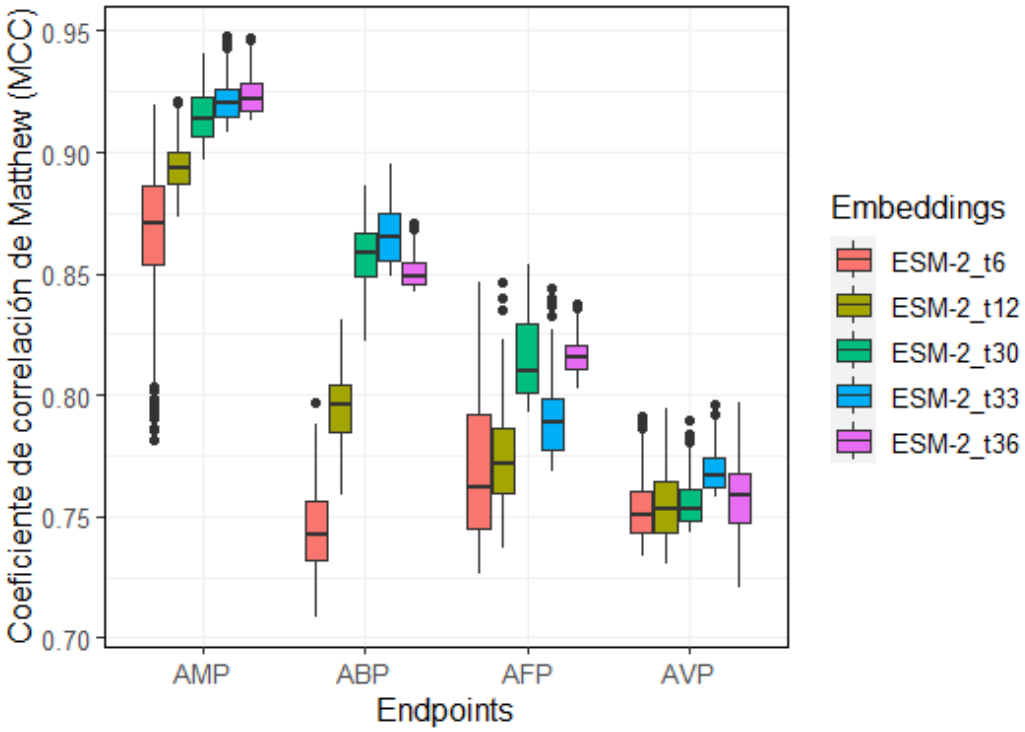

Figura 17. Diagrama de caja de los valores de MCCtest obtenidos en los conjuntos de prueba por los modelos construidos.

Además, se puede observar en la Figura 17 que en la predicción de AMP generales los valores más altos de MCCtest fueron obtenidos por los modelos derivados de las incrustaciones de 2560 dimensiones, seguidos por los modelos derivados de las incrustaciones de 1280 dimensiones. Pero los modelos derivados de las incrustaciones de 2560 dimensiones utilizaron una cantidad mucho mayor de características para predecir el *endpoint* mencionado anteriormente, como se muestra en la Figura 15. En la predicción de ABP, los valores más altos de MCCtest los obtuvieron los modelos creados a partir de la incrustación de 1280 dimensiones. De hecho, el 50 % de los valores de MCCtest logrados por esos modelos fueron mejores que aproximadamente el 75 % de los valores de MCCtest logrados por los modelos con el segundo mejor rendimiento, que se derivaron de la incrustación de 640 dimensiones. Sin embargo, en la predicción

de AFP y AVP, aunque las mejores distribuciones de los valores de MCCtest pertenecen a las tras incrustaciones más grandes, se puede notar que modelos con valores de MCCtest tan altos como los obtenidos a partir de esas tres incrustaciones también se obtuvieron desde las incrustaciones de 320 y 480 dimensiones, siendo los modelos más sencillos en cuanto al número de características utilizadas. Nótese también que en la predicción de AFPs, los modelos derivados de la incrustación de 1280 dimensiones presentaron un desempeño más bajo con respecto a los modelos construidos a partir de las incrustaciones de 640 y 2560 dimensiones, aunque en la predicción de AVP, sucede lo contrario con respecto a estas tres incrustaciones de mayor dimensión, ya que generalmente los modelos derivados de la incrustación de 1280 dimensiones fueron mejores.

En general, teniendo en cuenta la relación entre la capacidad de generalización y el número de características, se puede concluir que se crean los mejores modelos para predecir AMPs generales, ABPs y AVPs a partir de la incrustación de 1280 dimensiones, mientras que para la predicción de AFPs, los mejores modelos se desarrollan a partir de la incrustación de 640 dimensiones. Se realizó un estudio estadístico basado en la estimación bayesiana (Benavoli et al., 2017) para respaldar la conclusión anterior. Este análisis se llevó a cabo entre pares de incrustaciones para determinar la probabilidad de generar mejores modelos a partir de una incrustación en comparación con otra según los valores de MCCtest obtenidos por los modelos construidos a partir de cada una de ellas. La Figura 18 muestra las coordenadas baricéntricas de las distribuciones de probabilidad obtenidas mediante muestreo de Monte Carlo. Se distinguen tres áreas en cada una de esas coordenadas: una región superior, denominada rope, que representa equivalencia práctica, y una región inferior izquierda (o derecha) que corresponde al caso en el que los modelos construidos a partir de la incrustación representada por esa área tienen una mayor probabilidad de ser mejores que los modelos construidos a partir de la incrustación representada por la otra región inferior. Se utilizó un valor de rope igual a 0.01.

En la Figura 18 se muestran los resultados entre las tres incrustaciones más grandes. Como se puede observar, no hay diferencia al generar modelos a partir de las tres incrustaciones más grandes para la predicción de AMPs generales (primera fila), lo que justifica el uso de la incrustación de 1280 dimensiones ya que a partir de este se pueden construir modelos con los valores MCCtest más altos con un número adecuado de características. Para la predicción de ABPs (segunda fila) y AVPs (cuarta fila), se puede observar que las probabilidades más altas son también para los modelos creados a partir de la incrustación de 1280 dimensiones. En la predicción de ABPs, también se observa que los modelos construidos a partir de la incrustación de 640 dimensiones presentan una mayor probabilidad de ser mejores que los modelos desarrollados a partir de la incrustación de 2560 dimensiones, mientras que en la predicción de AVPs, no

hay diferencia entre estas dos incrustaciones mencionadas anteriormente. Finalmente, se puede observar que en la predicción de AFPs, los modelos construidos a partir de las incrustaciones de 640 y 2560 dimensiones tienen una mayor probabilidad de lograr valores MCCtest mejores que los modelos derivados de la incrustación de 1280 dimensiones, mientras que no hay diferencia entre los modelos creados a partir de las otras dos.

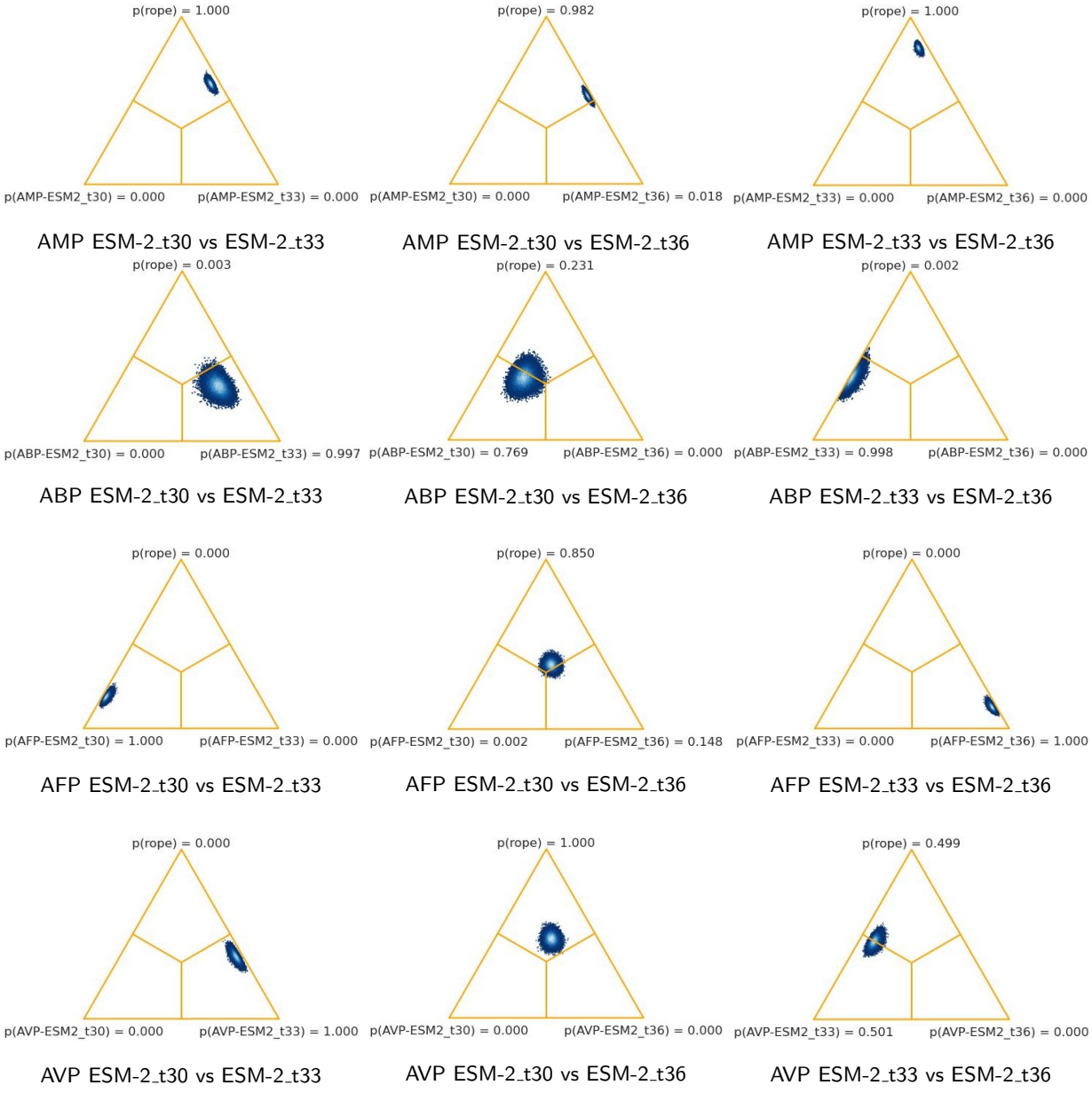

Figura 18. Coordenadas baricéntricas correspondientes al análisis estadístico basado en estimación bayesiana para estudiar a partir de qué incrustación ESM-2 hay mayor probabilidad de obtener los mejores modelos.

Tabla 7. Desempeño de los modelos no basados en fusión top-1 (y top-50) y los modelos basados en características fusionadas top-1 (y top-50). Los valores de desviación estándar se muestran entre paréntesis.

Modelo	Características	MCCtraining	SNtest	SPtest	ACCtest	MCCtest
General-AMP						
Mejor modelo sin fusionar características	147	0.951	0.9669	0.9811	0.9741	0.9482
Mejores 50 modelos sin fusionar características	109 (31)	.9465 (.003)	.9614 (.006)	.9826 (.006)	.9722 (.001)	.9447 (.001)
Mejor modelo con fusión de características	82	0.9418	0.9646	0.9819	0.9734	0.9469
Mejores 50 modelos con fusión de características	77 (25)	.9434 (.003)	.9629 (.005)	.9790 (.005)	.9711 (.001)	.9423 (.002)
ABP						
Mejor modelo sin fusionar características	72	0.9242	0.9457	0.9514	0.9491	0.895
Mejores 50 modelos sin fusionar características	80 (31)	.9200 (.007)	.9417 (.015)	.9421 (.010)	.9419 (.003)	.8808 (.005)
Mejor modelo con fusión de características	42	0.919	0.9435	0.9602	0.9533	0.9035
Mejores 50 modelos con fusión de características	59 (20)	.9219 (.008)	.9459 (.015)	.9435 (.014)	.9445 (.004)	.8863 (.007)
AFP						
Mejor modelo sin fusionar características	64	0.8444	0.897	0.955	0.926	0.8535
Mejores 50 modelos sin fusionar características	70 (26)	.8319 (.009)	.8963 (.011)	.9445 (.010)	.9204 (.002)	.8420 (.003)
Mejor modelo con fusión de características	93	0.8517	0.9181	0.9449	0.9315	0.8633
Mejores 50 modelos con fusión de características	77 (19)	.8382 (.008)	.9039 (.009)	.9453 (.008)	.9246 (.002)	.8500 (.004)
AVP						
Mejor modelo sin fusionar características	82	0.7911	0.8594	0.9352	0.8973	0.7969
Mejores 50 modelos sin fusionar características	104 (41)	.7838 (.012)	.8910 (.018)	.8957 (.019)	.8933 (.002)	.7872 (.004)
Mejor modelo con fusión de características	125	0.783	0.89	0.9156	0.9028	0.8059
Mejores 50 modelos con fusión de características	102 (24)	.7777 (.008)	.8927 (.008)	.9077 (.009)	.9002 (.002)	.8006 (.003)

Aunque las incrustaciones de 640 y 1280 dimensiones son las más apropiadas según los resultados estadísticos explicados anteriormente, es importante destacar que a partir de las otras incrustaciones, principalmente para la predicción de AFPs y AVPs, se construyeron modelos igual de buenos que los creados a partir de las dos incrustaciones mencionados anteriormente. Esto sugiere que el desarrollo de modelos basados en la fusión de diferentes incrustaciones podría ser una alternativa adecuada para generar mejores modelos, como se estudia a continuación.

4.4. Fusión de incrustaciones ESM-2 para mejorar la capacidad de generalización

En esta sección se evalúa si la fusión de características pertenecientes a diferentes incrustaciones ESM-2 es una estrategia adecuada para desarrollar mejores modelos QSAR. Para ello, primero seleccionamos por *endpoint* los mejores 50 modelos sin importar desde qué incrustación se crearon. De los 50 mejores modelos no basados en fusión, se seleccionó el mejor modelo por incrustación para fusionar las características incluidas en ellos. Así, se fusionaron las características incluidas en (1) el mejor modelo construido a partir de cada uno de las incrustaciones de 640 y 1280 dimensiones para predecir AMPs generales y ABPs, respectivamente; (2) el mejor modelo construido a partir de cada uno de las incrustaciones de 320, 480, 640 y 1280 dimensiones para predecir AFPs; y (3) el mejor modelo construido a partir de las cinco incrustaciones para predecir AVPs. El flujo de trabajo de KNIME se aplicó al conjunto fusionado correspondiente a cada *endpoint*, y se seleccionaron los mejores 50 modelos por *endpoint* en función de sus valores de MCCtest de la misma manera que se eligieron los mejores modelos no basados en fusión.

La Tabla 7 muestra los valores promedio de MCCtraining, SNtest, SPtest, ACCtest, MCCtest y del número de características utilizadas correspondientes a los mejores 50 modelos construidos al fusionar o no fusionar características de diferentes incrustaciones. También se muestran los valores de estas métricas correspondientes al modelo con el valor de MCCtest más alto. Por un lado, en lo que respecta al número de características, se puede observar que se construyeron modelos más simples para predecir AMPs generales y ABPs a partir de los conjuntos de características fusionados. De hecho, observe que el mejor modelo para cada uno de ellos usó 82 y 42 características, respectivamente, lo que representa un 44.22 % y un 41.67 % menos de características en comparación con el número de características utilizadas por los dos mejores modelos no basados en fusión. Así mismo, el número de características promedio utilizadas por los modelos basados en fusión de características es menor que el promedio de las características utilizadas por los modelos no basados en fusión. En la predicción de AFPs se puede notar que en

su mayoría se crearon modelos más complejos a partir de los conjuntos de características fusionadas, utilizando un 10 % más de características en promedio. Mientras que, en la predicción de AVPs, aunque el mejor modelo basado en características fusionadas utilizó 43 características (52.44 %) más que el mejor modelo no basado en fusión, se puede observar que el número promedio de características utilizadas por los modelos basados en características fusionadas fue ligeramente inferior al número promedio de características utilizadas por los modelos no basados en fusión.

Por otro lado, en lo que respecta a las capacidades de generalización (MCCtest), se puede observar que el rendimiento de los modelos basados en características fusionadas es ligeramente inferior al rendimiento de los modelos no basados en fusión para predecir AMPs generales. De hecho, en términos absolutos, la diferencia entre los mejores rendimientos y los rendimientos promedio de esos modelos es igual a 0.0013 y 0.0024, respectivamente. Pero a pesar del rendimiento ligeramente inferior, se puede concluir que los mejores modelos para predecir AMPs generales fueron los creados al fusionar características de diferentes incrustaciones porque utilizaron menos características y realizaron predicciones bastante similares a los modelos no basados en fusión. Esta conclusión está en correspondencia con la navaja de Ockham (Danek et al., 2022), que establece que entre modelos con rendimientos similares, se debe seleccionar el más simple para su uso prospectivo. Conclusiones similares también se pueden establecer para los modelos basados en características fusionadas para predecir ABPs. Pero en este caso, esos modelos lograron mejores valores de MCCtest que los modelos no basados en fusión y fueron más simples en cuanto al número de características. Observe que el mejor modelo basado en características fusionadas es el único que logró un valor de MCCtest superior a 0.9.

En la predicción de AFPs y AVPs, se puede observar que los modelos basados en características fusionadas lograron mejores valores de MCCtest pero utilizando un mayor número de características. Específicamente, el rendimiento del mejor modelo basado en características fusionadas fue mejor que el rendimiento del mejor modelo no basado en fusión en un 1.15 % (0.0098 en términos absolutos) y un 1.13 % (0.009) en la predicción de AFPs y AVPs, respectivamente, mientras que los 50 mejores modelos basados en características fusionadas fueron en promedio mejores que los 50 mejores modelos no basados en fusión en un 0.95 % (0.008) y un 1.7 % (0.0134) en la predicción de AFPs y AVPs, respectivamente. Sin embargo, estos resultados no indican que no se puedan construir modelos más simples y mejores a partir de los conjuntos de características fusionadas para predecir AFPs y AVPs.

De hecho, como se puede ver en la Tabla SI5 disponible como material suplementario en `https://drive.google.com/drive/folders/1WZ24Y4klj5xnrWjM6IjnkiVu9MBjODLU`, el sexto y séptimo mejor modelo basado en características fusionadas para predecir AFPs utilizaron 41 y 60 características

y lograron un valor de MCCtest igual a 0.8541 y 0.8537, respectivamente, mientras que el duodécimo, decimoctavo y trigésimo segundo mejor modelo basado en características fusionadas para predecir AVPs utilizaron 83, 63 y 56 características y lograron un valor de MCCtest igual a 0.8037, 0.8012 y 0.7984, respectivamente, siendo estos valores similares o mejores que los obtenidos por el mejor modelo no basado en fusión para predecir AFPs (64 características, MCCtest = 0.8535) y AVPs (82 características, MCCtest = 0.7969), respectivamente. En general, se puede concluir que la fusión de características de diferentes incrustaciones contribuye a obtener mejores modelos para predecir péptidos antimicrobianos que el uso exclusivo de una sola incrustación ESM-2 específica.

4.5. Análisis comparativo con respecto a arquitecturas de aprendizaje profundo

En esta sección, se compara la mejor capacidad de generalización lograda por los modelos construidos en este trabajo con la capacidad de generalización lograda por los modelos basados en aprendizaje profundo (deep learning) que construyeron sus autores a partir de los conjuntos de datos utilizados en este trabajo. Para ello, se consideró el mejor modelo por *endpoint* creado tanto a partir de las incrustaciones individuales como a partir de los conjuntos de características fusionadas (consultar Tabla 7). Los modelos basados en aprendizaje profundo analizados son AniAMPpred (Sharma et al., 2021a), Deep-ABPpred (Sharma et al., 2021b), Deep-AFPpred (Sharma et al., 2021c) y Deep-AVPpred (Sharma et al., 2021d), los cuales se construyeron para predecir AMPs generales, ABPs, AFP y AVPs, respectivamente. El modelo AniAMPpred utiliza una red neuronal convolucional unidimensional (1D-CNN). El modelo Deep-ABPpred se basa en una red neuronal recurrente del tipo de memoria a corto y largo plazo bidireccional (Bi-LSTM), mientras que Deep-AFPpred utiliza una 1D-CNN seguida de una Bi-LSTM para la clasificación. Por último, el modelo Deep-AVPpred se basa en una red neuronal convolucional (CNN).

La Figura 19 muestra una comparación entre los modelos mencionados anteriormente con respecto a las métricas SNtest, SPtest, ACCtest y MCCtest. En primer lugar, se puede observar que en la predicción de AMPs generales, tanto el mejor modelo no basado en fusión que se construyó a partir de las incrustaciones de 1280 dimensiones, como el mejor modelo basado en características fusionadas, lograron un mejor rendimiento que AniAMPpred según las métricas SNtest, ACCtest y MCCtest, mientras que AniAMPpred fue ligeramente mejor que el primero en cuanto a la métrica SPtest. AniAMPpred utilizó 200 características aprendidas a través de una arquitectura 1D-CNN para entrenar a cinco clasificadores individuales y tomar decisiones utilizando un enfoque de votación mayoritaria, mientras que el mejor modelo basado

en características fusionadas y el mejor modelo no basado en fusión utilizaron 82 y 147 características ESM-2, respectivamente. Es decir, estos modelos basados en incrustaciones ESM-2 utilizaron un 59 % y un 26.5 % menos de características para realizar mejores predicciones que AniAMPpred.

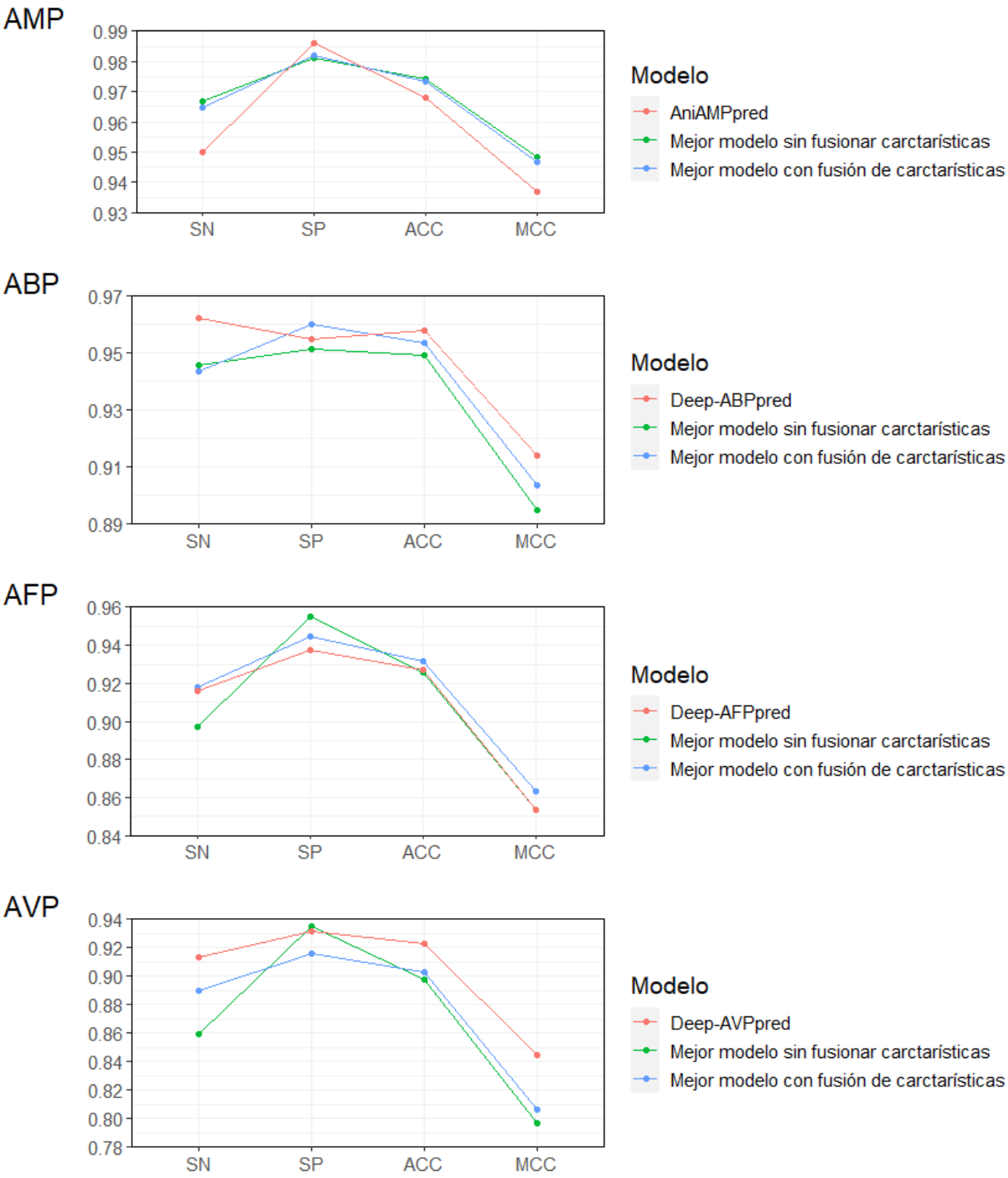

Figura 19. Comparaciones entre arquitecturas de aprendizaje profundo de última generación y los mejores modelos creados en este trabajo para predecir AMP generales, ABP, AFP y AVP.

En cuanto a las predicciones de ABPs, Deep-ABPpred fue mejor que el mejor modelo basado en características fusionadas (y el mejor modelo no basado en fusión) en un 1.96 % (1.72 %), 0.49 % (0.94 %) y 1.16 % (2.12 %) en cuanto a las métricas SNtest, ACCtest y MCCtest, respectivamente, pero fue ligeramente inferior al mejor modelo basado en características fusionadas en un 0.54 % en cuanto a la métrica SPtest. El mejor modelo basado en características fusionadas utilizó 42 características, mientras que el mejor modelo no basado en fusión utilizó 72 características, y ambos fueron entrenados con el clasificador SVM, que es más simple que la arquitectura Bi-LSTM utilizada por Deep-ABPpred. Por lo tanto, según la navaja de Ockham, el modelo basado en características fusionadas sería el más adecuado para estudios prospectivos, porque además de ser más simple, logró un rendimiento bastante similar al de Deep-ABPpred con respecto al MCCtest. Sin embargo, es importante destacar que en estudios de comparación entre modelos basados aprendizaje tradicional y modelos basados en aprendizaje profundo para la clasificación de AMP, Deep-ABPpred no superó al modelo de la literatura S1-ABP, consultar Tabla 3F en (García-Jacas et al., 2022b), S1-ABP utilizó 88 características calculadas con los programas ProtDCal e iLearn para entrenar un clasificador basado en concenso LogitBoost utilizando Random Forest como clasificador base. Es probable que esta sea la razón por la que nuestro flujo de trabajo no generó un mejor modelo que Deep-ABPpred en este trabajo, ya que LogitBoost y Random Forest no se pueden usar juntos porque la versión de Weka incorporada en la plataforma KNIME no admite tal combinación.

En cuanto a las predicciones de AFPs, se puede observar que el mejor modelo basado en características fusionadas tuvo un mejor rendimiento que Deep-AFPpred (y el mejor modelo no basado en fusión) en un 0.23 % (2.35 %), 0.51 % (0.59 %) y 1.09 % (1.15 %) con respecto a las métricas SNtest, ACCtest y MCCtest, respectivamente, mientras que el mejor modelo no basado en fusión obtuvo el valor más alto de SPtest, superando al mejor modelo basado en características fusionadas y a Deep-AFPpred en un 1.07 % y 1.81 %, respectivamente.

En lo que respecta a las predicciones de AVPs, se puede observar que Deep-AVPpred obtuvo un mejor rendimiento que los otros dos modelos estudiados, superando al mejor modelo basado en características fusionadas y al mejor modelo no basado en fusión en un 4.85 % y 6.04 % según la métrica MCCtest, respectivamente. Este resultado también se obtuvo en los estudios de comparación realizados entre modelos basados aprendizaje tradicional y modelos basados en aprendizaje profundo de García-Jacas et al. (2022b), donde Deep-AVPpred siempre fue mejor que todos los modelos basados aprendizaje tradicional creados con características calculadas con los programas ProtDCal e iLearn. Sin embargo, es importante mencionar que los autores de Deep-AVPpred utilizaron un conjunto de validación para la

optimización de hiperparámetros en la construcción del modelo, mientras que no se realizó optimización de hiperparámetros para construir los modelos en el flujo de trabajo implementado en KNIME. Esto podría ser la razón por la cual los modelos creados para predecir AVPs no tuvieron un mejor rendimiento que Deep-AVPpred, ya que los otros modelos basados en aprendizaje profundo analizados realizaron la optimización de hiperparámetros utilizando un pliegue del conjunto de entrenamiento y, en estos casos, se lograron resultados comparables o superiores a los modelos aquí construidos.

Los resultados explicados anteriormente confirman los resultados obtenidos estudios realizados por García-Jacas et al. (2022b,a), donde se demostró que los modelos basados en aprendizaje tradicional logran un rendimiento comparable o superior a los modelos basados en aprendizaje profundo en la predicción de AMPs cuando se llevan a cabo estudios metodológicamente fundamentados. Varios trabajos que justifican que los modelos basados en aprendizaje profundo mejoran la clasificación de AMPs, a menudo por estrechas diferencias de rendimiento, presentan sesgos de modelado relacionados con la calidad y diversidad de las características de proteínas basadas en secuencias, la falta de enfoques de selección de características y el uso deficiente de clasificadores basados en consenso. Para evitar algunos de estos sesgos en trabajos futuros, así como para proponer nuevos modelos basados en aprendizaje tradicional, se implementó el flujo de trabajo de KNIME abarcando varios de los aspectos metodológicos a considerar al construir modelos basados en aprendizaje tradicional a partir de datos tabulares, lo que permite obtener en un tiempo razonable cientos (o miles) de modelos basados en aprendizaje tradicional. Por ejemplo, el tiempo requerido en una estación de trabajo no dedicada para construir los mejores 900, 150, 500 y 400 modelos para predecir general-AMPs, ABPs, AFPs y AVPs a partir de la incrustación de 2560 dimensiones fue aproximadamente igual a 137, 6, 13 y 17 horas, respectivamente, requiriendo menos esfuerzo y recursos computacionales que cuando se construyen modelos basados en aprendizaje profundo. A continuación, se presentan los principales resultados obtenidos con los enfoques de modelado implementados en el flujo de trabajo de KNIME.

4.6. Contribución de las estrategias de selección de características y clasificadores aplicados

En esta sección, se evalúa la contribución de cada una de las estrategias de selección de características y algoritmos de aprendizaje aplicados con respecto al número de modelos construidos y sus capacidades de generalización. En este análisis no se incluyó los resultados obtenidos al fusionar las incrustaciones, sino solo los obtenidos a partir de cada una de ellas por separado. Los resultados se obtuvieron considerando

todos los *endpoints* así como todas las incrustaciones juntas. Como resultado, la Figura 20 muestra el número de modelos creados a partir de cada estrategia de selección de características. Como se puede observar, todas las estrategias de selección contribuyeron a generar una gran cantidad de modelos, siendo la estrategia la basada en el índice Gini la única con menos de 250 modelos construidos. La estrategia que más contribuyó fue la unión de los siete selectores individuales, seguida de las estrategias de fusionar el subconjunto de características obtenido con el método CFS con las características seleccionadas con los selectores de tipo ranking MI, Relief-F y Gini-index en ese orden.

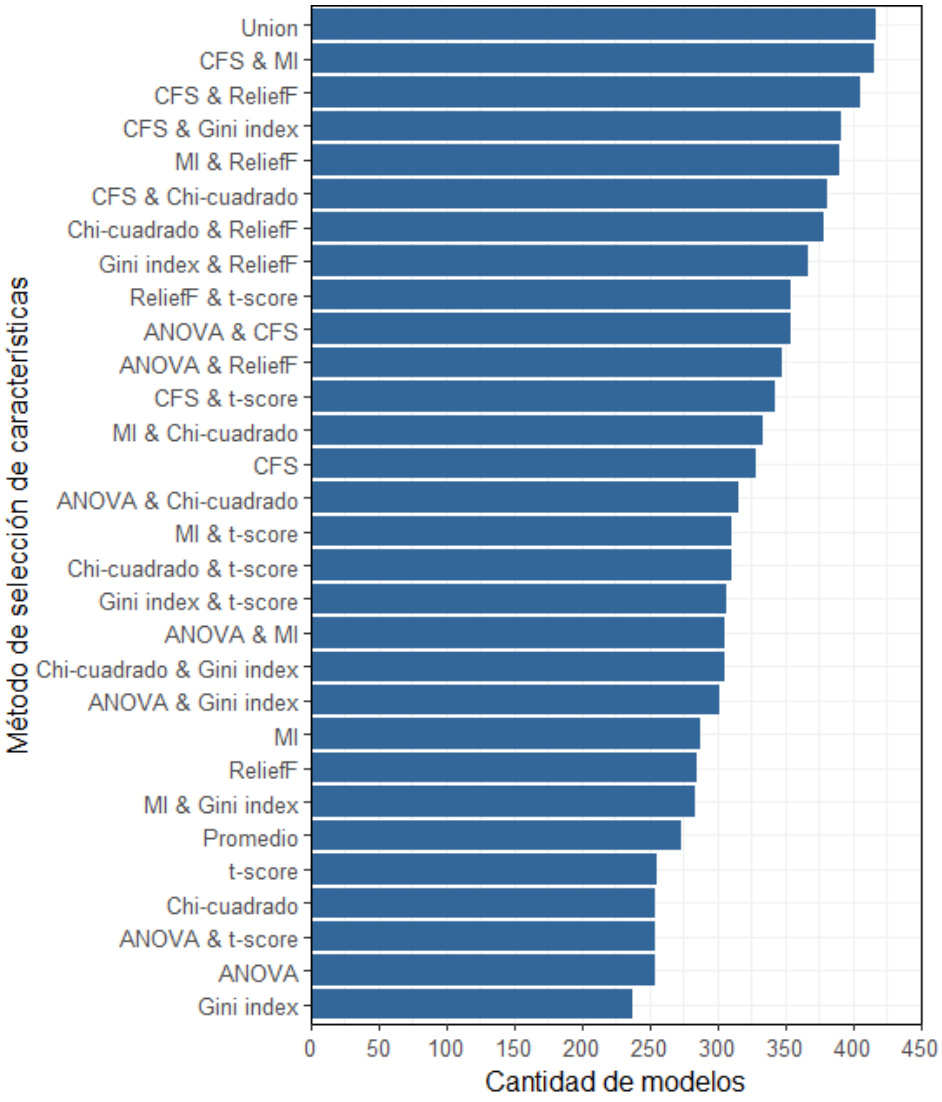

Figura 20. Número de modelos creados a partir de cada estrategia de selección de características implementada en el flujo de trabajo de KNIME. En esta gráfica se están contabilizando los modelos creados a partir de las cinco incrustaciones y los cuatro *endpoints*.

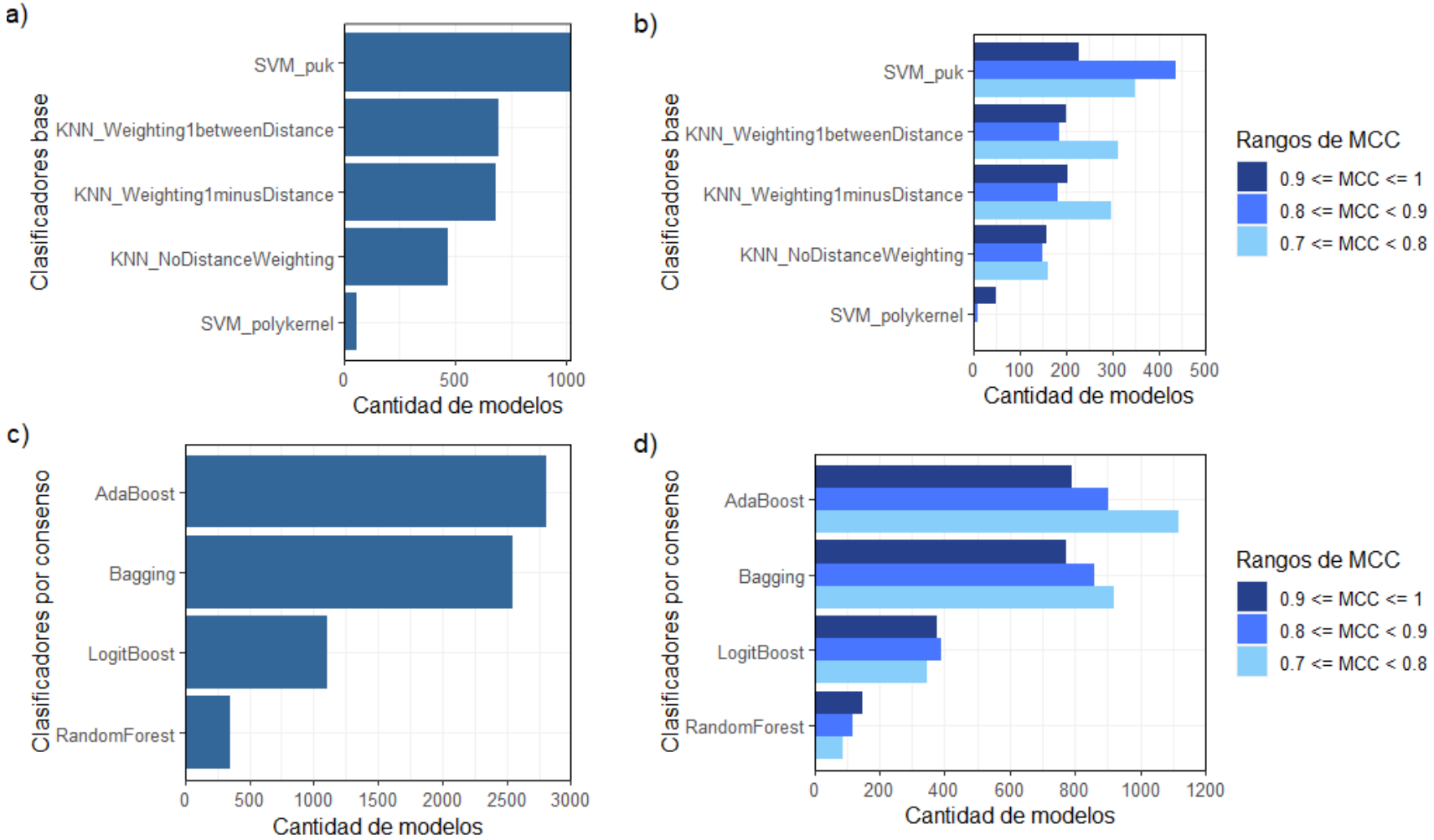

Figura 21. Gráficas de barras que representan el número de modelos producidos a partir de los mejores clasificadores individuales (a y b) y de consenso (c y d), tanto globalmente como por rango de valores de prueba de MCC. En estas gráficas se están contabilizando los modelos creados a partir de las cinco incrustaciones y los cuatro *endpoints*.

De hecho, el número de modelos creados al fusionar las características obtenidas con CFS y MI es similar al número de modelos construidos a partir de la unión de los siete selectores individuales, siendo estos dos conjuntos de características, junto con la fusión entre CFS y Relief-F, los únicos de los cuales se obtuvieron más de 400 modelos. Las otras estrategias de selección permitieron obtener entre 237 y 400 modelos. Además, al analizar las 10 estrategias de selección principales, se puede observar que el subconjunto de características seleccionadas con CFS, así como con Relief-F, se utilizaron 5 veces cada uno, los subconjuntos basados en los selectores MI, Chi-cuadrado e índice Gini se utilizaron dos veces cada uno, y los subconjuntos basados en los selectores ANOVA y t-score se utilizaron una vez cada uno. Finalmente, observe que la estrategia de selección basada en la fusión por promedio del ranking generó una baja cantidad (278) de modelos con respecto a 24 estrategias, lo que sugiere que el promedio no es una forma efectiva de combinar diferentes selectores de tipo ranking individuales.

Además, en la Figura 21 se muestra el número total de modelos creados a partir de las cinco incrustaciones y los cuatro *endpoints* distinguiendo el tipo de clasificador utilizado, así como cuántos de ellos lograron un rendimiento en un rango específico de valores de MCCtest. Por un lado, se puede observar en la Figura 21a que solo 2 de los 6 algoritmos de clasificación (RF se considera como clasificador por consenso) utilizados en el flujo de trabajo de KNIME contribuyeron al desarrollo de buenos modelos. Específicamente, los algoritmos k-NN y SVM con sus tres esquemas de ponderación y dos kernels, respectivamente, fueron los únicos útiles, siendo el clasificador SVM con núcleo Puk el que más contribuyó con más de 1000 modelos. También se puede observar que se obtuvieron más de 1500 modelos al analizar el número total de modelos creados con los tres clasificadores k-NN (uno por esquema de ponderación), lo que es una cantidad mayor que el número de modelos construidos con los dos clasificadores basados en SVM juntos. Sin embargo, la Figura 21b revela que los modelos basados en el clasificador SVM con núcleo Puk lograron las capacidades de generalización más altas. De hecho, se obtuvieron más de 600 modelos con valores de MCCtest entre 0.8 y 1 a partir de dicho clasificador, mientras que se obtuvieron entre 300 y 400 modelos en el rango mencionado anteriormente a partir de cada uno de los tres clasificadores k-NN aplicados.

Por otro lado, la Figura 21c muestra el número de modelos desarrollados a partir de cada clasificador basado en consenso aplicado en el flujo de trabajo de KNIME. Como se puede observar, el mayor número de modelos se derivó de AdaBoost y Bagging (sin incluir RF) con 2810 y 2552, respectivamente. Pero este resultado se debe a que AdaBoost se aplicó para todos los clasificadores base, mientras que Bagging se aplicó a 8 de los 9 clasificadores base (consultar la Tabla 4). El clasificador basado en consenso LogitBoost solo se aplicó para 4 de los 8 clasificadores base considerados. Observe que el clasificador

RF fue el peor de todos, incluso fue peor que 4 de los 5 clasificadores débiles representados en la Figura 21a en cuanto al número de modelos. Según las capacidades de generalización logradas por los modelos construidos a partir de cada uno de estos clasificadores basados en consenso, se puede observar en la Figura 21d que AdaBoost y Bagging tienen un rendimiento similar, ambos presentando 1691 y 1634 modelos con valores de MCCtest entre 0.8 y 1, respectivamente. Mientras que LogitBoost y RF generaron 766 y 267 modelos en el rango mencionado anteriormente.

4.7. Conclusiones parciales

En este capítulo se mostraron los resultados obtenidos correspondientes a la generación de modelos de clasificación binaria mediante la ejecución del flujo de trabajo implementado sobre las incrustaciones ESM-2 de 320, 480, 640, 1280 y 2560 dimensiones que se extrajeron sobre los conjuntos AMPs generales, ABP, AFP, AVP. Del total de modelos generados, el 90.4 % utilizan menos de 150 características. Además, se analizó que entre un 43 % y un 65 % de la información química contenida en las incrustaciones ESM-2 resultó ser inútil (o redundante) para las tareas de modelación realizadas. Esto incentiva a realizar más experimentos en trabajos futuros para ver si se puede identificar qué parte de las incrustaciones se pudiera prescindir en tareas downstream para tratar de evitar probemas de dimensionalidad cuando se utilizan pequeños conjuntos de datos.

Luego, se observa que de todos los modelos generados para predecir AMPs generales, ABPs y AVPs, respectivamente, más de la mitad tienen valores de MCCtest arriba de 0.9, 0.9, 0.7, respectivamente, mientras que para AFPs una tercera parte de los modelos tienen valores de MCCtest arriba de 0.8. Lo cual es una gran cantidad de modelos que no presentan sobreajuste ni subajuste. Aunque las tres incrustaciones de mayor dimensión utilizados son los que generan una mayor cantidad de estos modelos con alto valor de MCCtest, las dos incrustaciones de dimensión menor también generaron modelos igual de buenos. Lo cual sustentó la alternativa de desarrollar modelos a partir de la fusión de características de diferentes incrustaciones ESM-2.

Al comparar los modelos generados a partir de usar las incrustaciones por separado y los generados a partir de la fusión de incrustaciones, se observa que, a partir de la fusión de de características de diferentes incrustaciones ESM-2 se pudieron generar modelos con un rendimiento comparable a superior a los obtenidos a partir de las incrustaciones por separado. También se generaron modelos incluso más sencillos (en cuanto a la cantidad de características utilizadas) con un rendimiento comparable.

Finalmente se compararon el mejor modelo obtenido desde cada uno de los dos enfoques (fusionando incrustaciones o no fusionándolas) con los modelos basados en aprendizaje profundo reportados en la literatura que construyeron sus autores a partir de los conjuntos de datos utilizados en este trabajo. Donde se observó que los mejores modelos construidos logran un rendimiento comparable o superior a los basados en aprendizaje profundo.

Capítulo 5. Conclusiones y trabajo futuro

A partir de los resultados obtenidos se llegó a las siguientes conclusiones y recomendaciones para trabajo futuro.

5.1. Conclusiones

Se implementó un flujo de trabajo que puede generar automáticamente hasta 1980 modelos de clasificación binaria basados en aprendizaje tradicional. Estos modelos pueden exhibir una buena relación entre el sesgo y la varianza, además de contar con un número adecuado de características. El flujo de trabajo incorpora diversas técnicas de selección de características, algoritmos de clasificación y métricas de desempeño, además de una fase de limpieza de datos. Este flujo de trabajo está disponible en `https://github.com/cicese-biocom/classification-QSAR-bioKom`.

Se demostró que aumentar la capacidad de los modelos ESM-2 no implica una mejora en el rendimiento de modelos para predecir AMPs. De hecho, de los modelos ESM-2 estudiados en esta investigación, se concluye que los más apropiados para extraer características y mejorar la precisión en las predicciones de AMPs son los modelos ESM-2_t30 y ESM-2_t33. Además, se demostró que fusionar características de diferentes incrustaciones ESM-2 es una estrategia efectiva para construir mejores modelos QSAR en comparación con el uso exclusivo de características derivadas de un modelo ESM-2 específico. Por último, se construyeron modelos más simples con un rendimiento comparable o superior a modelos basados en aprendizaje profundo reportados en la literatura.

5.2. Trabajo futuro

Como trabajo futuro, se sugiere agregar un componente al flujo de trabajo para eliminar modelos redundantes en lo que respecta a las predicciones que realizan. Además, se propone incorporar un componente con técnicas de Inteligencia Artificial Explicable (IAE) para mejorar la interpretabilidad de los modelos. Incluir estos dos elementos en el flujo de trabajo fortalecería aún más su utilidad y capacidad para abordar desafíos en la predicción de AMPs.

Literatura citada

Acheampong, F. A., Nunoo-Mensah, H., & Chen, W. (2021). Transformer models for text-based emotion detection: a review of bert-based approaches. *Artificial Intelligence Review*, 54, 5789 – 5829. https://doi.org/10.1007/s10462-021-09958-2.

Aguilera-Mendoza, L., Marrero-Ponce, Y., Beltran, J. A., Tellez-Ibarra, R., Guillen-Ramirez, H. A., & Brizuela, C. A. (2019). Graph-based data integration from bioactive peptide databases of pharmaceutical interest: toward an organized collection enabling visual network analysis. *Bioinformatics*. https://doi.org/10.1093/bioinformatics/btz260.

Aguilera-Mendoza, L., Marrero-Ponce, Y., Tellez-Ibarra, R., Llorente-Quesada, M. T., Salgado, J., Barigye, S. J., & Liu, J. (2015). Overlap and diversity in antimicrobial peptide databases: compiling a non-redundant set of sequences. *Bioinformatics*, 31(15), 2553–9. https://doi.org/10.1093/bioinformatics/btv180.

Agüero-Chapin, G., Antunes, A., Mora, J. R., Pérez, N., Contreras-Torres, E., Valdes-Martini, J. R., Martinez-Rios, F., Zambrano, C. H., & Marrero-Ponce, Y. (2023). Complex networks analyses of antibiofilm peptides: An emerging tool for next-generation antimicrobialsrsquo; discovery. *Antibiotics*, 12(4). https://doi.org/10.3390/antibiotics12040747.

Agüero-Chapin, G., Galpert-Cañizares, D., Domínguez-Pérez, D., Marrero-Ponce, Y., Pérez-Machado, G., Teijeira, M., & Antunes, A. (2022). Emerging computational approaches for antimicrobial peptide discovery. *Antibiotics*, 11(7). https://doi.org/10.3390/antibiotics11070936.

Aha, D. & Kibler, D. (1991). Instance-based learning algorithms. *Machine Learning*, 6, 37–66. https://doi.org/10.1007/BF00153759.

Alberts, B., Bray, D., Hopkin, K., Johnson, A. D., Lewis, J., Raff, M., Roberts, K., & Walter, P. (2015). *Essential cell biology*. Garland Science. https://books.google.com.mx/books/about/Essential_Cell_Biology.html?id=Cg4WAgAAQBAJ&redir_esc=y.

Altschuh, D., Lesk, A. M., Bloomer, A. C., & Klug, A. (1987). Correlation of co-ordinated amino acid substitutions with function in viruses related to tobacco mosaic virus. *Journal of molecular biology*, 193(4), 693–707. https://doi.org/10.1016/0022-2836(87)90352-4.

Altschuh, D., Vernet, T., Berti, P. J., Moras, D., & Nagai, K. (1988). Coordinated amino acid changes in homologous protein families. *Protein engineering*, 2(3), 193–9. https://doi.org/10.1093/protein/2.3.193.

Beltran, J. A., Rio, G. D., & Brizuela, C. A. (2020). An automatic representation of peptides for effective antimicrobial activity classification. *Computational and Structural Biotechnology Journal*, 18, 455 – 463. https://doi.org/10.1016/j.csbj.2020.02.002.

Benavoli, A., Corani, G., Demsar, J., & Zaffalon, M. (2017). Time for a change: a tutorial for comparing multiple classifiers through bayesian analysis. *The Journal of Machine Learning Research*, 18(1), 2653–2688. https://jmlr.org/papers/v18/16-305.html.

Bishop, C. M. & Nasrabadi, N. M. (2006). *Pattern recognition and machine learning*. Springer. https://link.springer.com/book/9780387310732.

Bolón-Canedo, V. & Alonso-Betanzos, A. (2019). Ensembles for feature selection: A review and future trends. *Inf. Fusion*, 52, 1–12. https://doi.org/10.1016/j.inffus.2018.11.008.

Branden, C. I. & Tooze, J. (2012). *Introduction to protein structure*. Garland Science. https://books.google.com.mx/books/about/Essential_Cell_Biology.html?id=Cg4WAgAAQBAJ&redir_esc=y.

Breiman, L. (2001). Random forests. *Machine Learning*, 45, 5–32. https://doi.org/10.1023/A:1010933404324.

Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A., Grobler, J., Layton, R., VanderPlas, J., Joly, A., Holt, B., & Varoquaux, G. (2013). API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 108–122. https://doi.org/10.48550/arXiv.1309.0238.

Centers for Disease Control and Prevention (2019). Antibiotic resistance threats in the united states, 2019. [Archivo PDF] https://www.cdc.gov/drugresistance/pdf/threats-report/2019-ar-threats-report-508.pdf.

Centers for Disease Control and Prevention (2022). Antimicrobial resistance. [Hoja informativa en línea] Consultado el 24 Abril 2023 en https://www.cdc.gov/drugresistance/about.html.

Chicco, D. & Jurman, G. (2020). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21. https://doi.org/10.1186/s12864-019-6413-7.

Consonni, V. & Todeschini, R. (2009). *Molecular descriptors for chemoinformatics*, (2a ed.). John Wiley & Sons. https://doi.org/10.1002/9783527628766.

Cruz, E., Cozman, F. G., Souza, W., & Takiuti, A. D. (2021). The impact of teenage pregnancy on school dropout in brazil: a bayesian network approach. *BMC Public Health*, 21. https://doi.org/10.1186/s12889-021-11878-3.

Danek, A., Rainer, T. S., & Sala, S. D. (2022). Ockham's razor, not a barber's weapon but a writer's tool. *Brain : a journal of neurology*. https://doi.org/10.1093/brain/awac159.

Darwiche, A. (2010). Bayesian networks. *Communications of the ACM*, 53, 80 – 90. https://doi.org/10.1145/1859204.1859227.

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805. https://doi.org/10.48550/arXiv.1810.04805.

Divyashree, M., Mani, M. K., Reddy, D., Kumavath, R., Ghosh, P., Azevedo, V., & Barh, D. (2020). Clinical applications of antimicrobial peptides (amps): where do we stand now? *Protein and peptide letters*. https://doi.org/10.2174/0929866526666190925152957.

Dogan, T. (2018). Uniprot: a worldwide hub of protein knowledge. *Nucleic Acids Research*, 47, D506 – D515. https://doi.org/10.1093/nar/gky1049.

European Centre for Disease Prevention and Control and World Health Organization (2023). *Antimicrobial resistance surveillance in Europe 2023–2021 data*. European Centre for Disease Prevention and Control and World Health Organization. Regional Office for Europe. [Archivo PDF] https://www.who.int/europe/publications/i/item/9789289058537.

Fabian, B., Edlich, T., Gaspar, H., Segler, M. H. S., Meyers, J., Fiscato, M., & Ahmed, M. (2020). Molecular representation learning with language models and domain-relevant auxiliary tasks. *ArXiv*, abs/2011.13230. https://doi.org/10.48550/arXiv.2011.13230.

Floridi, L. & Chiriatti, M. (2020). Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30, 681–694. https://doi.org/10.1007/s11023-020-09548-1.

Freund, Y. & Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on International Conference on Machine Learning*, ICML'96, 148–156, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. http://dl.acm.org/citation.cfm?id=3091696.3091715.

Friedman, J., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors). *The Annals of Statistics*, 28(2), 337 – 407. https://doi.org/10.1214/aos/1016218223.

Gabere, M. N. & Noble, W. S. (2017). Empirical comparison of web based antimicrobial peptide prediction tools. *Bioinformatics*, 33, 1921–1929. https://doi.org/10.1093/bioinformatics/btx081.

García, G. C., Toledano, J. P.-P., de Haro García, A., & García-Pedrajas, N. (2019). Filter feature selectors in the development of binary qsar models. *SAR and QSAR in Environmental Research*, 30, 313 – 345. https://doi.org/10.1080/1062936X.2019.1588160.

García-Jacas, C. R., García-González, L. A., Martínez-Rios, F., Tapia-Contreras, I. P., & Brizuela, C. A. (2022a). Handcrafted versus non-handcrafted (self-supervised) features for the classification of antimicrobial peptides: complementary or redundant? *Briefings in bioinformatics*. https://doi.org/10.1093/bib/bbac428.

García-Jacas, C. R., Marrero-Ponce, Y., Vivas-Reyes, R., Suárez-Lezcano, J., Martínez-Rios, F., Terán, J. E., & Aguilera-Mendoza, L. (2020). Distributed and multicore qubils [U+2010] midas software v2.0: Computing chiral, fuzzy, weighted and truncated geometrical molecular descriptors based on tensor algebra. *Journal of Computational Chemistry*, 41, 1209 – 1227. https://doi.org/10.1002/jcc.26167.

García-Jacas, C. R., Pinacho-Castellanos, S. A., García-González, L. A., & Brizuela, C. A. (2022b). Do deep learning models make a difference in the identification of antimicrobial peptides? *Briefings in bioinformatics*. https://doi.org/10.1093/bib/bbac094.

Hall, M. A. & Smith, L. A. (1998). Practical feature subset selection for machine learning. In *Conference Proceedings*. [Archivo PDF] https://researchcommons.waikato.ac.nz/handle/10289/1512.

Huan, Y., Kong, Q., Mou, H., & Yi, H. (2020). Antimicrobial peptides: Classification, design, application and research progress in multiple fields. *Frontiers in Microbiology*, 11. https://doi.org/10.3389/fmicb.2020.582779.

Hughes, A. L. & Yeager, M. (1997). Coordinated amino acid changes in the evolution of mammalian defensins. *Journal of Molecular Evolution*, 44, 675–682. https://doi.org/10.1007/pl00006191.

Ilyas, I. F. & Chu, X. (2019). *Data cleaning*. Morgan & Claypool. https://www.morganclaypoolpublishers.com/catalog_Orig/product_info.php?products_id=1424.

Irwin, R., Dimitriadis, S., He, J., & Bjerrum, E. J. (2021). Chemformer: a pre-trained transformer for computational chemistry. *Machine Learning: Science and Technology*, 3. https://doi.org/10.1088/2632-2153/ac3ffb.

Kim, S., Chen, J., Cheng, T., Gindulyte, A., He, J., He, S., Li, Q., Shoemaker, B. A., Thiessen, P. A., Yu, B., Zaslavsky, L. Y., Zhang, J., & Bolton, E. E. (2018). Pubchem 2019 update: improved access to chemical data. *Nucleic Acids Research*, 47, D1102 – D1109. https://doi.org/10.1093/nar/gky1033.

Kokoska, S. & Zwillinger, D. (2000). *CRC standard probability and statistics tables and formulae*. Crc Press. https://www.routledge.com/CRC-Standard-Probability-and-Statistics-Tables-and-Formulae/Zwillinger-Kokoska/p/book/9780367399078.

Kumar, P., Kizhakkedathu, J. N., & Straus, S. K. (2018). Antimicrobial peptides: Diversity, mechanism of action and strategies to improve the activity and biocompatibility in vivo. *Biomolecules*, 8. https://doi.org/10.3390/biom8010004.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., rahman Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Annual Meeting of the Association for Computational Linguistics*. https://doi.org/10.48550/arXiv.1910.13461.

Li, J., Cheng, K., Wang, S., Morstatter, F., Trevino, R. P., Tang, J., & Liu, H. (2017). Feature selection: A data perspective. *ACM computing surveys (CSUR)*, 50(6), 1–45. https://doi.org/10.48550/arXiv.1601.07996.

Lin, Z., Akin, H., Rao, R., Hie, B., Zhu, Z., Lu, W., Smetanin, N., Verkuil, R., Kabeli, O., Shmueli, Y., dos Santos Costa, A., Fazel-Zarandi, M., Sercu, T., Candido, S., & Rives, A. (2023). Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637), 1123–1130. https://doi.org/10.1126/science.ade2574.

Liu, H. & Setiono, R. (1995). Chi2: feature selection and discretization of numeric attributes. *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*, 388–391. https://doi.org/10.1109/TAI.1995.479783.

Maimon, O. & Rokach, L. (2005). *Data mining and knowledge discovery handbook*. Springer. https://doi.org/10.1007/b107408.

Marrero-Ponce, Y., Terán, J. E., Contreras-Torres, E., García-Jacas, C. R., Pérez-Castillo, Y., Cubillán, N., Pérez-Giménez, F., & Valdés-Martiní, J. R. (2020). Lego-based generalized set of two linear algebraic 3d bio-macro-molecular descriptors: Theory and validation by qsars. *Journal of theoretical biology*. https://doi.org/10.1016/j.jtbi.2019.110039.

Mistry, J., Chuguransky, S. R., Williams, L., Qureshi, M., Salazar, G. A., Sonnhammer, E. L. L., Tosatto, S. C. E., Paladin, L., Raj, S., Richardson, L. J., Finn, R. D., & Bateman, A. (2020). Pfam: The protein families database in 2021. *Nucleic Acids Research*, 49, D412 – D419. https://doi.org/10.1093/nar/gkaa913.

Mitchell, M. (1998). *An introduction to genetic algorithms*. MIT press. https://mitpress.mit.edu/9780262631853/an-introduction-to-genetic-algorithms/.

Pes, B. (2019). Ensemble feature selection for high-dimensional data: a stability analysis across multiple domains. *Neural Computing and Applications*, 32, 5951–5973. https://doi.org/10.1007/s00521-019-04082-3.

Pinacho-Castellanos, S. A., García-Jacas, C. R., Gilson, M. K., & Brizuela, C. A. (2021). Alignment-free antimicrobial peptide predictors: Improving performance by a thorough analysis of the largest available data set. *Journal of chemical information and modeling*. https://doi.org/10.1021/acs.jcim.1c00251.

Pirtskhalava, M., Gabrielian, A. E., Cruz, P., Griggs, H. L., Squires, R. B., Hurt, D. E., Grigolava, M., Chubinidze, M., Gogoladze, G., Vishnepolsky, B., Alekseev, V., Rosenthal, A., & Tartakovsky, M. (2015). Dbaasp v.2: an enhanced database of structure and antimicrobial/cytotoxic activity of natural

and synthetic peptides. *Nucleic Acids Research*, 44, D1104 – D1112. https://doi.org/10.1093/nar/gkw243.

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc. https://doi.org/10.1016/C2009-0-27846-9.

Radford, A. & Narasimhan, K. (2018). Improving language understanding by generative pre-training. [Archivo PDF] https://api.semanticscholar.org/CorpusID:49313245.

Rives, A., Goyal, S., Meier, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., & Fergus, R. (2019). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *Proceedings of the National Academy of Sciences of the United States of America*, 118. https://doi.org/10.1101/622803.

Romero-Molina, S., Ruiz-Blanco, Y. B., Green, J. R., & Sánchez-García, E. (2019). Protdcal suite: A web server for the numerical codification and functional analysis of proteins. *Protein Science*, 28, 1734 – 1743. https://doi.org/10.1002/pro.3673.

Ross, B. C. (2014). Mutual information between discrete and continuous data sets. *PLoS ONE*, 9. https://doi.org/10.1371/journal.pone.0087357.

Schoch, C. L., Ciufo, S., Domrachev, M., Hotton, C. L., Kannan, S., Khovanskaya, R., Leipe, D. D., McVeigh, R., O'Neill, K., Robbertse, B., Sharma, S., Soussov, V., Sullivan, J. P., Sun, L., Turner, S., & Karsch-Mizrachi, I. (2020). Ncbi taxonomy: a comprehensive update on curation, resources and tools. *Database : the journal of biological databases and curation*. https://doi.org/10.1093/database/baaa062.

Sewald, N. & Jakubke, H.-D. (2015). *Peptides: chemistry and biology*, (2a ed.). John Wiley & Sons. https://www.wiley.com/en-mx/Peptides:+Chemistry+and+Biology,+2nd+Edition-p-9783527318674.

Shannon, C. E. (1948). A mathematical theory of communication. *Bell Syst. Tech. J.*, 27, 623–656. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x.

Sharma, R., Shrivastava, S., Singh, S. K., Kumar, A., Saxena, S., & Singh, R. K. (2021a). Aniamppred: artificial intelligence guided discovery of novel antimicrobial peptides in animal kingdom. *Briefings in bioinformatics*. https://doi.org/10.1093/bib/bbab242.

Sharma, R., Shrivastava, S., Singh, S. K., Kumar, A., Saxena, S., & Singh, R. K. (2021b). Deep-abppred: identifying antibacterial peptides in protein sequences using bidirectional lstm with word2vec. *Briefings in bioinformatics*. https://doi.org/10.1093/bib/bbab242.

Sharma, R., Shrivastava, S., Singh, S. K., Kumar, A., Saxena, S., & Singh, R. K. (2021c). Deep-afppred: identifying novel antifungal peptides using pretrained embeddings from seq2vec with 1dcnn-bilstm. *Briefings in bioinformatics*. https://doi.org/10.1093/bib/bbab422.

Sharma, R., Shrivastava, S., Singh, S. K., Kumar, A., Singh, A. K., & Saxena, S. (2021d). Deep-avppred: Artificial intelligence driven discovery of peptide drugs for viral infections. *IEEE Journal of Biomedical and Health Informatics*, 26, 5067–5074. https://doi.org/10.1109/JBHI.2021.3130825.

Singh, S., Chaudhary, K., Dhanda, S. K., Bhalla, S., Usmani, S. S., Gautam, A., Tuknait, A., Agrawal, P., Mathur, D., & Raghava, G. P. (2015). Satpdb: a database of structurally annotated therapeutic peptides. *Nucleic Acids Research*, 44, D1119 – D1126. https://doi.org/10.1093/nar/gkv1114.

Thakur, N., Qureshi, A., & Kumar, M. (2012). Avppred: collection and prediction of highly effective antiviral peptides. *Nucleic Acids Research*, 40, W199 – W204. https://doi.org/10.1093/nar/gks450.

Théolier, J., Fliss, I., Jean, J., & Hammami, R. (2014). Milkamp: a comprehensive database of antimicrobial peptides of dairy origin. *Dairy Science & Technology*, 94, 181–193. https://doi.org/10.1007/s13594-013-0153-2.

Todeschini, R. & Consonni, V. (2002). *Handbook of Molecular Descriptors*. Wiley-VCH. https://doi.org/10.1002/9783527613106.

Todeschini, R., Consonni, V., Mauri, A., & Pavan, M. (2003). Mobydigs: software for regression and classification models by genetic algorithms. *Data Handling in Science and Technology*, 23, 141–167. https://doi.org/10.1016/S0922-3487(03)23005-7.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023a). Llama: Open and efficient foundation language models. *ArXiv*, abs/2302.13971. https://doi.org/10.48550/arXiv.2302.13971.

Touvron, H., Martin, L., Stone, K. R., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D. M., Blecher, L., Ferrer, C. C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., ... , & Scialom, T. (2023b). Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288. https://doi.org/10.48550/arXiv.2307.09288.

Tropsha, A. (2010). Best practices for qsar model development, validation, and exploitation. *Molecular Informatics*, 29. https://doi.org/10.1002/minf.201000061.

Urias, R. W. P., Barigye, S. J., Marrero-Ponce, Y., García-Jacas, C. R., Valdés-Martiní, J. R., & Pérez-Giménez, F. (2015). Imman: free software for information theory-based chemometric analysis. *Molecular Diversity*, 19, 305–319. https://doi.org/10.1007/s11030-014-9565-z.

Üstün, B., Melssen, W. J., & Buydens, L. M. C. (2006). Facilitating the application of support vector regression by using a universal pearson vii function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81, 29–40. https://doi.org/10.1016/j.chemolab.2005.09.003.

Valdés-Martiní, J. R., Marrero-Ponce, Y., García-Jacas, C. R., Martínez-Mayorga, K., Barigye, S. J., d'Almeida, Y. S. V., Pham-The, H., Pérez-Giménez, F., & Morell, C. (2017). Qubils-mas, open source multi-platform software for atom- and bond-based topological (2d) and chiral (2.5d) algebraic molecular descriptors computations. *Journal of Cheminformatics*, 9. https://doi.org/10.1186/s13321-017-0211-5.

Voet, D., Voet, J. G., & Pratt, C. W. (2016). *Fundamentals of biochemistry: life at the molecular level*, (5ta ed.). John Wiley & Sons. https://www.wiley.com/en-us/Fundamentals+of+Biochemistry%3A+Life+at+the+Molecular+Level%2C+5th+Edition-p-9781118918401.

Waghu, F. H., Barai, R. S., Gurung, P., & Idicula-Thomas, S. (2015). Campr3: a database on sequences, structures and signatures of antimicrobial peptides. *Nucleic Acids Research*, 44, D1094 – D1097. https://doi.org/10.1093/nar/gkv1051.

Wang, G., Li, X., & Wang, Z. (2015). Apd3: the antimicrobial peptide database as a tool for research and education. *Nucleic Acids Research*, 44, D1087 – D1093. https://doi.org/10.1093/nar/gkv1278.

Witten, I. H., Frank, E., & Hall, M. A. (2011). *Data Mining: Practical Machine Learning Tools and Techniques*, (3a ed.). Morgan Kaufmann. https://doi.org/10.1016/C2009-0-19715-5.

Witten, I. H., Frank, E., Trigg, L. E., Hall, M. A., Holmes, G., & Cunningham, S. J. (1999). Weka: Practical machine learning tools and techniques with java implementations. In *Computer Science Working Papers*. [Archivo PDF] https://researchcommons.waikato.ac.nz/handle/10289/1040.

World Health Organization (2015). Global action plan on antimicrobial resistance. [Archivo PDF] https://www.who.int/publications/i/item/9789241509763.

World Health Organization (2021). Antimicrobial resistance. [Hoja informativa en línea] Consultado el 24 Abril 2023 en https://www.who.int/news-room/fact-sheets/detail/antimicrobial-resistance.

Yanofsky, C., Horn, V., & Thorpe, D. (1964). Protein structure relationships revealed by mutational analysis. *Science*, 146, 1593 – 1594. https://doi.org/10.1126/science.146.3651.1593.

yue Kang, X., Dong, F., Shi, C., Liu, S., Sun, J., Chen, J., Li, H., Xu, H., Lao, X., & Zheng, H. (2019). Dramp 2.0, an updated data repository of antimicrobial peptides. *Scientific Data*, 6. https://doi.org/10.1038/s41597-019-0154-y.

Zhang, Q., Yan, Z., Meng, Y., Hong, X., Shao, G., Ma, J., Cheng, X., Liu, J., Kang, J., & Fu, C. (2021). Antimicrobial peptides: mechanism of action, activity and clinical potential. *Military Medical Research*, 8. https://doi.org/10.1186/s40779-021-00343-2.

Anexos

En este anexo se presentan códigos en lenguaje Python implementados en nodos de *Python Script* de KNIME para incorporarlos al flujo de trabajo en la etapa de limpieza de datos (fragmentos 1 y 2) y de selección de características (fragmentos 3 y 4).

Fragmento 1. Código en python de la implementación del Filtro por Entropía.

```
import numpy as np
import pandas as pd
from scipy.stats import entropy
from joblib import Parallel, delayed
import multiprocessing
import math

def _remove_by_low_entropy(idx_col, percent, arr_val):
    carray = np.copy(arr_val[:, idx_col])
    carray.sort()
    min = carray.min()
    max = carray.max()
    bin_width = (max - min) / carray.size

    pk = np.zeros(carray.size)
    idx_pk = 0
    idx_arr = 0
    lower_limit_inter = carray[idx_arr]
    while idx_arr < carray.size:
        elem = carray[idx_arr]
        if lower_limit_inter <= elem < (lower_limit_inter + bin_width):
            pk[idx_pk] = pk[idx_pk] + 1
            idx_arr = idx_arr + 1
        else:
            lower_limit_inter = lower_limit_inter + bin_width
```

```
            if idx_pk < carray.size - 1:
                idx_pk = idx_pk + 1

    entropy_val = entropy(pk, base=2)
    entropy_max = math.log2(carray.size)
    entropy_cut = entropy_max * percent

    if entropy_val < entropy_cut:
        return idx_col
    return -1

headers_2 = input_table_1.columns.to_numpy()
arr_val_2 = input_table_1.to_numpy()

for idx2rem in Parallel(n_jobs=multiprocessing.cpu_count())(delayed(
    ↪ _remove_by_low_entropy)(col_temp - 1, 0.25, arr_val_2) for col_temp in
    ↪ range(headers_2.size, -1, -1)):
    if idx2rem != -1:
        headers_2 = np.delete(headers_2, [idx2rem])
        arr_val_2 = np.delete(arr_val_2, [idx2rem], axis=1)

output_table_1 = pd.DataFrame(arr_val_2, columns = headers_2)

# Show warning message to indicate that the filter has finished
import warnings
import datetime
warnings.warn("Entropy␣filter␣finished␣at␣" + str(datetime.datetime.now()))
```

Fragmento 2. Código en python del filtro de Correlación de Spearman

```
import numpy as np
import pandas as pd
from scipy import stats

def _remove_correlated_v2(ref_a, ref_b, cutoff, corr_matrix):
    corr_val = corr_matrix[ref_a][ref_b]
    if corr_val > cutoff:
        return ref_b
    return -1

headers_2 = input_table_1.columns.to_numpy()
arr_val_2 = input_table_1.to_numpy()
cutoff = 0.95

remove = np.zeros(headers_2.size)
corr = stats.spearmanr(a=arr_val_2).statistic
idx = 0

while idx < headers_2.size:
    if remove[idx] == 0:
        for col_temp in range(headers_2.size, idx + 1, -1):
            idx2rem = _remove_correlated_v2(idx, col_temp - 1, cutoff, corr)
            if idx2rem != -1:
                remove[idx2rem] = 1
    idx = idx + 1

for idx2rem in range(headers_2.size, 0, -1):
    if remove[idx2rem - 1] == 1:
        headers_2 = np.delete(headers_2, [idx2rem - 1])
        arr_val_2 = np.delete(arr_val_2, [idx2rem - 1], axis=1)
```

```
output_table_1 = pd.DataFrame(arr_val_2, columns = headers_2)


# Show warning message to indicate that the filter has finished
import warnings
import datetime
warnings.warn("Correlation␣filter␣finished␣at␣" + str(datetime.datetime.now()))
```

Fragmento 3. Código en python del algoritmo CFS.

```
import weka.core.jvm as jvm
import numpy as np
import pandas as pd
from weka.attribute_selection import ASSearch, ASEvaluation, AttributeSelection
from weka.core.dataset import create_instances_from_matrices
from weka.filters import Filter

jvm.start()
jvm.start(max_heap_size="10G")
jvm.start(system_info=True)

label = flow_variables['ClassName']
y = np.array(input_table_1[label], dtype='S20')
input_table_1 = input_table_1.drop([label], axis=1)
x = input_table_1.to_numpy()

dataset = create_instances_from_matrices(x, y, name="generated␣from␣mixed␣matrices
    ↪ ")
str2nom = Filter(classname="weka.filters.unsupervised.attribute.StringToNominal",
    ↪ options=["-R", "last"])
str2nom.inputformat(dataset)
dataset = str2nom.filter(dataset)
dataset.class_is_last()

search = ASSearch(classname="weka.attributeSelection.BestFirst", options=["-D", "1
    ↪ ", "-N", "5"])
evaluator = ASEvaluation(classname="weka.attributeSelection.CfsSubsetEval",
    ↪ options=["-P", "1", "-E", "1"])
selection = AttributeSelection()
```

```
selection.search(search)
selection.evaluator(evaluator)
selection.select_attributes(dataset)

best_subset = selection.selected_attributes
best_subset = np.delete(best_subset, -1)

n = len(input_table_1.columns)
aux = pd.DataFrame([n]*n, columns=['CFS'])
#index = [input_table_1.columns[i] for i in best_subset]
aux['CFS'][best_subset]=1

output_table_1 = pd.DataFrame()
output_table_1['Features'] = input_table_1.columns
output_table_1 = pd.concat([output_table_1, aux], axis=1)
```

Fragmento 4. Código en python del uso de los métodos de selección de características.

```
import pandas as pd
import numpy as np

label = flow_variables['ClassName']
y = np.ravel(input_table_1[label])
input_table_1 = input_table_1.drop([label], axis=1)
x = input_table_1.to_numpy()
n = len(input_table_1.columns)
output_table_1 = pd.DataFrame()
output_table_1['Features'] = input_table_1.columns

# --------------------
# SCIKIT-LEARN Methods
# --------------------

import sklearn
from sklearn.feature_selection import chi2, f_classif, mutual_info_classif,
    ↪ r_regression, SelectKBest

# Chi^2 ------------------------------------------------
result,_ = chi2(x, y)
df = pd.DataFrame(result, columns = ['chi2'])
df.sort_values('chi2',ascending=False,inplace=True)
df['chi2'] = range(1,n+1)
output_table_1 = pd.concat([output_table_1, df], axis=1)
print("Chi2␣Ended.")

# ANOVA F ----------------------------------------------
result, _ = f_classif(x, y)
df = pd.DataFrame(result, columns = ['ANOVA'])
```

```
df.sort_values('ANOVA',ascending=False,inplace=True)
df['ANOVA'] = range(1,n+1)
output_table_1 = pd.concat([output_table_1, df], axis=1)
print("ANOVA␣Ended.")

# Mutual information ----------------------------------
result = mutual_info_classif(x, y,random_state=1)
df = pd.DataFrame(result, columns = ['MI'])
df.sort_values('MI',ascending=False,inplace=True)
df['MI'] = range(1,n+1)
output_table_1 = pd.concat([output_table_1, df], axis=1)
print("MI␣Ended.")

# -----------------
# SKFEATURE Methods
# -----------------

from skfeature.function.statistical_based import t_score, gini_index
from skfeature.function.similarity_based import reliefF

# t-score ---------------------------------------------
nombre = 't_score'
result = t_score.t_score(x, y)
df = pd.DataFrame(result, columns = [nombre])
df.sort_values(nombre, ascending=False,inplace=True)
df[nombre] = range(1,n+1)
output_table_1 = pd.concat([output_table_1, df], axis=1)
print("t-score␣Ended.")

# gini_index ------------------------------------------
#Convert class to numeric
s = list(set(y))
y2 = y
```

```
y2[y2 == s[0]] = 0
y2[y2 == s[1]] = 1
nombre = 'gini_index'
result = gini_index.gini_index(x, y2)
df = pd.DataFrame(result, columns = [nombre])
df.sort_values(nombre, ascending=True,inplace=True)
df[nombre] = range(1,n+1)
output_table_1 = pd.concat([output_table_1, df], axis=1)
print("gini_index␣Ended.")

# reliefF ----------------------------------------------
nombre = 'reliefF'
result = reliefF.reliefF(x, y)
df = pd.DataFrame(result, columns = [nombre])
df.sort_values(nombre, ascending=False,inplace=True)
df[nombre] = range(1,n+1)
output_table_1 = pd.concat([output_table_1, df], axis=1)
print("relief-F␣Ended.")
```