UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA QUÍMICA, BIOTECNOLOGÍA Y MATERIALES

DISEÑO E IMPLEMENTACIÓN DE MODELOS PREDICTIVOS DE CLASIFICACIÓN Y REGRESIÓN DE PROTEÍNAS BASADOS EN LA DIGITALIZACIÓN DE PROPIEDADES FISICOQUÍMICAS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL EN BIOTECNOLOGÍA E INGENIERO CIVIL QUÍMICO

KEVIN JAVIER VERGARA VALENZUELA

PROFESOR GUÍA:
ÁLVARO OLIVERA NAPPA

MIEMBROS DE LA COMISIÓN:
JOSÉ SALGADO HERRERA
BÁRBARA ANDREWS FARROW

SANTIAGO DE CHILE
2021

RESUMEN DE LA MEMORIA PARA OPTAR AL TÍTULO DE: Ingeniero Civil en Biotecnología e Ingeniero Civil Químico
POR: Kevin Javier Vergara Valenzuela
FECHA: 2021
PROFESOR GUÍA: Álvaro Olivera Nappa

DISEÑO E IMPLEMENTACIÓN DE MODELOS PREDICTIVOS DE CLASIFICACIÓN Y REGRESIÓN DE PROTEÍNAS BASADOS EN LA DIGITALIZACIÓN DE PROPIEDADES FISICOQUÍMICAS

La ingeniería de proteínas emplea dos procedimientos principales para el mejoramiento de enzimas: diseño racional y evolución dirigida. A raíz de las limitaciones técnicas y económicas de estos métodos, se recurre al aprendizaje de máquinas. Se ha reportado en la literatura diversas formas de codificar las secuencias de péptidos/proteínas para este propósito, pero ha sido poco el uso de la digitalización de las propiedades fisicoquímicas como descriptor principal, la cual tiene el potencial de entregar un perfil íntegro e informativo de una molécula.

Es por esto que la presente Memoria tiene como objetivo plantear una metodología de diseño y validación de modelos predictivos basados en dicha digitalización. Para ello, se trabaja con 18 conjuntos de datos recolectados de distintas referencias. Se centra el estudio en 5 conjuntos con el propósito de abordar problemas de clasificación binaria, regresión, y clasificación multicategórica. La metodología propuesta consiste en cinco etapas secuenciales fundamentales: caracterización del conjuntos de datos, análisis de los espectros de Fourier de las secuencias del conjunto, identificación de la propiedad fisicoquímica más informativa, construcción de un modelo de aprendizaje supervisado optimizado para incorporar la digitalización, y validación de éste mediante el análisis de sus medidores de desempeño.

La caracterización de los conjuntos de datos y de los espectros entregó apreciaciones preliminares que permiten estimar el desempeño de los modelos predictivos. Del conjunto de los espectros se observó que existe posibilidad de identificar patrones de dispersión y de *peaks* entre conjuntos de datos suficientemente distintivos, pero falla al intentar diferenciar grupos con características similares. Sin embargo, se contrastó este proceso con la realización del alineamiento de las secuencias, logrando apreciar que existe más potencial de extraer información de calidad de los espectros de Fourier que de alineamiento de péptidos/proteínas con poca homología.

Para la construcción y validación de modelos predictivos, se tomó en consideración las formas de codificación por *onehot*, *ordinal*, composición aminoacídica y composición por dipéptidos. Se probaron con cinco distintos algoritmos para mantener la generalidad de aplicabilidad de la digitalización: *K-Nearest Neighbors*, *Random Forest*, *Support Vector Machine*, *Artificial Neural Network*, y *Convolutional Neural Network*. Se reporta robustez de la codificación por digitalización frente a las cuatro comunes, exhibiendo rendimientos similares o superiores para cada conjunto de dato de los casos de estudio. Asimismo, se contrastó con los rendimientos de referencia, logrando apreciar

desempeños similares o superiores para los problemas de clasificación binaria, pero falla al abordar problemas de regresión. Para estudios futuros, se propone implementar múltiples propiedades fisicoquímicas en un mismo modelo, junto con incorporar otras formas de codificación para generar un modelo híbrido y potencialmente lograr mejores resultados.

A mi mamá, mi papá, mi abuelita y mis amigos.

Agradecimientos

Han sido años difíciles, pero me ha sido posible avanzar gracias a la gente que quiero, que amo y que adoro. Agradezco muchísimo que jamás dejaron de creer en mí, que, a pesar de las dificultades e incertidumbres, siempre confiaban en que lo lograría. Deseo retribuir todo el cariño, pero es algo que difícilmente sentiré que lo he hecho. Los agradecimientos en palabras no son suficiente para comunicar verdaderamente lo que mi corazón manifiesta.

Quiero agradecer a mi familia, a mi mamá por apoyarme cuando nadie más lo podía hacer, por entregarme años hermosos de infancia y por aguantarme todos estos años. Muchas cosas de mi vida habrían sido más difíciles si no hubiese sido por ella, por su amor y su ayuda. Quiero agradecer a mi padre por enseñarme cosas valiosas de la vida, por siempre intentar dar lo mejor de él mismo y por apoyarme en mis años de adolescencia, a pesar de las diferencias. Agradezco a mis hermanos por soportarme y por hacer más ameno esta vida. Agradezco a mi abuelita por siempre estar completamente segura de mis capacidades y ser un apoyo totalmente incondicional, me siento profundamente agradecido de una abuelita como ella.

Quiero mostrar mi profundo agradecimiento y cariño a mis amigos, a la Vani, al Ale, al Benja, a la Momo, al Raúl, al Tero, y seguiría infinitamente porque hay mucha gente hermosa en mi vida, pero ellos son los que más me han marcado. Espero que se quedaran siempre a mi lado. Simplemente no sé qué habría sido de mí sin ellos, soy completamente mejor persona gracias a su apoyo y nuestras mutuas enseñanzas. Son mis amigos de años, y los amo mucho.

Agradezco a mis amigos de la universidad también, quienes hicieron que mis horribles primeros dos años se transformaran en algo mucho más bonito en especialidad. Me hicieron ver la magia y la belleza en la U. Agradezco mucho que hayan aparecido en mi camino. Ojalá haber compartido mucho más con ellos, pero los recuerdos que tengo serán por siempre algo precioso para mí.

También hay algunas personitas que han aparecido en estos último tiempo que han vuelto estos meses mucho más llevaderos. Gracias por rescatar mi sanidad y hacer que mantenga vivo mi cariño y esperanzas, mis noches habrían sido terribles sin su apoyo.

Por último, agradezco a Pumpkin, mi gatito, que, aunque a veces no quiera recibir mi cariño, lo amo muchísimo, con todo mi corazón.

Tabla de Contenido

Índice de Figuras

Índice de Tablas

1. Introducción

1.1. Antecedentes

1.1.1. Antecedentes Generales

Las proteínas forman parte de diversas estructuras moleculares en absolutamente todos los organismos. Pueden encontrarse en forma de enzima, receptor de membrana, anticuerpo, estructura citoesquelética, precursores de otros compuestos, entre otros [1]. En particular, las enzimas y poseen gran relevancia hoy en día debido a que son capaces de acelerar reacciones químicas hasta 17 órdenes de magnitud [2]. Asimismo, en la actualidad ha crecido el interés en la identificación y el mecanismo de acción de péptidos, secuencias aminoacídicas relativamente cortas (hasta aproximadamente 50 residuos de extensión), esto es a causa de potencial terapéutico, sobre todo para combatir el problema moderno de las resistencias de los patógenos a diversos fármacos [3][4].

Un aspecto importante para considerar es que enzimas que son de distintos dominios o reinos pueden poseer funciones catalíticas similares, pero diferir en ciertas propiedades fisicoquímicas [5]. Esto ocurre a causa de que el sitio catalítico de dichas enzimas es conservado en los aminoácidos que la componen, mientras que se consta de una similitud del 10% para el resto de la secuencia [5]. Además, esta variación de residuos en el 90% incurre en un cambio en las características de plegamiento, estabilidad, funciones adicionales, mecanismos catalíticos, toxicidad, entre otros [5]. No sólo se pueden inducir cambios en aspectos secundarios de las proteínas, sino que también es posible lograr actividades catalíticas mayores al modificar aminoácidos cruciales del sitio activo [6].

1.1.2 Métodos de Ingeniería de Proteínas

La ingeniería de proteínas es un área que posee como objetivo la modificación de la estructura de proteínas (específicamente enzimas) para optimizar otros procesos o aplicaciones. Existen dos grandes métodos ampliamente empleados para este fin [7]. Uno de ello es el Diseño Racional, el cual se basa en proponer mutaciones selectivas (sitio-dirigida), pero requiere de conocimiento previo de la estructura tridimensional, función, mecanismo catalítico, estabilidad frente a temperatura y pH, entre muchas otras propiedades [8]. El otro método es Evolución Dirigida, la cual mimetiza la evolución natural en el laboratorio; es decir, mediante procesos iterativos de mutaciones aleatorias, se logra seleccionar enzimas incrementalmente mejoradas [9].

En los últimos años, se ha vuelto popular la técnica de aprendizaje de máquinas (en inglés, *machine learning*) para poder enfrentar problemas complejos que son difíciles que el ser humano desempeñe. También es útil para automatizar procesos y evitar dependencia de decisiones humanas o trabajos rutinarios. Para ello, es necesario codificar las proteínas en algún formato utilizable. Un gran número de aplicaciones se ha centrado en utilizar información exclusivamente contenida en la secuencia de las cadenas de aminoácidos. Esta forma de abordar el problema es el que se emplea en el presente trabajo.

Otras formas de codificación de proteínas se centran en el uso de su información estructural [10], información evolutiva [11], oncología de genes [12] o el uso de herramientas de procesamiento de lenguaje natural, como es el caso del *word2vec* [13]. En el presente trabajo de título se aborda una gran gama de problemáticas de mundo real

en el ámbito de ingeniería de proteínas, tales como clasificación de péptidos según acción antimicrobiana, identificación de péptidos inmunomoduladores, determinación de la temperatura media de denaturación, entre otros. Dos problema ampliamente referenciado en la literatura que no se aborda en el presente trabajo es sobre las interacciones proteína-proteína o sustrato-proteína [14], y sobre la predicción de estructura proteica [15].

1.2. Motivación

Últimamente, el uso de péptidos como agentes terapéuticos ha adquirido importancia. En la década pasada, esto no era un tema de interés por parte de compañías farmacéuticas debido a limitaciones de especificidad, transporte, productividad enzimática, baja biodisponibilidad, y alta exposición a peptidasas. Sin embargo, gracias a avances en la ingeniería de procesos y la ingeniería de proteínas, la industria de salud se reenfocó en la utilización de péptidos terapéuticos para enfrentar diversas patologías, entre ellas destacan enfermedades cardiovasculares, cáncer e infecciones microbianas [16]–[18].

Existen limitaciones importantes de los métodos de ingeniería de proteínas mencionados anteriormente, lo cual ha hecho fracasar a varios intentos previos de mejoramiento de enzimas, o simplemente se torna imposible de hacer. En el Diseño Racional, el conocimiento de estructura tridimensional es un requerimiento imprescindible, lo cual no suele ser posible de lograr experimentalmente debido a la incapacidad de cristalizar la proteína [19]. Sin embargo, técnicas de modelamiento computacional de esta característica puede efectuarse para suplir esta falencia, pero tampoco es sugerible basar propuestas de diseño únicamente dichas predicciones a causa de la dificultad de aclarar las limitaciones que presenta el modelo [20]. Con respecto a la Evolución Dirigida, se debe disponer de una gran cantidad de recursos materiales y temporales para poder llevar a cabo el proceso iterativo [21].

No obstante, estos problemas pueden ser superados al emplear aprendizaje de máquinas para evitar los requerimientos de conocimiento complejo de proteínas y de recursos materiales/temporales. Al emplear una cantidad de datos de péptidos o proteínas caracterizados anteriormente, es posible implementar un modelo predictivo que sea capaz de estimar la productividad de éstos o la identificación de alguna cualidad importante (por ejemplo, que una proteína sea capaz de unirse a una molécula de ADN).

El presente trabajo se limitaría únicamente a la predicción de la productividad de péptidos y no a posibles mutaciones. De lo contrario, se requeriría mayor tiempo para poder lograr un trabajo satisfactorio. Sin embargo, dado a que la productividad sería la propiedad que se desea maximizar, se tendría una función objetivo preparada para un posterior trabajo de optimización que lograría proponer mutaciones.

1.4. Objetivos

1.4.1. Objetivo General

Diseñar y validar una metodología de desarrollo de modelos predictivos para problemas de clasificación y regresión de proteínas mediante la digitalización de propiedades fisicoquímicas y técnicas de *machine learning*.

1.4.2. Objetivos Específicos

1. Recolectar y caracterizar distintos conjuntos de datos de secuencias de proteínas o péptidos para problemas de clasificación y de regresión.
2. Codificar los conjuntos de datos según: *One-hot encoder*, *Ordinal encoder*, composición aminoacídica, y digitalización de propiedades fisicoquímicas.
3. Caracterizar y analizar los espectros de propiedades fisicoquímicas para la identificación preliminar de patrones cualitativos que permitirían estimar de manera anticipada el desempeño de modelos predictivos.
4. Implementar algoritmos de aprendizaje supervisado que consideren un método de validación cruzada, para problemas de regresión y de clasificación mediante el uso de cada tipo de codificación empleada.
5. Evaluar los distintos modelos mediante indicadores de desempeño y contrastar con otras formas de codificación.

1.6. Resultados Esperados

Caracterización de los conjuntos de datos: se espera extraer información relevante de los conjuntos de datos que permitan estimar cualitativamente su potencial atribución de información a los modelos predictivos. Se espera preprocesar los datos para facilitar los pasos siguientes.

Digitalización: se deberían lograr espectros que caractericen cada péptido por el conjunto de residuos, en lugar de sólo presentar información individual de los aminoácidos.

Algoritmos de aprendizaje supervisado: constar con varios algoritmos que logren predecir distintos indicadores y/o clases de péptidos, cuyos parámetros estén optimizados a partir del método de validación cruzada.

Evaluación de modelos y validación de la metodología: se espera que las métricas de desempeño sean similares o superiores a otras formas de codificación.

2. Marco Teórico

2.1. Codificación

Para poder trabajar con algoritmos de aprendizaje supervisado, es necesario transformar las secuencias aminoacídicas en vectores numéricos y, dependiendo del algoritmo escogido, es posible que todos los vectores del conjunto de datos deban tener el mismo número de dimensiones. Existe una gran variedad de métodos de codificación, y se detallan a continuación [22]:

2.1.1. One-hot encoder

Este tipo de codificación entrega información sobre la posición y orden de los aminoácidos de la proteína. Asimismo, se entrega cierta representación implícita de la composición de aminoácidos presentes en la secuencia [23]. En este formato, cada aminoácido se representa por un vector fila donde cada columna corresponda a uno de los 20 aminoácidos naturales. Si el aminoácido en cuestión corresponde a la columna, entonces ese elemento se denota con un "1" mientras a los demás se le asignan el valor "0". Al concatenarse con el resto de los vectores para cada aminoácido, se obtiene la matriz sparse de perfil binario. Sea AA el conjunto de 20 aminoácidos naturales ordenados en un vector fila y $s_i \in S$ el i-ésimo aminoácido de la secuencia S. El elemento de matriz $a_{i,j} \in \boldsymbol{A}$ resultante es:

$$a_{i,j} = \begin{cases} 1, & si\ s_i = AA_j \\ 0, & si\ s_i \neq AA_j \end{cases} \tag{2.1}$$

Esto puede ilustrarse mejor mediante un ejemplo: si se considera un péptido de secuencia "MVGHAR", entonces la matriz resultante **A** es:

$$\boldsymbol{A} = \begin{pmatrix} A & R & N & D & C & E & Q & G & H & I & L & K & M & F & P & S & T & W & Y & V \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.2}$$

Dado a que algunos de los algoritmos de *machine learning* requieren una entrada de tamaño uniforme entre todos sus ejemplares, la codificación *one-hot* puede requerir un procesamiento adicional para cumplir esta necesidad. Para ello, se puede introducir un aminoácido "falso" que tiene el único propósito de representar brechas entre las secuencias de distintos tamaños [22]. De esta manera, funciona similar a un tipo de *zero-padding*, donde simplemente se agregan ceros hasta lograr un largo igual al de la secuencia de mayor tamaño del conjunto de datos. Otra forma de atacar este problema es mediante el uso de *sliding-windows*, donde cada secuencia se particiona en pequeños fragmentos para lograr un conjunto de datos con secuencias más pequeñas de igual tamaño [22] [23] [24].

Una vez se tiene la matriz sparse **A**, se procede a transformarla en un vector largo formado por la concatenación de cada vector fila de la matriz **A**. Por ende, una secuencia de 50 aminoácidos de largo sería representado por un vector de dimensión 50 x 20 = 1000

dimensiones (previo al procesamiento por *sliding-windows* o *zero-padding*). Este tipo de representación sufre de dificultades dada al gran tamaño de los vectores de entrada, por lo que no es una forma de codificación preferida actualmente. Sin embargo, es útil al emplear la representación de perfil binario únicamente en los extremos terminales de la secuencia en conjunto con otras formas de codificación [23].

2.1.2. Composición de aminoácido (AAC)

Dado a que la codificación *one-hot* causa un gran número de dimensiones en los vectores de entrada a los algoritmos de aprendizaje, se ha utilizado ampliamente la representación de la secuencia peptídica en base a su composición aminoacídica [23], [25]-[26]. En este formato de representación, cada secuencia se codifica en un vector de largo 20 (un elemento para cada aminoácido natural), y el valor que se le asigna al i-ésimo elemento es la frecuencia de aparición del i-ésimo aminoácido en la totalidad de la cadena. De esta forma, toda proteína se condensa a un vector denso de largo fijo. De esta forma, el elemento del vector que codifica la secuencia quedaría definido por:

$$AAC_i = f(aa_i) = \frac{N° \; aminoácidos \; de \; aa_i}{N° \; aminoácidos \; totales} \tag{2.3}$$

A modo de ejemplo, si se considera el siguiente péptido: "GLWSKIKEAAKTAGLMAMGFVNDMV", la cantidad de aminoácidos de glicina (G) es 3, mientras que la cadena tiene un largo total de 25 aminoácidos. Por lo tanto, $f(G) = \frac{3}{25} = 0{,}12$. Se procede de manera análoga para el resto de los 19 aminoácidos. La composición de aminoácidos ha demostrado ser útil en problemas de clasificación de péptidos antimicrobianos (AMPs por sus siglas en inglés, *antimicrobial peptides*) [23][25][27].

2.1.4. Composición de dipéptidos y tripéptidos

Al utilizar la codificación por AAC, se pierda información en torno a la vecinidad de los residuos de la secuencia de la proteína. Una forma de abordar este problema es similar al indicador de transición (T) de la sección anterior: en lugar de sólo determinar frecuencia de aparición de transiciones, se representa la frecuencia normalizada de la presencia de cada par de aminoácidos posibles (i.e. 20^2 pares). Se ha observado que en algunos casos, se prefiere esta forma de codificación, pero suele ser usada en conjunto con las anteriormente mencionadas [23][28][29][30]. El elemento del vector que recopila la composición de dipéptidos (DPC) se define a continuación:

$$DPC_i = f\left(\left(aa_i, aa_j\right)\right) = \frac{N° \; de \; pares \; \left(aa_i, aa_j\right)}{N° \; de \; pares \; totales \; posibles \; en \; la \; secuencia} \tag{2.4}$$

Donde aa_i y aa_j son uno de los 20 aminoácidos naturales. En ocasiones, es posible preferir la evaluación de tripéptidos [28]. Sin embargo, en estos casos, el número de dimensiones del vector resultante sería de 20^3, lo cual genera dificultades para trabajar con los algoritmos de *machine learning*. Para enfrentar este problema, se puede recurrir a realizar un análisis composicional del conjunto de datos completo y rescatar únicamente los tripéptidos que se encuentran en mayor frecuencia [28]. Para ello, es posible utilizar la entropía de información, la cual está definida (en un dipéptido), por [30]:

$$I(a,b) = f(a,b) \cdot \ln\left(\frac{f(a,b)}{f(a)f(b)}\right) \tag{2.5}$$

Donde a, b son residuos, $f(a,b)$ representa la frecuencia normalizada del dipéptido (a,b) y $f(a)$ es la frecuencia normalizada del residuo a. Para un tripéptido, se tiene:

$$I(a,b,c) = I(a,b) - I(a,b|c) \tag{2.6}$$

A su vez:

$$I(a,b|c) = H(a|c) - H(a|b,c) \tag{2.7}$$

Y, por último:

$$H(a|c) = -\frac{f(a,c)}{f(c)} \ln\left(\frac{f(a,c)}{f(c)}\right) \tag{2.8}$$

$$H(a|b,c) = -\frac{f(a,b,c)}{f(b,c)} \ln\left(\frac{f(a,b,c)}{f(b,c)}\right) \tag{2.9}$$

Luego, aquellos dipéptidos o tripéptidos que tengan entropías altas (en comparación a las demás entropías) serán las que entregan menos información, por lo que pueden despreciarse. No obstante, este paso no ha sido ampliamente reportado en la literatura.

2.1.5. Composición terminal

El formato del vector de composición terminal es equivalente al de AAC. Sin embargo, en estos casos sólo se prefiere trabajar con los extremos de los péptidos. Por lo tanto, las cadenas se reducen a secuencia pequeña de una determinada cantidad de aminoácido, ya sea del extremo terminal C ó N. También puede ocurrir concatenación de estos terminales en una única secuencia [23]. Se ha empleado esta representación en los estudios [23], [28] y [29]. En general, este formato de codificación se prefiere cuando se tiene un conocimiento previo de que estas regiones tienen relevancia sobre la actividad de los péptidos.

2.1.3. Composición, transición y distribución (CTD)

Este formato de codificación pretende expandir la información contenida en la composición aminoacídica, pues ésta carece de detalle respecto a la vecinidad de los aminoácidos. Esencialmente, la AAC no puede mostrar patrones relevantes en la secuencia (ej: si un sector de la cadena presenta una alta concentración de cierto residuo, o si existe una interacción repetitiva entre ciertos aminoácidos) [27]. Para ello, se emplean tres indicadores adicionales que se concatenarán entre sí para formar el vector de entrada: composición (C), transición (T) y distribución (D). Esta forma de representación ha sido relevante para numerosos estudios [31][26][30].

La primera de las características (C) es básicamente la misma composición aminoacídica explicada en la sección anterior (AAC). El segundo indicador (T) muestra la frecuencia de transiciones entre el aminoácido en cuestión y otros residuos, en relación con el número de transiciones totales posibles. Por lo tanto, para dos residuos r y s, su frecuencia de transición estaría definida de la siguiente manera [30]:

$$T(r,s) = \frac{N(r,s) + N(s,r)}{N-1}, \qquad r,s \in \{(aa_i, aa_j) \mid aa_i, aa_j \in AA \wedge i \neq j\} \tag{2.10}$$

Donde $N(r,s)$ es el número de veces que aparece la transición del residuo r al s dentro de la secuencia, N es el largo total de la cadena, y AA es el conjunto de los 20 aminoácidos naturales y, además, $i,j \in \{1,2,\dots 20\}$. Luego, el vector de transiciones contendría el valor de $T(r,s)$ para cada par de residuos. Tomando como ejemplo la secuencia utilizada en la sección anterior, "GLWSKIKEAAKTAGLMAMGFVNDMV", la frecuencia normalizada de transición para el par (G,L) sería $T(G,L) = \frac{2}{24} = 0{,}12$.

En cuanto a la tercera característica (D), ésta indica la posición del aminoácido en determinados porcentajes en relación con su número de apariciones totales, en particular, en la primera aparición, 25%, 50%, 75% y 100% de progreso (llámense $D_1(r), D_{25}(r), D_{50}(r), D_{75}(r) y\ D_{100}(r)$, respectivamente, siendo r un residuo). En la cadena empleada como ejemplo, los valores de distribuciones para el aminoácido alanina serían:

- El primero aparece en la posición 9. Por lo tanto, $D_1(A) = \frac{9}{25} = 0{,}36$.
- Dado a que el número total de alaninas son 4, el progreso del 25% se obtiene también en la posición 9, por lo que $D_{25}(A) = 0{,}36$.
- El progreso al 50% se obtiene en la posición 10, lo cual equivale a $D_{50}(A) = \frac{10}{25} = 0{,}40$.
- El progreso al 75% se obtiene en la posición 13. Por ende, $D_{75}(A) = \frac{13}{25} = 0{,}52$.
- Por último, el progreso al 100% se logra en la posición 17. Es decir, $D_{100}(A) = \frac{17}{25} = 0{,}68$.

Las dimensiones de las tres características serían: para (C), es un vector fijo de tamaño 20 (por el número de aminoácidos naturales); para (T), se tiene una combinatoria de 20 sobre 2 posibilidades totales (190); para (D), se tiene 5 valores por cada uno de los 20 aminoácidos, lo cual da un total de 100 dimensiones. Por lo tanto, al concatenar los tres indicadores en un único vector, se obtienen 310 dimensiones totales.

2.1.6. Composición de pseudo-aminoácidos (PseAAC)

La composición de pseudo-aminoácidos fue introducida por primera vez por Ken-Chou Chen en 2001 [32], y ha sido utilizado en diversos modelos recientes [25][26][30][33][34]. La codificación por PseAAC incorpora información sobre las interacciones entre residuos en determinados gaps de distancia, en particular, utiliza algún indicador relevante, como alguna propiedad fisicoquímica. En la Figura 2 se expone de una manera más fácil de entender las interacciones por dichos gaps [34].

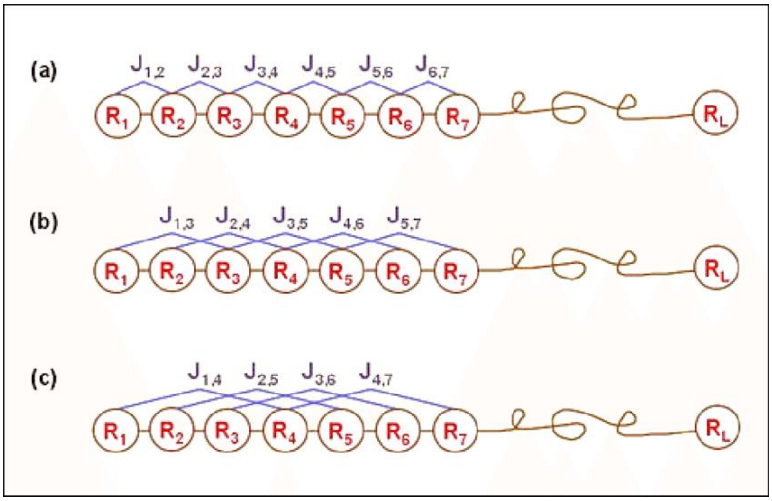


Figura 1: interacción entre residuos ($J_{i,j}$): (a) en gaps de 1 residuo de distancia, (b) en gaps de 2 residuos de distancia, (c) en gaps de 3 residuos de distancia. Fuente: [34].

El vector que representa el PseAAC de una secuencia S se compone de $20 + \lambda$ elementos. Los primeros 20 corresponden a la composición usual de aminoácidos (AAC). Los siguientes λ presenta la información de interacción entre residuos en gaps variantes, desde 1 hasta λ. Esto implica que λ debe ser menor que el largo total de la cadena polipeptídica. Para tener vectores de igual magnitud en un conjunto de proteínas, se emplea un λ de 1 unidad menor que la cadena más corta. A continuación, se muestra la definición del vector P que codifica a S, junto con la definición de cada elemento [33]:

$$P = [p_1, p_2, \cdots, p_{20}, p_{20+1}, \cdots, p_{20+\lambda}] \tag{2.11}$$

$$p_u = \begin{cases} \dfrac{f_u}{\sum_{i=1}^{20} f_i + w\sum_{j=1}^{\lambda}\theta_j}, & (1 \le u \le 20) \\ \dfrac{w\theta_{u-20}}{\sum_{i=1}^{20} f_i + w\sum_{j=1}^{\lambda}\theta_j}, & (20+1 \le u \le 20+\lambda, \quad \lambda < L) \end{cases} \tag{2.12}$$

Donde w es una constante ($w = 0{,}05$ según [30] y [32]) y θ_j es un factor de correlación en un gap j definido por:

$$\theta_j = \frac{1}{L-j}\sum_{i=1}^{L-j}\Theta(R_i, R_{i+1}), \quad (1 \le j \le \lambda, \ \ \lambda < L) \tag{2.13}$$

A su vez, $\Theta(R_i, R_j)$ denota la función de correlación, la cual está dada por:

$$\Theta(R_i, R_j) = H(R_i) \cdot H(R_j) \tag{2.14}$$

Donde $H(R)$ es el valor de la propiedad/indicador del aminoácido R.

2.1.7. Alfabeto reducido de aminoácidos

En este formato de codificación, los 20 aminoácidos naturales se agrupan en clases definidas según algún atributo fisicoquímico. Posteriormente, se trabajan con las representaciones mencionadas en las secciones anteriores. En esencia, lo único que hace este tipo de codificación es cambiar la secuencia de la proteína por una más simple, pero que logra rescatar información local, en particular, de variación en las secuencias [22]. En la Figura 3, se ilustra de manera sencilla el agrupamiento de residuos. Este tipo de representación ha sido reportado en varios modelos recientes [35][36][37]. Un ejemplo de un alfabeto reducido es la agrupación de aminoácidos según hidrofobicidad (débilmente polar, fuertemente polar, apolar).

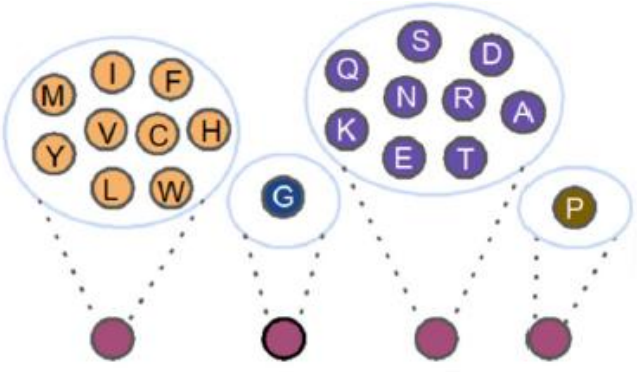


Figura 2: agrupación de los 20 aminoácidos en 4 clusters. Cada una de las agrupaciones denotará una nueva letra que represente la propiedad fisicoquímica. Fuente: [22].

2.1.8. Modelo N-gram

Básicamente, un modelo N-gram es simplemente la generalización de los casos de composición de dipéptidos y tripéptidos. Es decir, que se obtienen combinaciones de m^n polipéptidos posibles, donde m es la cantidad de clases presentes (20 en el caso de no utilizar un alfabeto reducido), y n es el gram empleado (largo de los polipéptidos considerados) [24][38].

2.1.9. Propiedades fisicoquímicas

Como se mencionó en las codificaciones de PseAAC y de alfabeto reducido, se recurre a indicadores de propiedades fisicoquímicas. Asimismo, es posible emplear dichas propiedades fisicoquímicas de manera directa, es decir, que un vector correspondiente a X propiedad fisicoquímica tendrá el largo de la secuencia original, donde cada elemento es asignado el valor correspondiente según el aminoácido en esa posición. Una amplia cantidad de modelos han recurrido a la utilización de propiedades fisicoquímicas [25][26][31][30][29][36][37][39][40][41][42][43]. AAindex corresponde a una base de datos con 565 propiedades fisicoquímicas reportados para cada uno de los 20 aminoácidos naturales [44].

2.1.10. Autocorrelación

Una autocorrelación muestra la interdependencia que existe entre dos señales en una serie de tiempo. En el caso de secuencias aminoacídicas, esto denota los patrones de periodicidad que pueden estar ocultos [22]. Existen dos autocorrelaciones ampliamente

utilizadas: la correlación de Moron, que representa relaciones locales, y la correlación de Broto-Moreau, que indica relaciones globales. Estos indicadores pueden presentar valores positivos (indica que las propiedades se extienden o imitan), o valores negativos (implica interdependencia entre residuos) [22]. La codificación por autocorrelación ha sido reportado en varios modelos recientes [26][30][37][42][45]. Las siguientes ecuaciones definen las correlaciones de Moran y Broto-Moreau, respectivamente [30]:

$$MA(k) = \frac{\frac{1}{N-k}\sum_{i=1}^{N-k}(P(A_i) - \bar{P})(P(A_{i+1}) - \bar{P})}{\frac{1}{N}\sum_{i=1}^{N}(P(A_i) - \bar{P})^2}, \quad (1 \leq k \leq lag \tag{2.15}$$

$$MBA(k) = \frac{\sum_{i=1}^{N-k}\left(P(A_i)P(A_{i+1})\right)}{N-k}, \quad (1 \leq k \leq lag) \tag{2.16}$$

Donde k denota el k-ésimo descriptor (propiedad fisicoquímica), A_i es el i-ésimo aminoácido de la secuencia, $P(A_i)$ es la propiedad fisicoquímica normalizada del aminoácido A_i, $\bar{P}$ denota el promedio de dicha propiedad, N es la cantidad total de residuos de la cadena y lag es un parámetro que debe ser ajustado.

2.1.11. Matriz de puntaje de posición específica (*Position Specific Scoring Matrix, PSSM*)

Este formato de codificación requiere información evolutiva de las proteínas empleadas, de esta manera, se tiene una idea de los residuos propensos a mutación y aquellos que se han conservado más a lo largo de las generaciones. Esto ha sido útil para una gran gama de problemas, desde identificación de interacciones proteína-proteína, clasificación de péptidos antimicrobianos, determinación de sitios de unión, y obtención de actividades [30][28][46]–[53]. La secuencia de proteína se representa en una matriz $PSSM = \{PSSM_{i,j} : i = 1, \cdots, L,\ j = 1, \cdots, 20\}$, donde i representa el i-ésimo residuo de la proteína de largo L, y j denota los 20 aminoácidos naturales. Luego, cada elemento de $PSSM$ se define como sigue [48]:

$$PSSM_{i,j} = \sum_{k=1}^{20} p(i,k) \cdot q(j,k) \tag{2.17}$$

Donde $p(i,k)$ representa la frecuencia normalizada de aparición del aminoácido k en la posición i de la secuencia, y $q(j,k)$ denota el valor de la matriz de mutación de Dayhoff (u otra matriz de sustitución) entre el aminoácido j y k. La información de dichos valores se puede obtener mediante herramientas de alineamiento de posición específica iterada (PSI-BLAST).

2.1.12. Transformada de Fourier (digitalización de propiedades fisicoquímicas)

Este tipo de codificación puede ser empleado para detectar patrones ocultos al transformar el perfil de algún indicador de la proteína (esto sería equivalente al dominio temporal) al dominio de frecuencias. Se ha empleado para detectar similitudes entre proteínas o para extraer información estructural de ella [53][54]. Asimismo, también se

ha usado para mejorar el rendimiento de modelos de *machine learning* [40][41]. Básicamente, lo que se hace es aplicar la transformada de Fourier sobre un perfil de alguna propiedad fisicoquímica, mediante la siguiente expresión [40][41]:

$$f_j = \sum_{k=0}^{N-1} x_k e^{\frac{-2i\pi}{N} jk} \tag{2.18}$$

Donde j es un índice de la transformada de Fourier, x_k denota propiedad fisicoquímica del k-ésimo residuo de la secuencia de largo N, e i es el número imaginario ($i^2 = -1$). Se prefiere trabajar con el módulo de la transformada ($|f_j|$). En la Figura 3 es posible ver un contraste entre los espectros de frecuencia para una proteína nativa y su variante. La digitalización de propiedades fisicoquímicas permite modificar todo el espectro de frecuencia al introducir un único cambio a una secuencia.

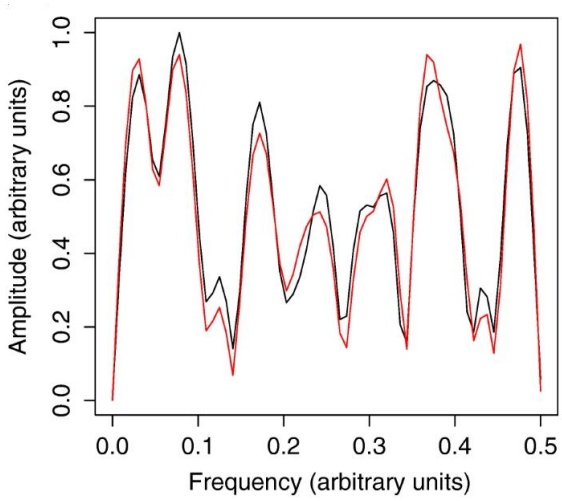


Figura 3: Espectro de frecuencia de para una proteína nativa GLP-1 (rojo) y su variante producto de una mutación puntual (negro), para una misma propiedad fisicoquímica. Fuente: [41].

2.2. Preprocesamiento de los datos

Al tener un conjunto de datos, se suelen tener magnitudes de varianzas distintas entre cada atributo. Esto repercutiría de manera negativa en el algoritmo de aprendizaje supervisado debido a que se estaría otorgando más influencia a aquellas variables con mayor varianza, lo cual se entiende como un sesgo. Por esto, se puede aplicar un escalamiento de los datos: *min-max scaling*, que es básicamente la normalización de cada atributo de manera que tengan un rango de [0,1] [55].

$$x^*_{i,j} = \frac{x_{i,j} - \min(x_j)}{\max(x_j) - \min(x_j)} \tag{2.19}$$

Donde *i* representa una muestra del conjunto de datos y *j* un atributo. Además, se deben identificar cuáles atributos son continuos y cuáles son discretos (categóricos o nominales), pues esto es lo que condicionará métodos futuros de selección de características. Luego,

se procede a reordenar la matriz de conjuntos para que los atributos estén separados según tipo. De esta forma se facilita manipulación posterior de los datos.

No obstante, si bien el método de *min-max scaling* es un paso viable para muchos problemas de *machine learning*, en algunos casos puede introducir otro tipo de sesgo. En el caso del trabajo presente, se emplean espectros de frecuencia para cada muestra de un conjunto de datos, por lo que cada atributo corresponde a una frecuencia dentro de dicho espectro. Si se realizara un escalamiento por el proceso descrito, se perdería información de los *peaks*, ya que se les atribuiría una importancia similar a magnitudes absolutas más pequeñas (generalmente ruido). En su lugar, es posible realizar un escalamiento simple de todo el conjunto mediante la amplitud máxima, es decir:

$$x_{i,j}^{*} = \frac{x_{i,j}}{\max(X)} \tag{2.20}$$

Donde X corresponde a todos los valores del conjunto de datos. Es decir, que se emplea el máximo de todo el conjunto considerando todas las frecuencias. De esta forma, se asegura $0 \leq x_{i,j}^{*} \leq 1$ a nivel de conjunto en lugar de atributo. No se opta por estandarizar las muestras debido a que se requiere información de las distintas varianzas de cada frecuencia, pues puede ser que a lo largo de un mismo conjunto de clasificación la gran mayoría de los péptidos tengan amplitudes similares para frecuencias bajas, pero exhiban diferencias notorias en frecuencias altas entre las clases.

En el caso de problemas de clasificación, es posible incluir un paso adicional en el preprocesamiento, que corresponde a amplificar las clases subrepresentadas (sobremuestreo o *oversampling*) o reducir aquellas sobrerrepresentadas (submuestreo oo *undersampling*) en el caso de que se tengan clases no balanceadas [56]. Para el sobremuestreo, se escogen aleatoriamente muestras de las clases minoritarias y se agregan a los conjuntos de entrenamiento, hasta que las clases queden suficientemente balanceadas. Se procede de manera análoga para el submuestreo. Sin embargo, suele ser complicado determinar las proporciones óptimas que logran los mejores resultados en el desempeño final [56].

Otro punto importante en el preprocesamiento de datos es la identificación de *outliers*. En el presente estudio en particular, al tratarse de bases de datos de proteínas/péptidos, es de especial interés la magnitud de respuesta en problemas de regresión y el balance de los largos de secuencias de las muestras, transversal a todos los casos de estudio. Para ello, se determinan los *outliers* mediante el criterio del z-score [57]. Específicamente, se considera *outlier* cualquiera que tenga un z-score de al menos 3, es decir, que el valor se aleje del promedio en ± 3 unidades de desviación estándar.

Para poder visualizar mejor la distribución de los largos de secuencia, junto con su relación con las distintas clases de un conjunto, se puede emplear la estimación de densidad de kernel, pues es un formato más cómodo para superponer múltiples distribuciones, en lugar de utilizar superposiciones de histogramas. Asimismo, se puede observar funciones continuas en lugar de valores discretos, lo cual ayuda a determinar con mayor precisión valores intermedios que suelen faltar en histogramas. Esto se lleva a cabo de la siguiente manera [58][59]:

$$\hat{p}_n(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \quad (2.21)$$

Donde n es el número de datos, h es ancho de banda empleado, d es la dimensionalidad de los datos, K es una función kernel (i. e. satisface no-negatividad y condiciones de normalización), y X_i es cada dato. En el presente trabajo, se emplean la configuración por defecto de la librería Pandas de Python, la cual utiliza un kernel Gaussiano y un ancho de banda determinado por la regla de Scott [60].

2.3. Selección de características

Cada péptido/proteína es una cadena de varios aminoácidos, por lo que al emplear la codificación por M propiedades fisicoquímicas se tendría una matriz de MxN por cada muestra (M: número de atributos fisicoquímicos, N: número de residuos). Se ha reportado anteriormente la implementación de algoritmos que recogen 500 atributos, lo cual implica que la cantidad de datos que deben manejarse es muy alta [61]. Lo anterior puede incurrir en una pérdida de poder predictivo [62]. Actualmente, existen 565 propiedades fisicoquímicas en la base AAindex. Esto se ocasiona a causa de la necesidad de disponer de un conjunto mayor de péptidos para entregar un volumen suficiente de datos en el grupo de entrenamiento.

Para resolver este asunto, es posible determinar un número pequeño de propiedades fisicoquímicas que logren preservar la mayor información posible. Esto fue realizado por un trabajo inmediatamente previo [63], donde se agruparon distintos índices de la base AAindex en *clusters*. Las propiedades que pertenecen a un mismo *cluster* exhiben similitudes entre sí y presentan diferencias sustanciales del resto de los grupos generados. Esto se realizó mediante algoritmos de *clusterización* que escapan del alcance del presente trabajo. Se formaron un total de 8 grupos, cuyas propiedades fisicoquímicas más relevantes se presentan a continuación:

Tabla 1: Propiedades fisicoquímicas empleadas en este trabajo.

Cluster	Índice de propiedad más representativa	Nombre de la propiedad
Alfa	PRAM900102	Frecuencia relativa en hélices alfa
Beta	PRA900103	Frecuencia relativa en hojas beta
Energético	COSI940101	Valores de potencial de interacción electrón-ion
Hidrofílico	HOPT810101	Valor de hidrofilicidad
Hidrofóbico	JOND750101	Hidrofobicidad
Estructura secundaria	RADA880106	Área de superficie accesible
Volumen	GRAR740103	Volumen
Índice	FASG760101	Peso Molecular

2.6. Algoritmos de aprendizaje supervisado

Se han empleado una gran variedad de algoritmos de aprendizaje para enfrentar tanto problemas de regresión como de clasificación binaria y multicategórica [64]. A continuación, se detallará a grandes rasgos los métodos más empleados, y algunos que pueden figurar como interesantes debido a su rendimiento o a su propuesta de novedad.

2.6.1. K-nearest neighbor (KNN)

Corresponde a un algoritmo relativamente simple que no requiere parámetro alguno aparte de la definición de un K para la determinación del número de vecinos próximos. El funcionamiento del método es el siguiente: dado un conjunto de datos de entrenamiento y un K, se determina la distancia existente entre la nueva observación y sus K-vecinos más cercanos. En problemas de clasificación, se decide por un voto mayoritario, mientras que en problemas de regresión se determina por la media de la variable objetivo. Se pueden emplear distintas métricas para el cálculo de distancias, sin embargo, las más empleadas son las distancias euclidianas y de Hamming para variables continuas y categóricas, respectivamente:

$$D_{Euclidiana} = \sqrt{\sum_{i=1}^{k} (x_i - \hat{x}_i)} \tag{2.22}$$

$$D_{Hamming} = \sum_{i=1}^{k} |x_i - \hat{x}_i| \tag{2.23}$$

Donde x_i es el i-ésimo componente del vector del conjunto de entrenamiento, mientras $\hat{x}_i$ es el vector de la nueva observación. En la distancia de Hamming, si $x_i = \hat{x}_i$ entonces la diferencia es 0, y si $x_i \neq \hat{x}_i$ la diferencia es 1. KNN ha sido reportado en [26] y [42]. Para determiner el parámetro K, es posible emplear una función de costos que se pretenda minimizar, mediante el uso de un conjunto de validación. Es decir, que se evalúa dicha función (o métrica de desempeño) a lo largo de un rango predefinido de K, y se escoge aquel que optimice el costo/métrica.

2.6.3. Support vector machine (SVM)

Ha sido utilizado en un gran número de modelos [26][28][29][42][43][47][53]. Para problemas de clasificación binaria, el algoritmo realiza una separación de los datos mediante un hiperplano w con máximo grosor que permita la distinción entre dos clases. Por lo tanto, el problema de optimización es el siguiente [26]:

$$\min \frac{1}{2} \|w\|^2 \tag{2.24}$$
$$s.a. \quad y_i(w \cdot x_i + b) \geq 1, \qquad \forall x_i$$

Donde $x_i \in X$ es el conjunto de datos de entrada (péptidos), $y_i \in Y$ son las clases dos clases ($y_i = 1$ si es que x_i corresponde a una clase y_i, y $y_i = -1$ en caso contrario), y $|b|/\|w\|$ es la distancia perpendicular del hiperplano al origen.

Cuando se tiene un conjunto linealmente separable (i. e. que los elementos de una clase no se encuentran en la clase contraria delimitado por el hiperplano), SVM logra altos rendimientos. Sin embargo, en pocas ocasiones se logra tal separación. Por lo tanto, pero mantener la generalidad, es posible incorporar un término de penalización a la función a minimizar:

$$\begin{gathered} \min \frac{1}{2}\|w\|^2 + C\sum_i \xi_i \\ s.a.\ y_i(w \cdot x_i + b) \geq 1 - \xi_i, \qquad \forall x_i \\ \xi_i \geq 0 \end{gathered} \tag{2.25}$$

Donde ξ_i es la distancia de x_i al límite del hiperplano que separa la clase a la cual debería pertenecer. En el caso de que x_i esté caracterizado correctamente por el hiperplano, entonces $\xi_i = 0$. También puede ocurrir que un conjunto de datos no pueda ser separable por un hiperplano lineal, pero sí por uno no-lineal. Para enfrentar esta dificultad, se mapea el conjunto a un hiperespacio de dimensión mayor para lograr separar linealmente los datos en dicho hiperespacio. Esto se realiza mediante una función Kernel $K(x_i, x_j)$. Existen distintos tipos de funciones Kernel, a modo de ejemplo, se tiene el Kernel Gaussiano (también conocido como *Radial Basis Kernel*) [26]:

$$K(x_i, x_j) = e^{-\frac{\|x_j - x_i\|^2}{2\sigma^2}} \tag{2.26}$$

Donde σ es un parámetro. Luego, los parámetros a ajustar son C y σ, junto con la decisión de la función Kernel a emplear. Para abordar problemas de clasificación multicategóricos, es posible emplear el método de *One vs All* y determinar distintos hiperplanos. Por último, los problemas de regresión también pueden solucionarse mediante SVM. La manera de proceder posee ciertas diferencias en la formulación de la función objetivo y las restricciones, pero el razonamiento es esencialmente el mismo [65]. Existen algunas variaciones reportadas en algunos modelos recientes: Relevance Vector Machine (RVM) [48], Discriminant Vector Machine (DVM) [50], y Probabilistic Classification Vector Machine (PCVM) [52].

2.6.4. Decision Trees

Los árboles de decisión trabajan mediante la partición de los atributos de entrada mediante el uso de alguna función de costos, generalmente ganancia de información (*cross-entropy*) o índice Gini. Empleando la ganancia de información como ejemplo, ya que ha sido reportado en [66] y [67]:

$$IG(A, S) = H(S) - \sum_{t \in T} p(t)H(t) \tag{2.27}$$

Donde $IG(A, S)$ es la ganancia de información al separar el conjunto de datos S a partir del atributo A. $H(S)$ es la entropía del conjunto S, $p(t)$ es la proporción de elementos de t en el conjunto S, t es subconjunto de S originado al emplear A como indicador, tal que $S = \bigcup_{t \in T} t$, y $H(t)$ es la entropía del subconjunto t.

$$H(S) = \sum_{c \in C} -p(c) \cdot \log_2\big(p(c)\big) \tag{2.28}$$

Donde $c \in \{0,1\}$ (i. e., indica pertenencia cierta clase o no). El cálculo es similar entre atributos continuos y categóricos, pues los árboles de decisión siempre particionan el conjunto de datos. Una vez determinada la ganancia de información, se opta por particionar el conjunto mediante el atributo que maximice dicha ganancia. Posteriormente, se itera hasta lograr una cantidad de particiones suficientes. Dado a que la cantidad de particiones puede variar (e incluso ser infinita en atributos continuos), se requiere optimizar el árbol de decisión. Sin embargo, este método posee la ventaja de mostrar los criterios de separación empleado en cada nodo, por lo que es posible obtener conclusión fenomenológica a partir de este algoritmo.

2.6.5. Random Forest (RF) y Gradient Boosting Decision Trees (GBDT)

Dado a que el algoritmo de árboles de decisión suele tener un poder predictivo débil a causa de la inestabilidad frente a variaciones en el conjunto de datos, se suele preferir emplear algoritmos como RF y GBDT [36][30][42]. Básicamente, en el primer método se genera un gran número de instancias aleatorias de árboles de decisiones. Luego, en problemas de clasificación, la clase predicha se obtiene por voto mayoritario de los árboles de decisión, mientras que en problemas de regresión se determina el promedio de dichos árboles. En el caso de GBDT, el modelo se basa en ensamblar de manera constructiva los árboles de decisión, es decir, que los resultados de una instancia de árbol generan cambios en el siguiente mediante una función que permite ajustar parámetros a partir de una función de costos. Este algoritmo se encuentra detallado en [31].

2.6.6. Artificial Neural Network (ANN)

Uno de los algoritmos más ampliamente usados en años recientes en diversos problemas de predicción (no sólo de bioinformática) son las redes neuronales artificiales (ANN por sus siglas en inglés). En cuanto a problemas de proteínas, su uso ha sido reportado en [26][37][42][49][68]. El funcionamiento de las redes neuronales se basa en capas construidas por nodos: capa de entrada, capas ocultas (optativas) y capa de salida. Cada nodo recibe un escalar de cada nodo de la capa anterior (en el caso de la entrada, cada nodo recibe un elemento del vector codificado del conjunto de datos), y se incorporan en una función de activación, junto con un peso asignado a cada nodo. Posteriormente, esta función de activación se vuelve el *input* de cada uno de los nodos siguientes. La capa de salida genera la predicción final. En la siguiente figura se resume la explicación. w_i es el peso asignado al i-ésimo nodo, x_i es el valor de la función de activación del i-ésimo nodo, n es la cantidad de nodos de la capa anterior, y f es la función de activación. En el caso de que la arquitectura de la red presente alguna capa oculta, se les atribuye el nombre de *Deep Neural Network*, pues esencialmente lo que realizan estas capas es la extracción de características importantes para poder generar una respuesta más favorable [69]

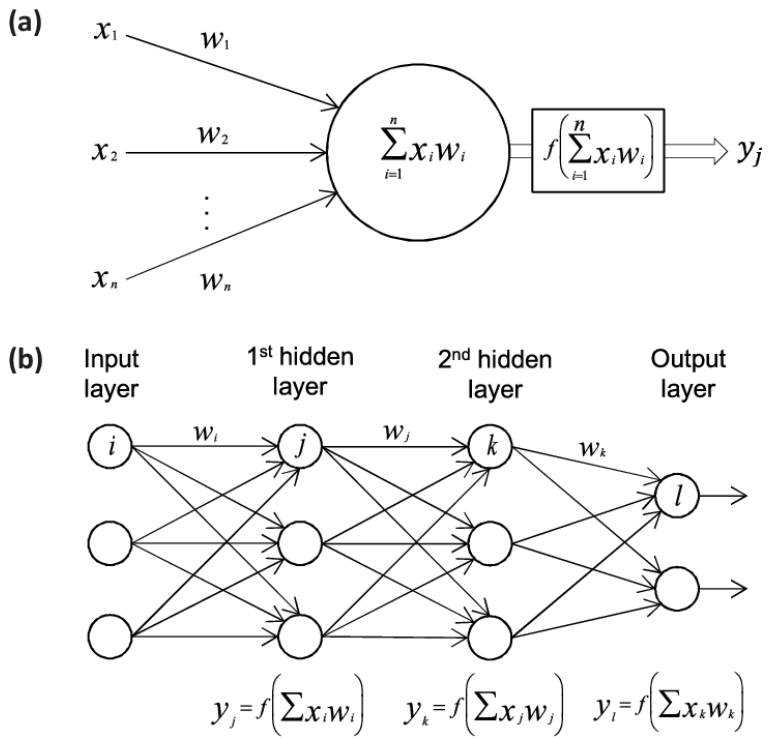


Figura 4: Explicación gráfica del funcionamiento de una red neuronal de dos capas ocultas [70]. *(a) construcción de la función de activación, (b) arquitectura de la red.*

También es possible incorporar un factor *b* (llamado "sesgo" o "*bias*") a cada función de activación de una misma capa:

$$y_j = f\left(\sum_{i=1}^{n} (x_i w_i) + b\right) \tag{2.29}$$

En general, se busca que una función de activación varíe de 0 a 1 ó de -1 a 1. El entrenamiento de una red neuronal se basa en el ajuste de los valores de los pesos $w_{i,j}$. Para ello, existen diversos métodos, siendo el más ampliamente usado el algoritmo de *backward propagation*, el cual se basa en la minimización de una función de costos derivable con respecto a los parámetros de la red (pesos y sesgos).

Existen diversas funciones de activación, siendo unas de las estándares la función sigmoide:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.30}$$

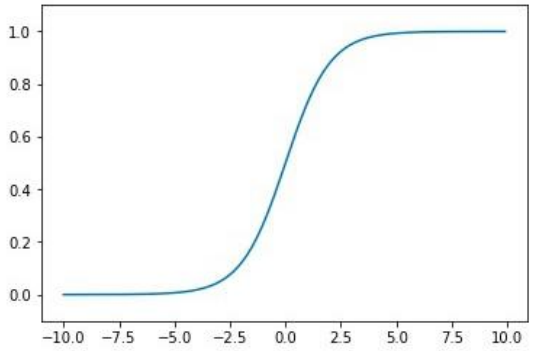


Figura 5: Función sigmoide.

Sin embargo, esta función es propensa a presentar el problema del desvanecimiento del gradiente (*vanishing gradient problem)*, el cual ocurre cuando las funciones de activación generan respuestas muy cercanas a regiones donde su derivada es casi nula. Esto provoca que los pesos sean altamente difíciles de modificar por *backward propagation*. Para corregir este problema, es posible emplear la función ReLU, la cual ha tenido gran éxito en la actualidad y es la función preferida al momento de construir arquitecturas iniciales [71][72].

$$ReLU(x) = \max(0, x) \tag{2.31}$$

A pesar de la fama de ReLU, puede presentar un problema conocido como *Dying ReLU*, el cual consiste en que algunas neuronas pueden inactivarse por el resto del proceso de entrenamiento luego de generar una respuesta nula. Dicho problema no ha sido clarificado del todo en su fundamento teórico, pero se han hecho intentos de generar explicaciones [73].

2.6.6. Convolutional Neural Network (CNN)

En el presente trabajo se incluyen modelos de una variante de las redes neuronales, conocidas como redes neuronales convolucionales. Este tipo de red neuronal presenta una arquitectura para mejorar la extracción de características de información organizada en grillas. Su aplicación más conocida es en el campo de *Computer Vision* para la identificación de patrones en imágenes. Sin embargo, en el presente trabajo se empleará CNN debido a que los espectros de frecuencia igualmente se encuentran organizados en grillas. En el caso de datos en 1 dimensión (ej: series de tiempo o espectros), la capa de entrada de dimensión Nx1 se conecta a una capa convolucional, donde ocurre el proceso de convolución mediante el uso de M filtros, de esta manera, se obtienen M convoluciones del mismo vector de entrada, por lo que la capa convolucional presenta dimensiones Mx(Nx1). Posteriormente, ocurre una capa de *max pooling*, donde se reduce la dimensión de los M vectores de la capa convolucional mediante la preservación del valor máximo de los elementos de cada vector en ventanas de K unidades. En la Figura 6 se aprecia de manera gráfica lo recientemente descrito.

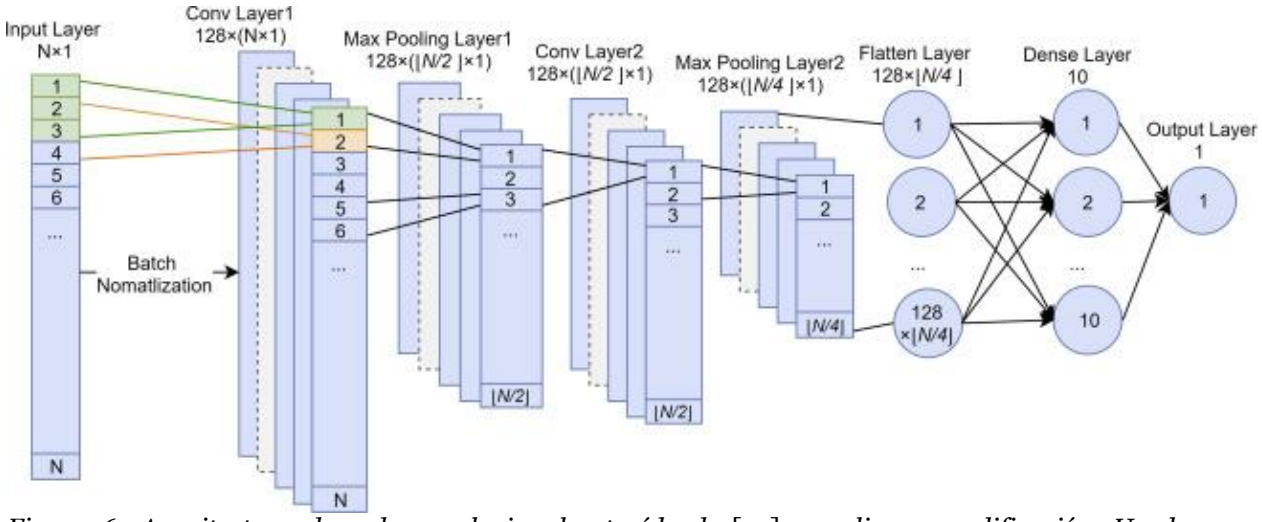


Figura 6: Arquitectura de red convolucional extraído de [74]*, con ligera modificación. Usada para clasificar péptidos antimicrobianos.*

Para el caso del ejemplo explicativo presentado, las capas de *max pooling* emplean ventanas de 3 unidades. Las capas convolucionales y de *max pooling* también puede utilizar un *stride*, que corresponde a las unidades que se desplaza el kernel/ventana para realizar la siguiente convolución/*pooling*. En el caso de la Figura 6, las capas *max pooling* emplean un *stride* de 2 unidades, lo cual causa que las dimensiones de los vectores resultantes sean reducidas a la mitad.

Finalmente, al terminar las capas exclusivas de la arquitectura convolucional, se conecta a una red densa para poder procesar la información resultante. Para ello, se concatenan los vectores de la última capa convolucional para que puedan ser alimentadas a los nodos siguientes.

Existen otras arquitecturas de redes neuronales reportadas para problemas de péptidos/proteínas, como las redes neuronales de larga memoria a corto plazo (en inglés, *Long Short-Term Memory Artificial Neural Network*, LSTM-ANN) [68] y *Autoencoders* [37].

2.7. Validación cruzada

Al entrenar un algoritmo, es necesario disponer de una manera para verificar que esté ajustándose correctamente a los datos. Lo ideal es que se logre generalización, pues un sobreajuste impedirá que logre predecir la productividad de péptidos para datos que no estén dentro del conjunto de entrenamiento. Por otra parte, si se incurre en un subajuste, quiere decir que el algoritmo no está aprendiendo.

Para desarrollar más este concepto, el entrenamiento pretende ajustar los parámetros del algoritmo mediante la minimización de una función de costos. Sin embargo, un valor demasiado pequeño no es lo que se desea (Figura 6c). Si la función no logra converger apropiadamente, se tendría la Figura 6a. Lo que se busca es preservar la generalidad (Figura 6b).

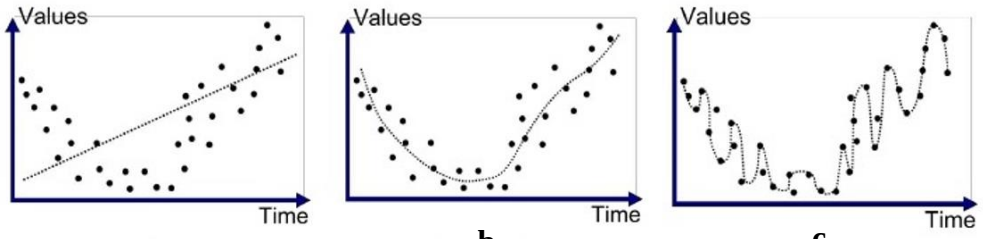


Figura 7: comparación de los distintos tipos de ajustes. Fuente: [75].

Para evitar el sobreajuste, se emplea el método de validación cruzada. Por lo general, se utilizan las técnicas de *k-fold* [29][41][43] y *Leave-One-Out* (LOOCV) [40][41]. El conjunto de datos recolectado se particiona en *k* grupos. Uno de dichos grupos será el de validación, mientras que los demás permiten el entrenamiento. Se vuelve a iterar hasta que cada grupo haya sido un conjunto de validación. Esto permite ajustar hiperparámetros de los modelos para seleccionar aquel que presente mejor desempeño de validación. Una vez se escoge el mejor modelo, se entrena utilizando la totalidad del conjunto de entrenamiento y se prueba en un conjunto independiente que no fue empleado para la validación cruzada. De esta forma, se obtiene un indicador realista del desempeño del modelo. LOOCV es equivalente a *k-fold* cuando k=número de muestras del conjunto. En general, mientras mayor es el tamaño de datos a disposición, menor será el valor de *k* [76].

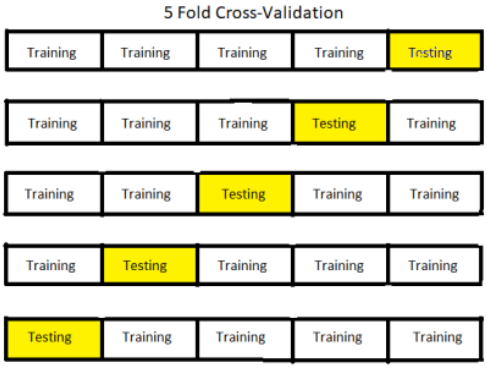


Figura 8: Esquema de validación cruzada para k = 5. Fuente: [77].

2.8. Indicadores de desempeño

2.8.1. Problemas de clasificación binaria

Accuracy
Indica la capacidad del modelo de identificar las distintas clases correctamente [42][29][43].

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.32}$$

Donde TP, TN, FP, FN son el número de verdaderos positivos, verdaderos negativos, falsos positivos y falsos negativos, respectivamente. Esta medida de desempeño es la más utilizada como referencia para ajustar hiperparámetros. Sin embargo, la *accuracy* no logra identificar los sesgos presentes en la capacidad predictiva del modelo. Para abordar este problema, se suelen usar las medidas de desempeño que siguen.

Sensitivity o Recall
Se utiliza ampliamente para modelos de clasificación [29][43][26]. El indicador de sensibilidad indica la capacidad del modelo de predecir correctamente que una proteína pertenece a determinada clase.

$$Sensibility = \frac{TP}{TP + FN} \tag{2.33}$$

Specificity
Este medidor muestra desempeño del modelo al momento de discernir que cierta proteína no pertenece a una determinada clase [29][43][26].

$$Specificity = \frac{TN}{TN + FP} \tag{2.34}$$

Precision
Corresponde al cociente entre las clases positivas (i.e. que el péptido pertenece a dicha clase) predichas correctamente y las instancias observadas totales que se le atribuyen a dicha clase [42][26].

$$Precision = \frac{TP}{TP + FP} \tag{2.35}$$

F-score
La definición de *recall* y *precisión* conllevan a la definición de F-score, el cual corresponde a la media armónica entre los dos indicadores [42].

$$F - score = \frac{2 \times Recall \times Precision}{Recall + Precisión} \tag{2.36}$$

Coeficiente de correlación de Mathews (MCC)
Una limitante de los indicadores anteriores es que no muestran información de toda la matriz de confusión. Para ello, se emplea el coeficiente de correlación de Mathews, que va desde -1 (capacidad predictor totalmente erróneo) a 1 (predicción perfecta) [42][29][43]. Cabe destacar que un valor de 0 denota completa aleatoriedad.

$$MCC = \frac{(TP \times TN) - (FP \times FN)}{\sqrt{(TP \times FP)(TP \times FN)(TN \times FP)(TN \times FN)}} \tag{2.37}$$

Receiver Operating Characteristic curve (ROC)
Se le conoce como ROC a una distribución probabilística del TPR (*true positivve rate*) v/s FPR (*falsa positivve rate*). En clasificadores binarios, los algoritmos de predicción asignan un dato a una de las dos clases basándose en probabilidades de que pertenezcan a una de las dos clases [42][29][43]. Por ende, es necesario establecer cierto umbral para permitir dicha separación. Dado a que el TPR y FPR se relacionan de manera directa, es posible obtener un gráfico al explorar distintos valores de umbral de separación. Luego, la curva bajo la ROC (*Area Under Curve, AUC*) indica el desempeño global del clasificador binario. Un valor de AUC=0 denota predicción perfectamente contraria de las clases, mientras que un valor de 1 indica predicción perfectamente correcta. En consecuencia, un valor de 0,5 implica que el modelo carece de capacidad predictiva.

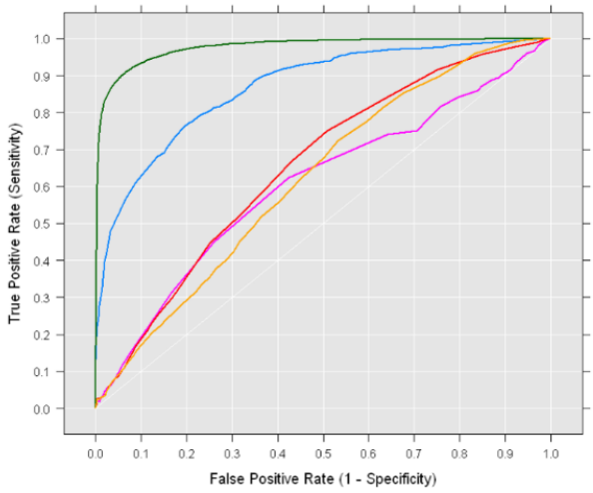


Figura 9: Ejemplo gráfico de un ROC, extraído de [78].

2.8.2. Problemas de clasificación de múltiples clases

En este caso, se procede de manera análoga que los indicadores para problemas de clasificación binaria, pero calculando los medidores para cada una de las clases. Es decir, que se toma el resto de las clases como un único grupo que engloba a todas (conocido también como el método de *One vs All*). Luego, los indicadores globales se determinan como sigue:

$$MR_i = \frac{1}{n}\sum_{k=1}^{n} I_i(Y_k = Z_k) \tag{2.38}$$

Donde MR representa "*Exact Match Ratio*", i denota el i-ésimo indicador, Y es la clase predicha y Z es la clase real. De manera similar, el procedimiento de *One vs All* también se emplea para la realización de ROCs.

2.8.3. Problemas de regresión

Coeficiente de determinación

Ampliamente utilizado en problemas de regresión [40][38][79][80]. Se utiliza para determiner la correlación existente entre valores predichos y valores observados.

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})} \tag{2.39}$$

Donde y_i e $\hat{y}_i$ son los valores predichos y observados, respectivamente, $\bar{y}$ es el promedio de los valores observados. Un valor nulo del coeficiente de determinación indica que el modelo no es mejor a uno basal que siempre predice el promedio de los valores observados, mientras que un valor igual a 1 indica predicción perfecta. A partir de esta definición de R^2, es posible obtener valores negativos, lo cual indica que es incluso peor que el modelo basal.

Error cuadrático Medio

Se emplea para determinar errores basado en la varianza de los valores predichos [40]. Esta medida también corresponde a la función de costos de los problemas de regresión.

$$ECM = \sum_{i=1}^{S} \frac{(y_i - \hat{y}_i)^2}{S} \tag{2.40}$$

2.9. Alineamiento

La mayoría de los conjuntos de datos utilizados en el presente trabajo tienen la particularidad de provenir de distintas especies y/o tener secuencias altamente distintas. Por lo tanto, el alineamiento se vuelve complicado de hacer. Sin embargo, es necesario tener algún indicador de qué tan cercano es una secuencia de otra para poder contrastar la calidad de información que entrega el alineamiento con la calidad de los espectros de frecuencia. Para ello, es posible utilizar una matriz de distancias, donde cada secuencia se compara con otra mediante un valor que represente similitud o potencial homología. La distancia más simple corresponde a la *p-distance*, la cual simplemente corresponde a la proporción de aminoácidos distintos al comparar dos secuencias previamente alineadas:

$$p = \frac{n_{aa}}{N} \tag{2.41}$$

Con n_{aa} el número de aminoácidos distintos y N el largo de la secuencia. Sin embargo, esta medida es simplista al no considerar la tasa de mutación de los distintos aminoácidos. Dado a que algunos residuos son más propensos a ciertos aminoácidos que otros, dos secuencias con la misma *p-distance* pueden presentar distinto grado de verdadera similitud. Ante esto, surgen las distancias Gamma, que toman en consideración la tasa de sustitución de cada aminoácido por otro [81]. Se debe determinar previamente el parámetro α de la distribución Gamma en este caso [82].

3. Metodología

La metodología presentada en este trabajo es, esencialmente, un resultado en sí mismo, pues corresponde a una forma de validar la capacidad predictiva de un modelo que emplea la codificación de la digitalización de propiedades fisicoquímicas, el cual figura como el objetivo general propuesto. La metodología se compone de tres aspectos principales:

3.1. Conjunto de Datos

- Se recolectan conjuntos de datos de péptidos y proteínas que hayan sido reportados en la literatura en un período de no más de 5 años atrás para poder ser consistente con el contraste del desempeño logrado en el trabajo. Se apunta a una diversidad de conjunto de datos: preferentemente deben diferenciarse en largos de secuencias, distribución de clases (para problemas de clasificación), cantidad de muestras, y distintos tipos de problemas a predecir: regresión, clasificación y clasificación multicategórica.
- Se caracterizan a los conjuntos de datos a partir de aspectos intrínsecos de las secuencias: largo de secuencia, composición aminoacídica y composición de dipéptidos. Esto con el objetivo de visualizar patrones útiles que permiten conformar una estimación preliminar de las distintas formas de codificación y la identificación de posibles sesgos involucrados en los conjunto de datos.
- Se caracterizan a los conjuntos de datos desde su respuestas. Es decir, se visualiza la distribución de respuestas para los casos de regresión, mientras que para la clasificación se determina el balance de clases.

3.2. Alineamiento

Se obtiene información a partir del alineamiento de las secuencias de los conjuntos de datos para poder contrastarlo con la calidad de información que se obtiene desde los espectros de Fourier. Para ello:

- Se guardan las secuencias de cada conjunto de dato en un archivo de texto FASTA para poder utilizar algún software que facilite el trabajo de alineamiento. En el presente trabajo, se optó por MEGA. Se realiza el alineamiento múltiple mediante el algoritmo MUSCLE con las configuraciones por defecto.
- Se procede a determinar la matriz de distancia de cada conjunto de dato. Se utiliza el modelo Dayhoff que proporciona MEGA, con configuraciones por defecto. Se transforman los resultados numéricos a un formato gráfico para permitir facilidad de visualización.
- Para conjuntos de secuencias cortas, es posible obtener los gráficos de logo de secuencia para verificar la existencia de secuencias de consenso de interés.

3.3. Caracterización de los espectros:

- Se digitaliza cada conjunto de dato según las 8 propiedades fisicoquímicas presentadas en la Tabla 1.
- Se realizan espectros multicanales y espectrogramas (Figura 10) para facilitar la visualización global de los conjuntos de datos. Esto permite evaluar de manera cualitativa si los espectros de Fourier entregan información interpretable por humanos, lo cual también es útil para estimar preliminarmente su calidad de información para las etapas de entrenamiento.

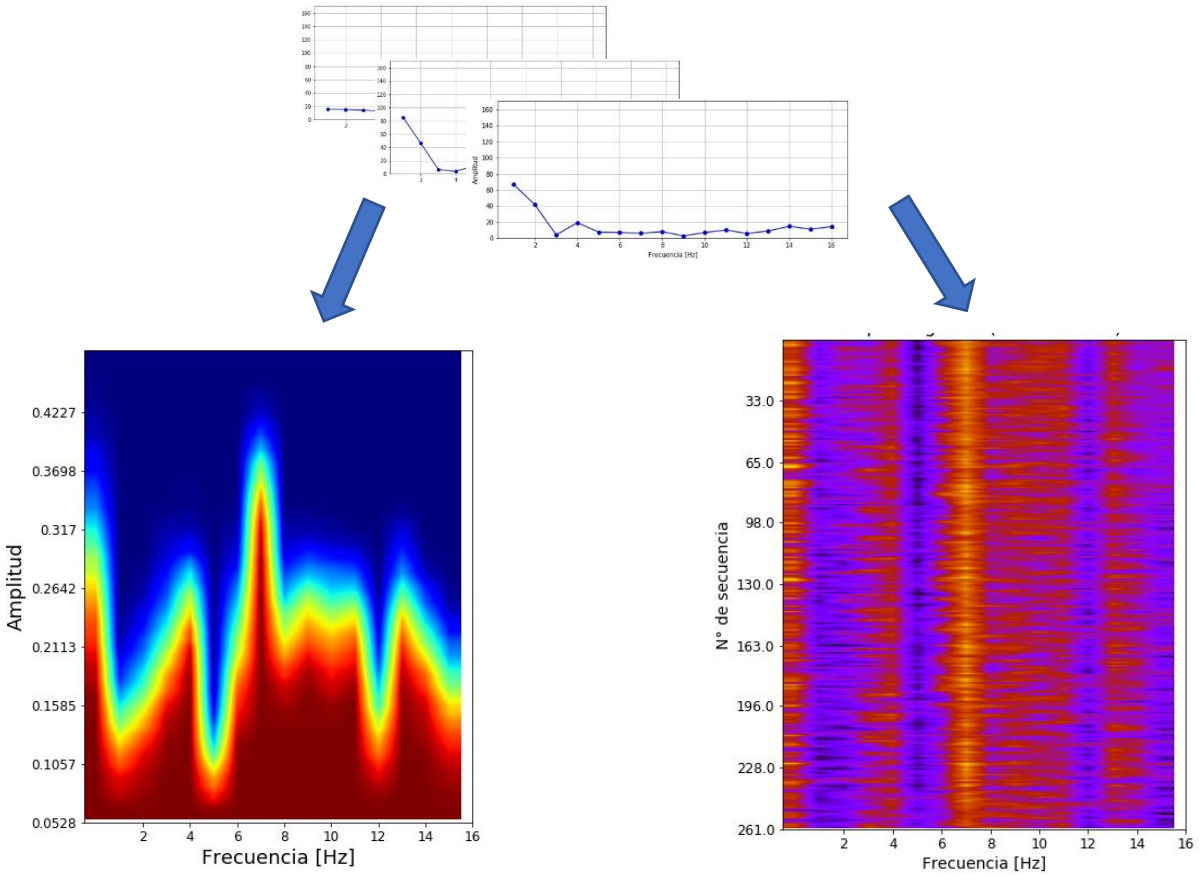


Figura 10: Visualización de los espectros de todo un conjunto en formato de (izquierda) espectro multicanal y (derecha) espectrograma. La idea de los espectros multicanales es poder superponer todas las señales de un mismo conjunto (o una misma clase de él) y notar diferencias en los peaks y distribuciones para las distintas frecuencias. En cuanto a los espectrogramas, se construyen a partir de una matriz de NxM, donde N es el número de muestras totales y M es la frecuencia máxima, por lo tanto, corresponde a otro formato para complementar la información entregada por los espectros multicanales.

3.4. Entrenamiento de los modelos predictivos

- Se codifica cada conjunto de datos en formato *onehot*, *ordinal*, composición aminoacídica y composición de dipéptido para poder compararlos con el desempeño de la digitalización.
- Se escoge algún algoritmo de aprendizaje supervisado para utilizar en la primera etapa de la construcción de los modelos. En esta fase se identifica la propiedad

fisicoquímica más informativa para los conjuntos de datos, mediante la técnica de validación cruzada. Se emplea *kfold* con k=5 y se deja un 20% de los datos como conjunto de prueba (independiente) para los conjuntos con más de 500 muestras, y se utiliza k=10 con un 30% conjunto de prueba en caso contrario.

- Se construyen modelos optimizados para cada una de las formas de codificación. Los algoritmos de aprendizaje empleados en el presente trabajo son cinco: KNN, RF, SVM, ANN y CNN. Se emplean múltiples algoritmos para verificar la generalidad de la digitalización de propiedades fisicoquímicas.
- Se escoge el mejor modelo para cada forma de codificación para cada conjunto de datos y se procede a compararlos mediante los conjuntos de prueba respectivos.
- Finalmente, se muestran todos los medidores de desempeño de interés para las codificaciones por digitalización, y se concluye al respecto.

4. Conjuntos de datos y caracterización

4.1. Recolección de los conjuntos de datos

En esta sección se presentan los 13 conjuntos independientes recolectados de otros estudios que se usaron para validar la metodología presentada. De estos 13, 6 conjuntos corresponden a problemas de clasificación binaria, 1 de clasificación multiclase y 6 de regresión. De esta manera, se abarcan distintos casos de interés en el área de la ingeniería de proteínas. Un tipo de problema que no se estudia en este trabajo de título es el de múltiples etiquetas (*multi-labels*). Algunos de los conjuntos deben dividirse en subconjuntos independientes por el tipo de datos que presentan, este criterio se indica en la columna "¿Mezclar?" de la Tabla 2. Los nombres de los conjuntos se definieron según su referencia respectiva. Una descripción más detallada de los conjuntos se encuentra disponible en Anexo A.

Tabla 2: Conjuntos de datos utilizados.

Conjunto de datos	Subconjunto	Descripción	¿Mezclar?[1]	Problema	Ref.
AntiTb	AntiTb_primary	246 péptidos antituberculares y 246 péptidos antibacterianos sin actividad antitubercular.	No	Clasificación binaria	[23]
	AntiTb_secondary	246 péptidos antituberculares y 246 péptidos sin actividad antitubercular.			
ACP-DL	ACP-DL_240	129 péptidos anticancerígenos y 111 péptidos no-anticancerígenos.	Sí	Clasificación binaria	[24]
	ACP-DL_740	376 péptidos anticancerígenos y 364 péptidos no-anticancerígenos.			
iACP	iACP_benchmark	138 péptidos anticancerígenos y 206 péptidos no-anticancerígenos.	Sí	Clasificación binaria	[27]
	iACP_independent	150 péptidos anticancerígenos y péptidos 150 no-anticancerígenos.			
QSP	QSP	220 péptidos percebidores de cuórum y 220 péptidos sin capacidad de sensar cuórum.	-	Clasificación binaria	[29]
iAMP-2L_multiclass	iAMP-2L_AB	769 péptidos antibacterianos.	Sí	Clasificación multiclase	[33]
	iAMP-2L_AC	140 péptidos anticancerígenos.			
	iAMP-2L_AF	366 péptidos antifúngicos.			
	iAMP-2L_AHIV	86 péptidos anti-VIH.			
	iAMP-2L_AV	124 péptidos antivirales.			
	iAMP-2L_nonAMP	2405 péptidos sin actividad antimicrobiana.			
iAMP-2L_binary	iAMP-2L_binary	920 péptidos antimicrobianos y 920 péptidos no-antimicrobianos.	-	Clasificación binaria	[33]
DBP	DBP	524 proteínas con afinidad por moléculas de ADN y 550 proteínas sin afinidad por moléculas de ADN.	-	Clasificación binaria	[47]
Pop	Pop_ara	Observabilidad de 60288 péptidos en *Arabidopsis thaliana*.	No	Regresión	[80]
	Pop_chlamy	Observabilidad de 97757 péptidos de *Chlamydomonas reinhardtii*.			
	Pop_yeast	Observabilidad de 94320 péptidos de *Saccharomyces cerevisiae*.			
VaxinPad	VaxinPad	304 péptidos inmunomoduladores y 385 péptidos no-inmunomoduladores.	-	Clasificación binaria	[83]
Solub	Solub	Razón entre gramos de proteína soluble y proteína total, para 3148 péptidos/proteínas.	-	Regresión	[84]

[1]: Indica si los subconjuntos pueden tratarse como un único conjunto independiente (Sí) o si deben estudiarse por separado (No).

Tabla 2: Conjuntos de datos utilizados (continuación).

Conjunto de datos	Subconjunto	Descripción	¿Mezclar?[1]	Problema	Ref.
enantioselectivity	enantioselectivity	1 – 8 Mutaciones puntuales en el sitio de unión de epóxido hidrolasa de *Aspergilius niger*. Se determina la enantioselectividad de 152 variantes.	-	Regresión	[85]
localization	localization	Recombinación quimérica de la proteína ChR de células embriónicas humanas de riñón a partir de tres proteínas parentales: CheRiff, C1C2 y CsChrimsonR. Se determina la localización de membrana para 254 variantes,	-	Regresión	[85]
t50	t50	Recombinación quimérica de citocromo P450 de *E. coli* a partir de tres proteínas parentales: CYP102A1, CYP102A2 y CYP102A3. Se determina la temperatura T50 para 261 variantes.	-	Regresión	[85]
RT	RT_hela	Tiempo de retención en RPLC para 3413 péptidos de la línea celular HeLa.	No	Regresión	[79]
	RT_yeast	Tiempo de retención en RPLC para 14361 péptidos de levadura.			

Luego del criterio indicado por la columna "¿Mezclar?", se tienen un total de 18 conjuntos de datos a estudiar, de los cuales 8 corresponden a problemas de clasificación binaria, 1 de clasificación multiclase y 9 de regresión. A continuación, en la Tabla 3 se caracterizan aspectos generales de los 18 conjuntos. Las columnas de "Min", "Max", "Prom." y "STD" corresponden al mínimo, máximo, promedio y desviación estándar de las respuestas de los problemas de regresión.

Tabla 3: Caracterización general sobre largo de secuencias y las respuestas de los conjuntos de datos.

Subconjunto	Largo de cadenas [residuos]	Clase positiva [%]	Clase negativa [%]	Min	Max	Prom.	STD
AntiTb_primary	5 – 61	50.0	50.0	-	-	-	-
AntiTb_secondary	5 – 61	50.0	50.0	-	-	-	-
ACP-DL	11 – 207	51.5	48.5	-	-	-	-
iACP	11 – 138	44.7	55.5	-	-	-	-
QSP	5 – 60	50.0	50.0	-	-	-	-
iAMP-2L_AB*	5 – 107	19.8	-	-	-	-	-
iAMP-2L_AC*	5 – 98	3.6	-	-	-	-	-
iAMP-2L_AF*	6 – 105	9.4	-	-	-	-	-
iAMP-2L_AHIV*	10 – 46	2.2	-	-	-	-	-
iAMP-2L_AV*	5 – 98	3.2	-	-	-	-	-
iAMP-2L_nonAMP*	5 – 100	61.8	-	-	-	-	-
iAMP-2L_binary	5 – 131	50.0	50.0	-	-	-	-
DBP	50 – 2226	48.8	51.2	-	-	-	-
Pop_ara	6 – 319	-	-	1	5.4E+9	5.1E+6	7.2E+7
Pop_chlamy	7 – 2984	-	-	1	7.4E+7	2.7E+4	4.9E+5
Pop_yeast	7 – 356	-	-	1	1.6E+11	5.6E+7	8.9E+8
VaxinPad	3 – 30	44.1	55.9	-	-	-	-
Solub	29 – 1367	-	-	0.0	147.0	49.0	33.4
enantioselectivity	389	-	-	4.0	158.0	50.8	33.8

*: Estas filas corresponden a distintas clases de un mismo conjunto (iAMP-2L_multiclass), por lo que deben compararse a lo largo de la columna.

Tabla 3: Caracterización general sobre largo de secuencias y las respuestas de los conjuntos de datos (continuación).

Subconjunto	Largo de cadenas [residuos]	Clase positiva [%]	Clase negativa [%]	Min	Max	Prom.	STD
localization	329 – 361	-	-	-9.51	-3.96	-6.61	1.26
t50	466	-	-	36.0	64.4	51.8	5.9
RT_hela	7 – 50	-	-	1494	6494	4049	1300
RT_yeast	6 – 38	-	-	681	15759	8930	3791

Se optó por centrar la discusión en torno a cinco casos de estudio, cuyo fundamento se explica a continuación. No obstante, los resultados de los demás conjuntos se encuentran en Anexos A y B.

- **iAMP-2L_multiclase:** se escogió este conjunto de datos por dos razones. En primer lugar, porque corresponde al único tipo de problema de clasificación multiclase que se dispone, por lo que figura como una única oportunidad para validar la metodología para dicho tipo de problema. En segundo lugar, porque logra abarcar el interés recurrente de clasificar péptidos según su capacidad antimicrobiana.
- **VaxinPad:** se decidió emplear un conjunto de dato que tuviera un bajo largo de secuencia, de esta manera se valida la capacidad de los algoritmos predictivos de extraer información de los espectros de Fourier para frecuencias bajas. Además, el tipo de datos es interesante para el área de bioprocesos, ya que VaxinPad se conforma de péptidos inmunomoduladores para facilitar el mecanismo de acción de las vacunas.
- **DBP:** este conjunto figura como el único de clasificación binaria que está formado por proteínas en lugar de péptidos. Por lo tanto, esto permite validar la capacidad de los algoritmos predictivos para extraer información de espectros de Fourier extensos.
- **enantioselectivity:** corresponde al único conjunto de regresión que está formado por variantes de 1 – 8 mutaciones puntuales de una misma proteína *wild-type*. De esta forma, se logra validar la sensibilidad de los espectros de frecuencia frente a pequeñas variaciones en la secuencia original.
- **t50:** junto con localization, figuran como los dos conjuntos de regresión formados por recombinación quimérica de una proteína a partir de distintas proteínas parentales. Esto otorga la posibilidad de validar la capacidad de los espectros de frecuencia de presentar información relevante a un cambio local importante en la secuencia de aminoácidos sin inducir un cambio global radical. Si bien cualquiera de los dos conjuntos t50 o localization servirían para este objetivo, se optó por t50 debido a su relevancia en la industria.

4.2. Caracterización de largo de secuencias

A continuación, se realiza un preprocesamiento de los datos. En la Tabla XX2 es posible verificar que los rangos de largo de secuencias son amplios para algunos datos, lo cual indica una probabilidad importante de que tenga *outilers* presentes. En el caso de que se alimentaran los datos a los algoritmos de aprendizajes sin eliminar dichas muestras, el *zero-padding* necesario de realizar en la etapa de codificación por DFT y *Onehot* podría ser muy extenso, lo cual provocaría mayor tiempo de ejecución de los modelos y una pérdida de representatividad de los espectros de secuencias en rangos más cercanos al promedio. En consecuencia, se preprocesan todos los conjuntos para eliminar los *outliers* según el método del Z-score (para cualquier z-score = ± 3).

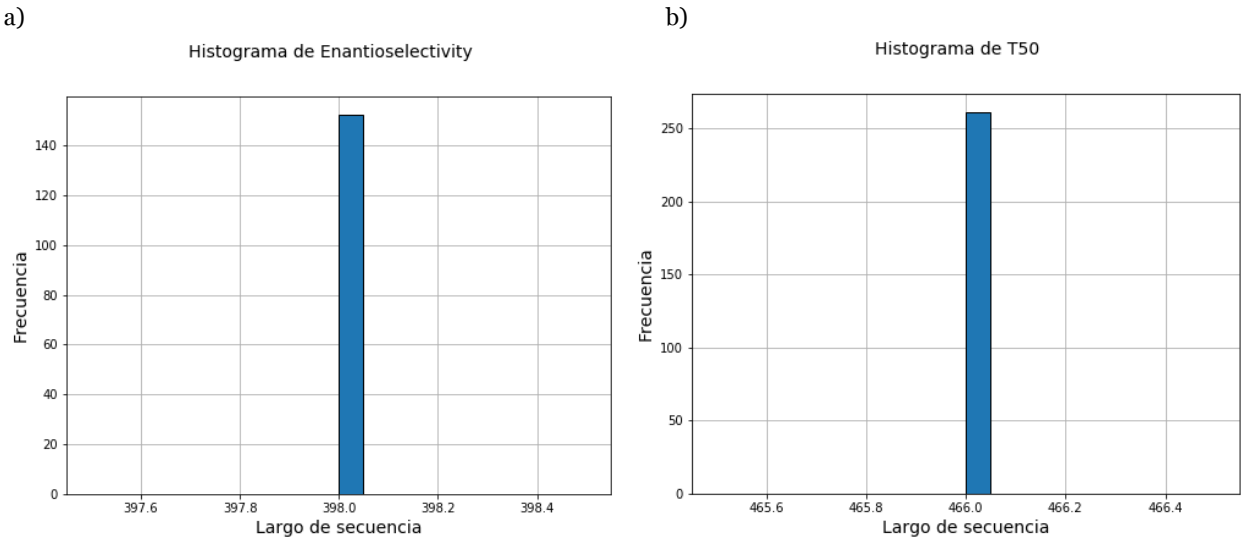


Figura 11: Histograma de largo de secuencias para (a) enantioselectivity y (b) t50.

De la Figura 10 es posible verificar que el largo de secuencia para ambos conjuntos es único (igual a lo largo de todo el conjunto), por lo que no existen *outliers*. Se optó por mostrar estos histogramas para ser consistente con la información mostrada.

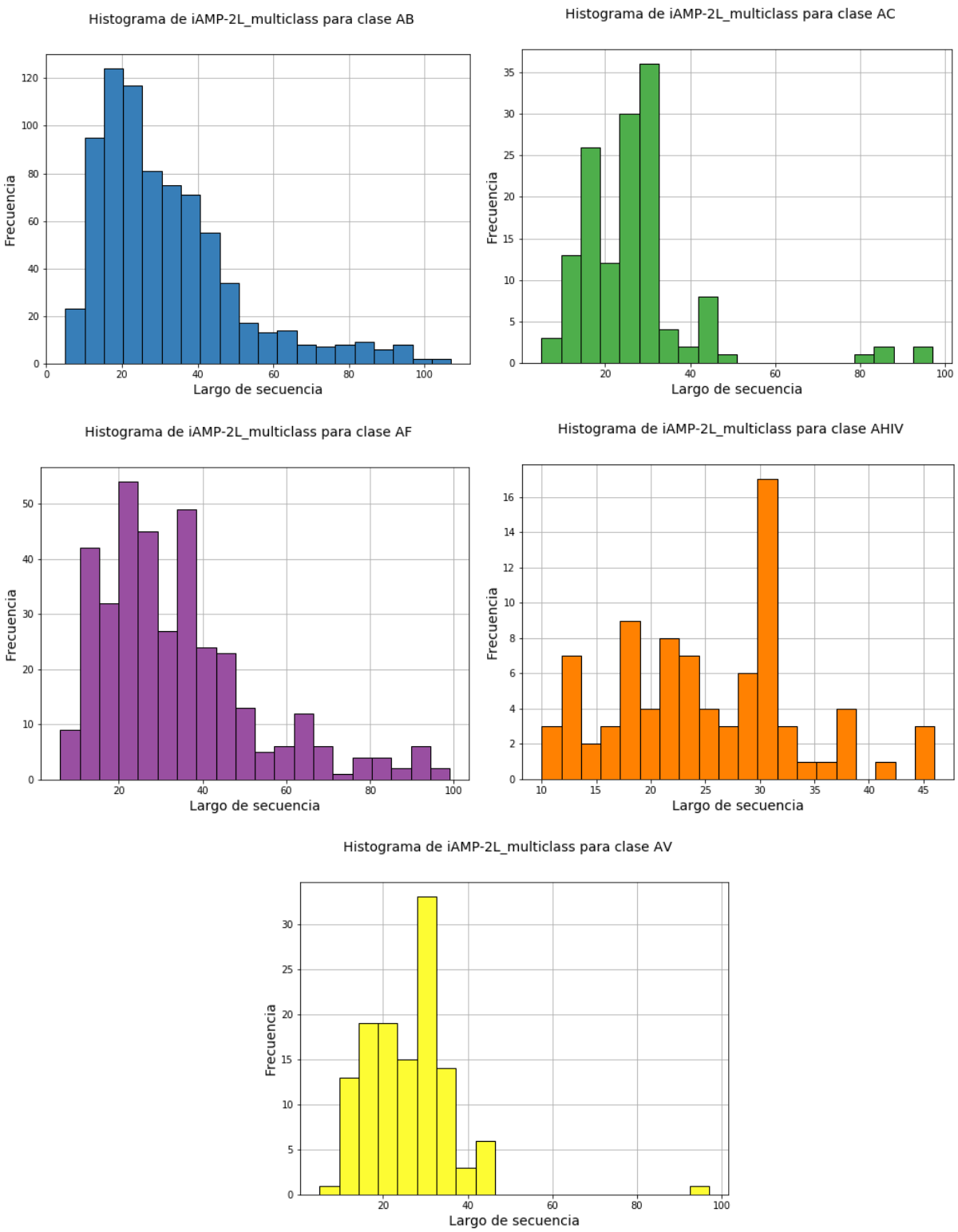
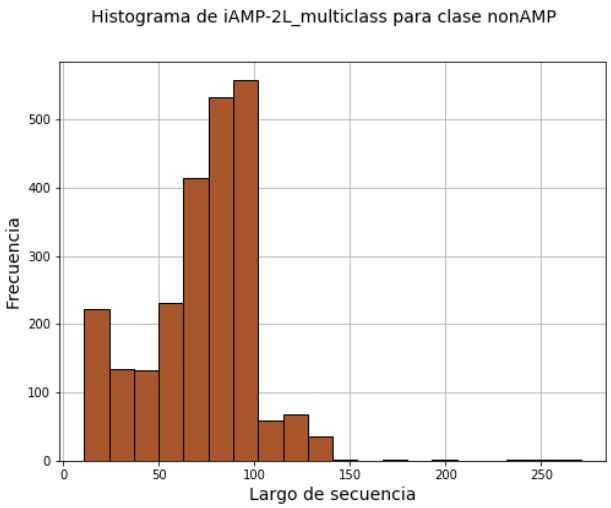


Figura 12: Histogramas de largo de secuencias para las distintas clases de iAMP-2L_multiclass.

(a)



(b)

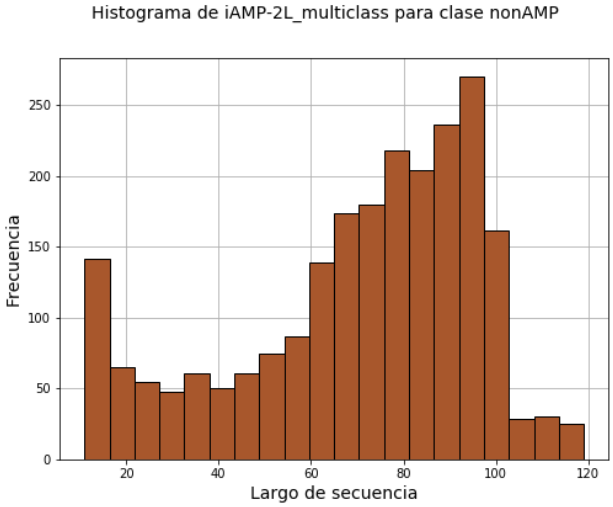
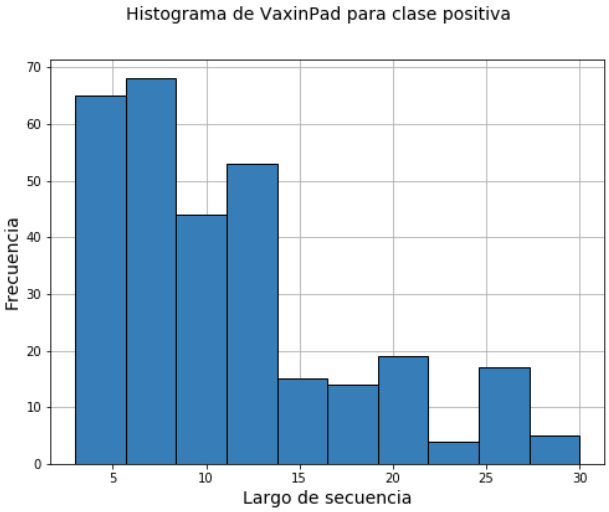


Figura 13: Histograma de largo de secuencias para la clase nonAMP de iAMP-2L_multiclass. (a) Con outliers *presentes. (b) Sin* outliers *presentes.*

En iAMP-2L_multiclass, todos los *outliers* identificados pertenecen a la clase nonAMP, por lo que a las clases presentadas en la Figura 12 no se les eliminó alguna muestra. En la Figura 13 se aprecia el efecto de remover los *outliers*, pues la distribución de la clase nonAMP se aprecia mejor en (b) debido a que el largo de secuencia máximo se redujo a aproximadamente la mitad. Esto tiene como consecuencia que en el *zero-padding* para la codificación por DFT el largo de los espectros se reduzca de 512 a 128 puntos (frecuencias).



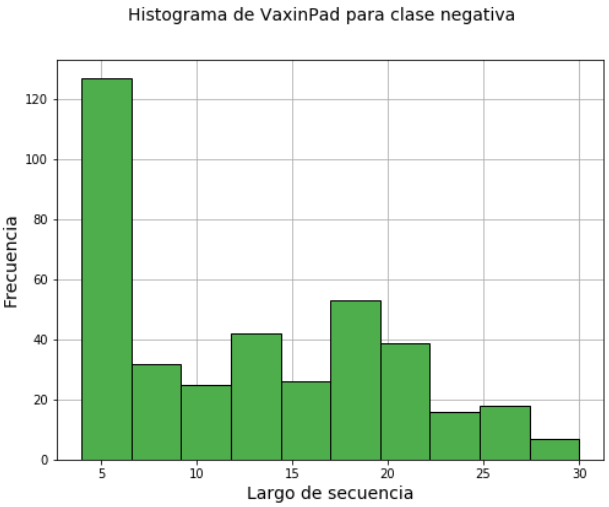


Figura 14: Histogramas de largo de secuencias de las dos clases de VaxinPad para el caso con presencia de outliers.

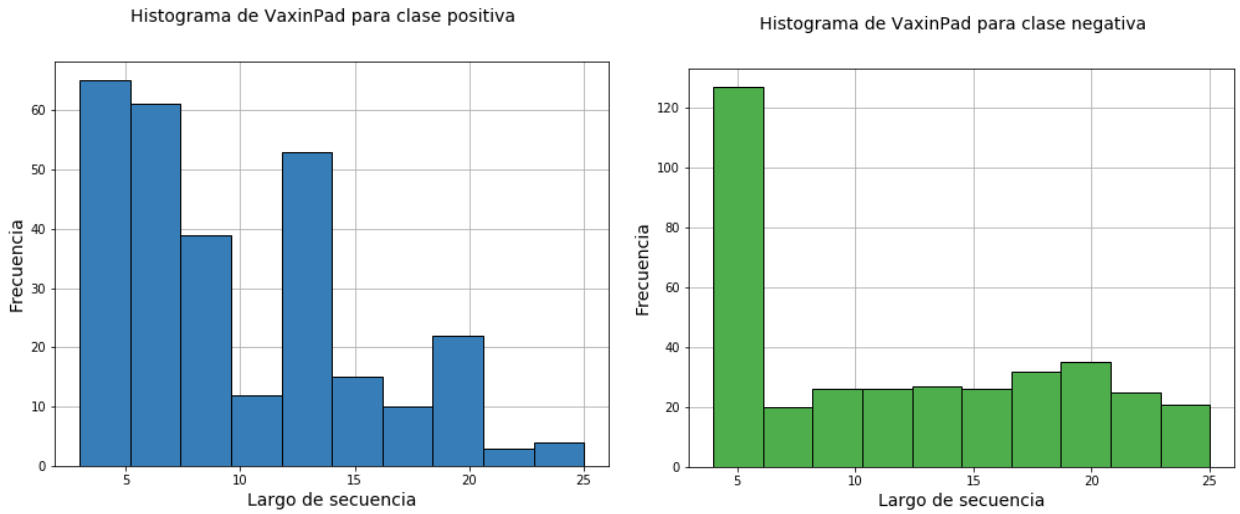


Figura 15: Histogramas de largo de secuencias de las dos clases de VaxinPad para el caso en ausencia de outliers.

En el caso de VaxinPad, la eliminación de *outliers* causó que el largo de secuencia máximo se redujera de 30 a 25 aminoácidos. Sin embargo, dicho cambio no causa una modificación en el *zero-padding* necesario de realizar, pues la potencia de 2 mayor más cercana sigue siendo 32. De todas maneras, esto logra enfocar aún más las frecuencias bajas para los algoritmos de aprendizaje.

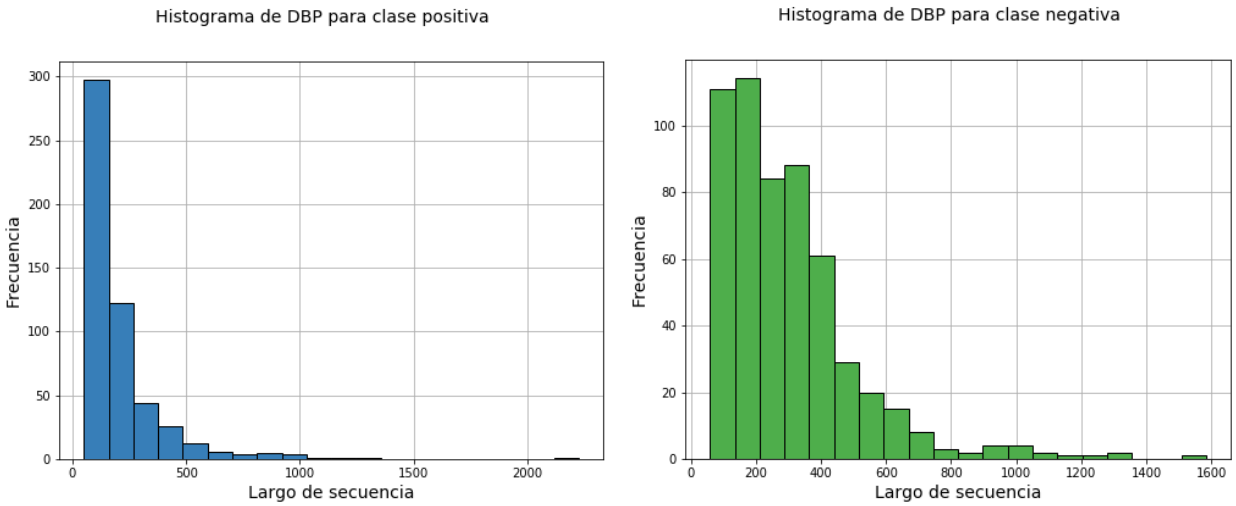


Figura 16: Histogramas de largo de secuencias de las dos clases de DBP para el caso con presencia de outliers.

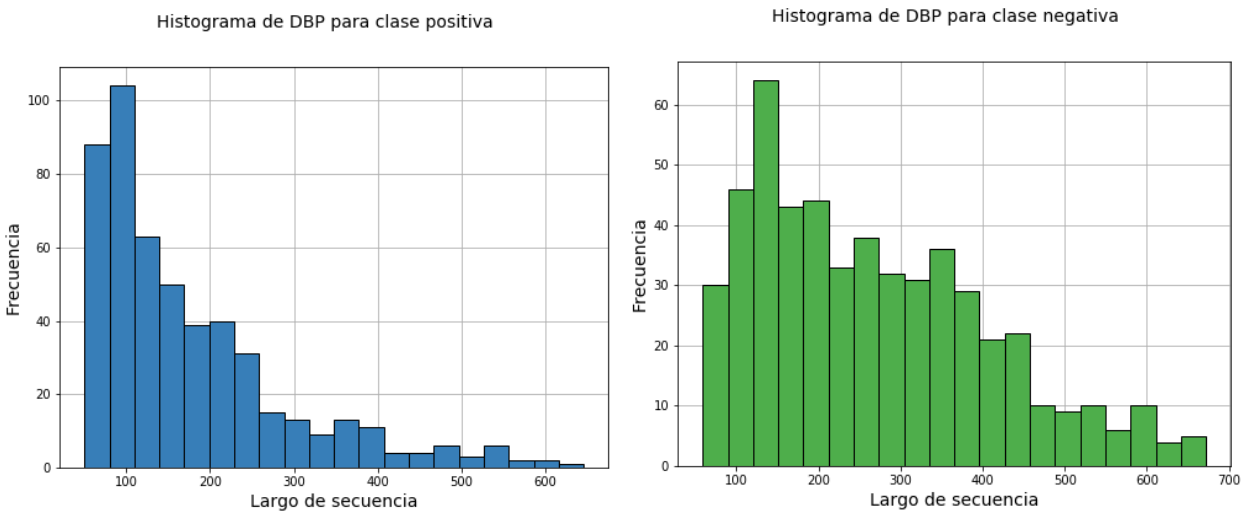


Figura 17: Histogramas de largo de secuencias de las dos clases de DBP para el caso en ausencia de outliers.

El efecto de la eliminación de *outliers* para el conjunto DBP es similar al caso de iAMP-2L. El largo de secuencia máximo se redujo desde un número superior a 2000 a aproximadamente 650. Esto implica que los puntos necesarios para el *zero-padding* disminuyen de 4096 a 1024.

Para poder visualizar mejor la distribución de los largos de secuencias resultantes según las distintas clases de los tres conjuntos de clasificación, se muestran los gráficos de la estimación de la densidad de kernel para cada uno:

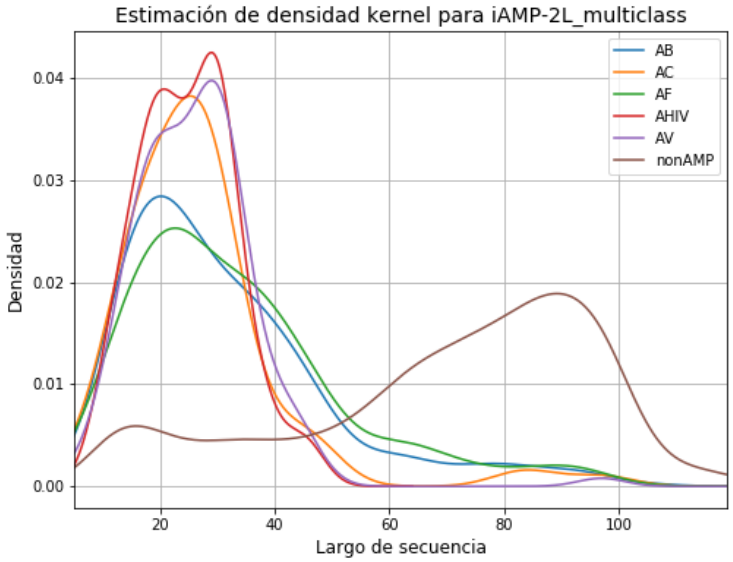


Figura 18: Estimación de la densidad de kernel para el conjunto iAMP-2L_multiclass.

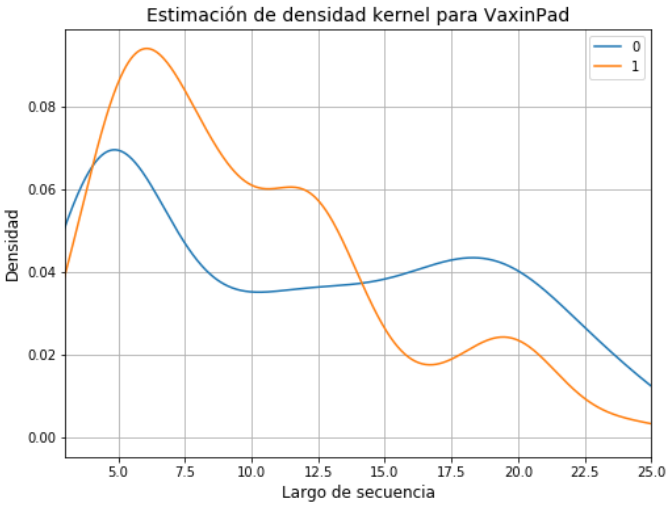


Figura 19: Estimación de la densidad de kernel para el conjunto VaxinPad.

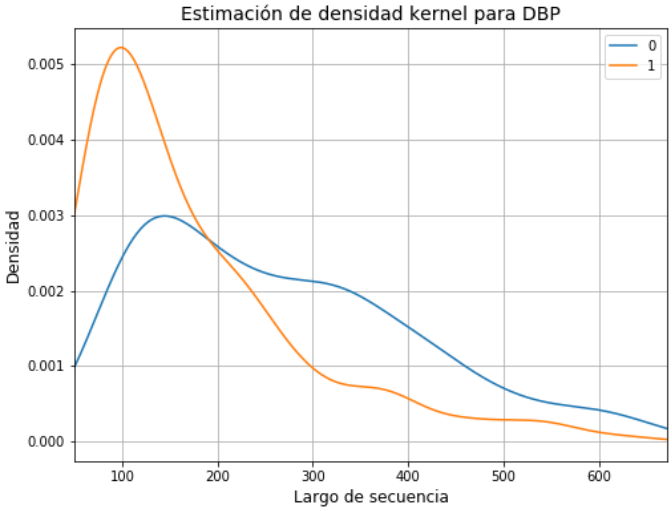


Figura 20: Estimación de la densidad de kernel para el conjunto DBP.

En los tres gráficos presentados (Figura 18 – Figura 20) se marcan distintas particularidades según las clases. Para VaxinPad y DBP, existe una concentración mayor de muestras en rangos bajos de largo de secuencia para las clases positivas, lo cual induciría error de sesgo para las etapas de entrenamiento de los algoritmos predictivos, pues estarán condicionados para extraer más información de las zonas de baja frecuencia de los espectros. Sin embargo, esto está justificado en la literatura, ya que se ha indicado que la información más relevante de las proteínas/péptidos se encuentran en las frecuencias bajas [86]. Para el caso de iAMP-2L_multiclass, existe una marcada tendencia de la clase nonAMP a presentar largo de secuencias mayores a las demás clases, sin embargo, esto puede favorecer a los algoritmos predictivos a discernir mejor los casos de *One vs all* para la clase nonAMP, ya que ésta tenderá a presentar más información a frecuencias más altas.

4.3. Caracterización de las respuestas

En la Tabla 4 se presentan los datos corregidos de la Tabla 3, al tomar en consideración la eliminación de *outliers* para cada conjunto.

Tabla 4: caracterización general sobre largo de secuencias y las respuestas de los conjuntos de datos sin outliers *presentes.*

Subconjunto	Largo de cadenas [residuos]	Clase positiva [%]	Clase negativa [%]	Min	Max	Prom.	STD
AntiTb_primary	5 – 36	50.0	50.0	-	-	-	-
AntiTb_secondary	5 – 36	50.0	50.0	-	-	-	-
ACP-DL	11 – 58	50.8	49.2	-	-	-	-
iACP	11 – 62	45.5	54.5	-	-	-	-
QSP	5 – 52	52.8	47.2	-	-	-	-
iAMP-2L_AB	5 – 107	20.3	-	-	-	-	-
iAMP-2L_AC	5 – 98	3.7	-	-	-	-	-
iAMP-2L_AF	6 – 105	9.6	-	-	-	-	-
iAMP-2L_AHIV	10 – 46	2.3	-	-	-	-	-
iAMP-2L_AV	5 – 98	3.3	-	-	-	-	-
iAMP-2L_nonAMP	5 – 100	60.9	-	-	-	-	-
iAMP-2L_binary	5 – 118	50.1	49.8	-	-	-	-
DBP	50 – 678	49.1	50.9	-	-	-	-
Pop_ara	6 – 34	-	-	1	5.4E+9	5.2E+6	7.3E+7
Pop_chlamy	7 – 53	-	-	1	7.4E+7	2.8E+4	5.0E+5
Pop_yeast	7 – 36	-	-	1	1.6E+11	5.7E+7	8.2E+8
VaxinPad	3 – 25	43.8	56.2	-	-	-	-
Solub	29 – 691	-	-	0.0	147.0	50.0	33.5
Enantioselectivity	389	-	-	4.0	158.0	50.8	33.8
Localization	329 – 361	-	-	-9.51	-3.96	-6.61	1.26
T50	466	-	-	36.0	64.4	51.8	5.9
RT_hela	7 – 39	-	-	1493	6494	3995	1285
RT_yeast	6 – 22	-	-	681	15523	8729	3727

A partir de la Tabla 4 es posible verificar que todos los conjuntos de clasificación a excepción de iAMP-2L_multiclass poseen proporciones balanceadas de sus clases (todos están entre 43% y 57%). Para el caso de iAMP-2L_multiclass, la mayor parte de las muestras corresponden a la clase nonAMP (60.9%). Esto puede generar un error de sesgo importante al momento de entrenar los modelos predictivos, pues éstos estarían condicionados para clasificar mejor los péptidos nonAMP como tal, mientras que tendría problemas de clasificar correctamente las demás clases. En un caso hipotético, si el modelo solo predijera la clase nonAMP para todo el conjunto de prueba, la medida de desempeño *accuracy* indicaría algo aproximado a 60%, mientras que las demás medidas (*precision*, *F1-score*, *recall*, MCC) entregarían peores resultados. No obsante, para poder enfrentar el problema del desbalance de clases es posible asignarle pesos distintos a cada clase en la etapa de entrenamiento. De esta forma, se evita tener que amplificar las muestras de las clases subrepresentadas, lo cual provocaría mayor consumo de recursos computacionales durante la etapa de entrenamiento.

A continuación, se presentan los histogramas para las respuestas de los conjuntos de regresión enantioselectivity y t50:

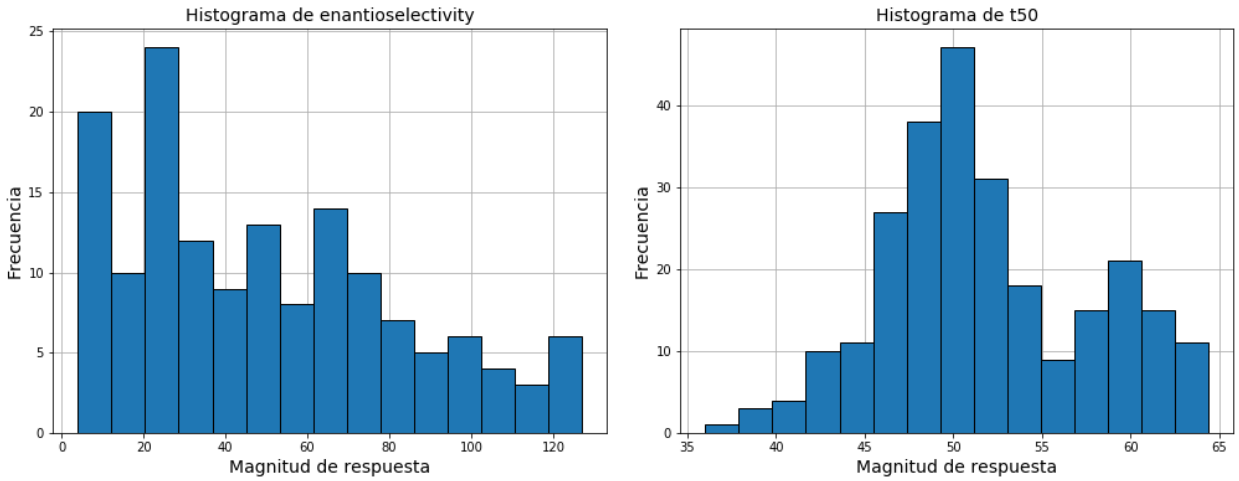


Figura 21: Histograma de la magnitud de respuesta para los conjuntos de (a) enantioselectivity y (b) t50.

En las Figura 21 es posible evidenciar una cantidad nula de *outliers* para enantioselectivity y un pequeño grupo de *outliers* en las regiones bajas para t50. Si bien es posible eliminar los *outliers* de respuestas, no se realizará, ya que se desea verificar la capacidad de generalización a casos extremos también. En los conjuntos de datos de Pop_ara, Pop_chlamy y Pop_yeast es posible observar la importancia de la existencia de una porción pequeña de grupos fuera del margen normal de los datos (Anexo C). En estos últimos, lo que se desea predecir es la observabilidad de péptidos en un especie de microorganismo en específico: los valores altos de observabilidad indican ausencia del péptido mientras que valores bajos indican gran probabilidad de que exista.

4.4. Composición aminoacídica de los conjuntos

Entre los tipos de codificaciones que se estudian en este trabajo de título figura la composición de aminoácidos (AAC). Para poder evaluar si preliminarmente es posible extraer información del AAC, se disponen de gráficos de barra que describen al conjunto entero. Esto sólo se muestra para los conjuntos de clasificación, pues no aportan información relevante en los problemas de regresión debido a que no es posible visualizar una comparación a lo largo de la magnitud de respuestas.

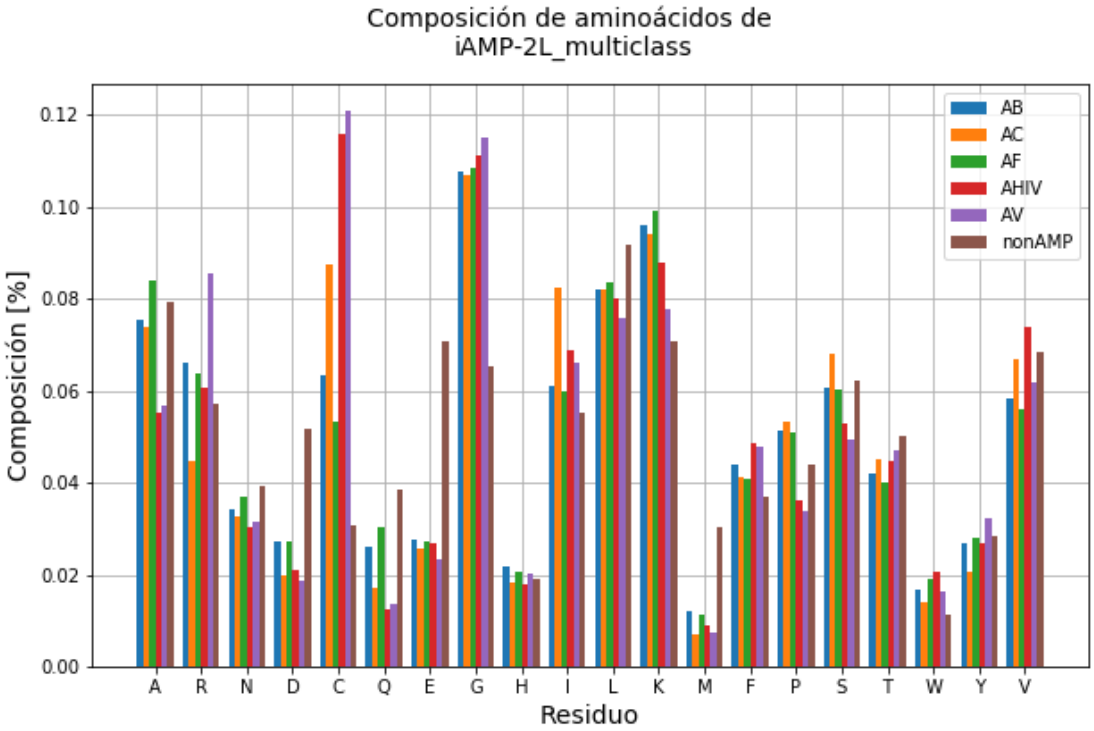


Figura 22: Composición de aminoácidos global para el conjunto iAMP-2L_multiclass.

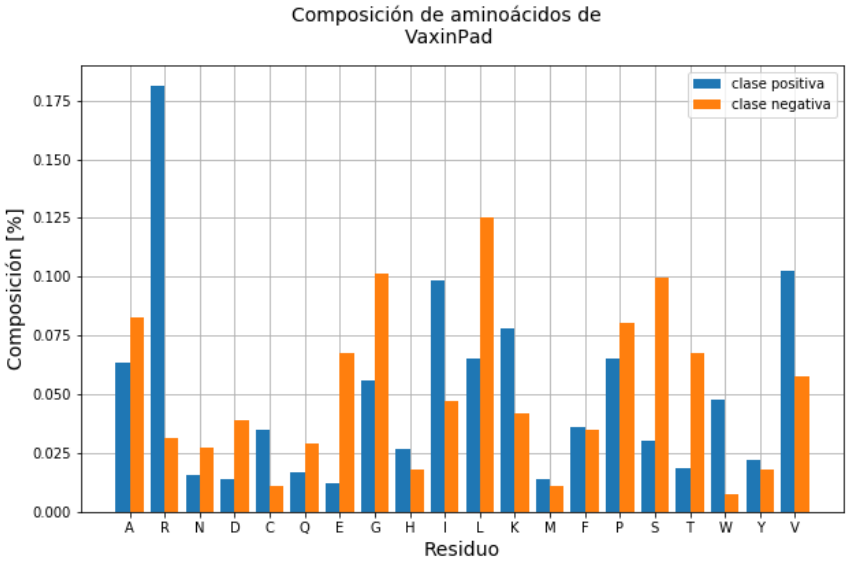


Figura 23: Composición de aminoácidos global para el conjunto VaxinPad.

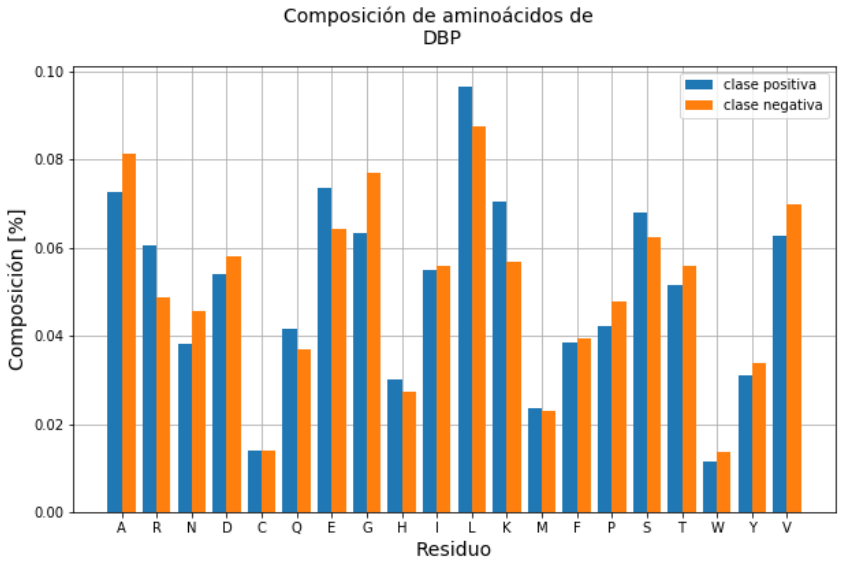


Figura 24: Composición de aminoácidos global para el conjunto DBP.

De la Figura 22 se hace dificultoso la identificación preliminar de patrones de composición aminoacídica que permita tener una idea de la distinción entre las distintas clases de péptidos antimicrobianos. Sin embargo, se puede observar que la clase nonAMP tiene composiciones notoriamente distintas para los aminoácidos D, E, G y M. Esto facilitaría la partición de los conjuntos de péptidos no-antimicrobianos y antimicrobianos.

En cuanto a la Figura 23 (VaxinPad), existe un exceso de aminoácidos de arginina (R) para la clase positiva. Esto se condice con la literatura, pues se ha reportado que los péptidos inmunomoduladores poseen mejor mecanismo de acción al presentar residuos adicionales de arginina en su composición terminal [87]. Si bien esto implica mayor facilidad de separación de los conjuntos, es importante destacar que los gráficos presentados corresponden a composiciones globales de las distintas clases. Para poder tener una idea más acertada de la utilidad de la composición aminoacídica, se podría elaborar distintas estimaciones de densidades de kernel para cada residuo e identificar aquel aminoácido que permita una mejor separación.

Para el caso de DBP, no se observan diferencias significativas para poder declarar útil la composición aminoacídica en una primera instancia. Para proteínas (i. e. no péptidos), es esperable que se presente mayores obstáculos para diferenciar conjuntos según composición aminoacídica, pues dependiendo de la cualidad en particular, es posible que se logren características similares con *motifs* distintos [88]. Explicado con otras palabras, las regiones más relevantes de las proteínas pueden diferir notoriamente en los residuos, pero podría conservar alguna propiedad en particular: por ejemplo, que se tienda a conservar aminoácidos polares para lograr algún mecanismo de acción específico. Para ahondar más en este asunto, es posible rehacer los gráficos de composición aminoacídica, pero empleando un alfabeto reducido.

A continuación, se muestran distintos mapas de calor que permiten entregar un perfil de composición de dipéptidos. La razón de optar por este formato de visualización (y codificación) es porque la composición aminoacídica no logra entregar información sobre

la interacción local entre residuos de la secuencia. Esto podría ayudar a comprender qué interacción de aminoácido es necesario para lograr el efecto de la clase en cuestión [89]. Al igual que en el caso de los gráficos de la subsección anterior, sólo se realizaron para los problemas de clasificación.

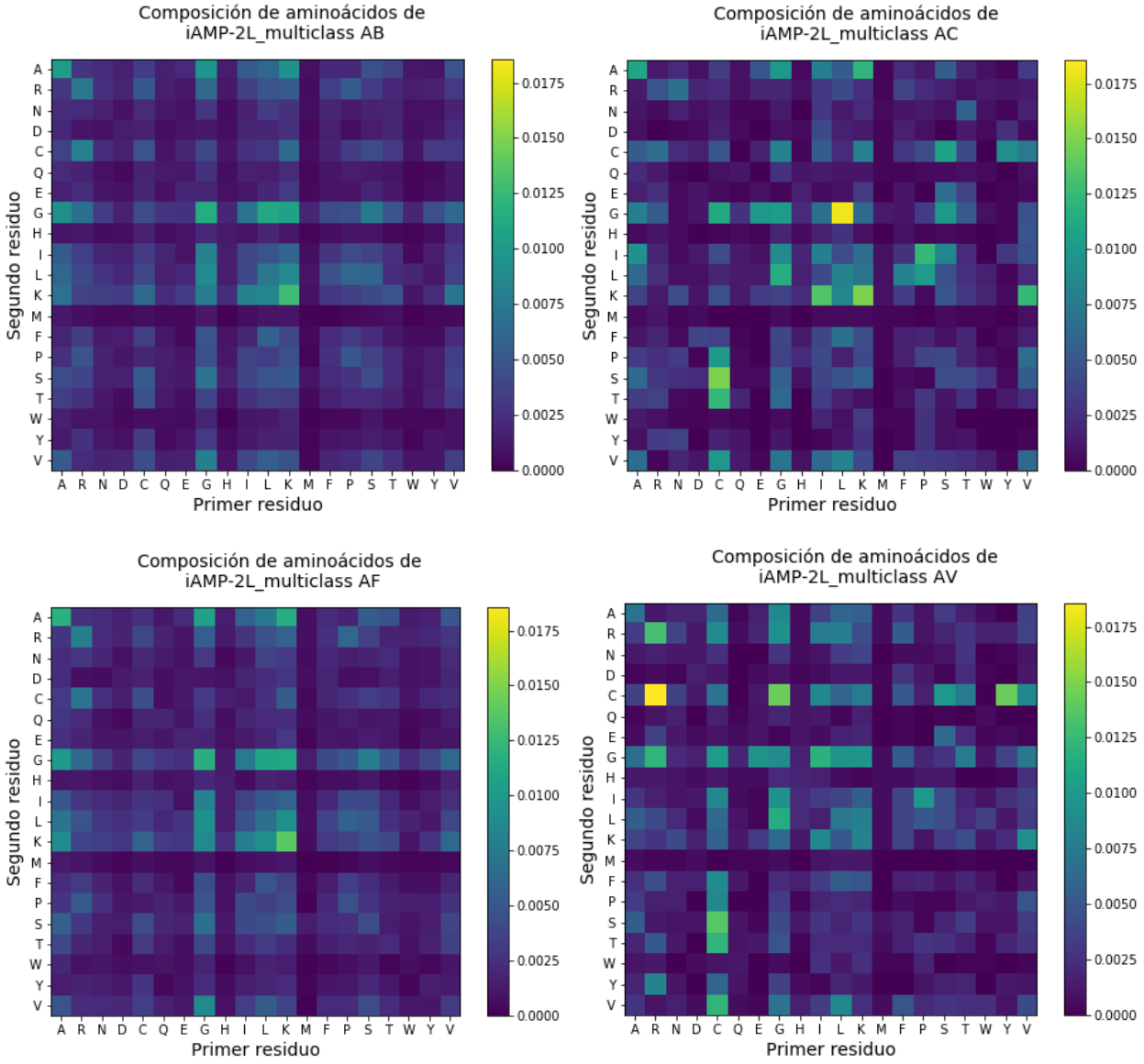


Figura 25: Composición de dipéptidos para las clases AB, AC, AF y AV de iAMP-2L_multiclass.

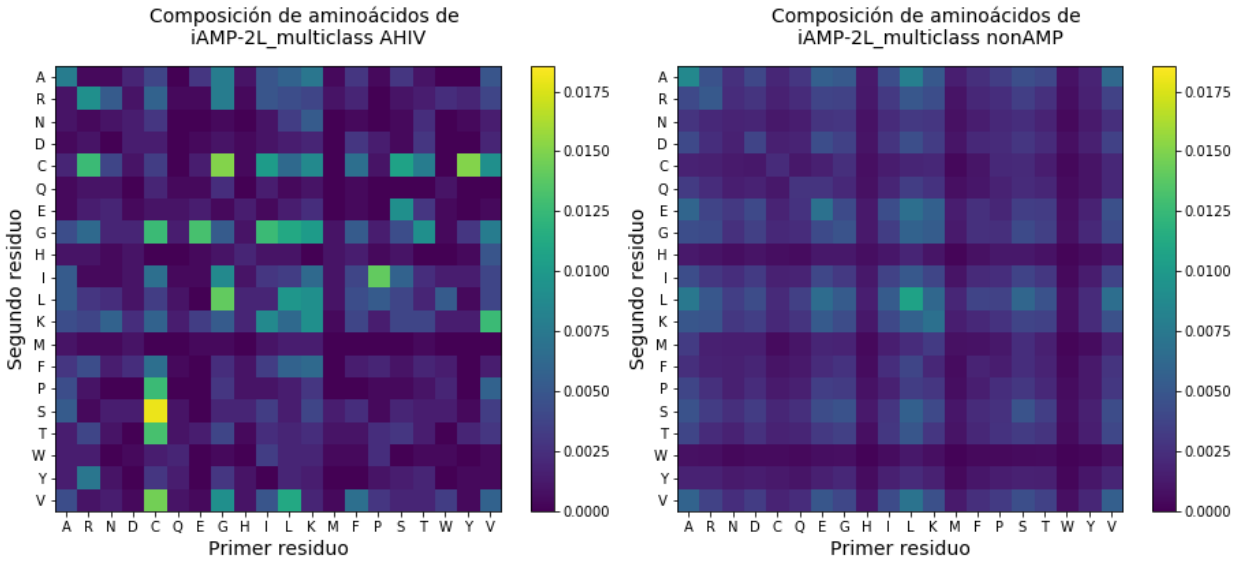


Figura 26: Composición de dipéptidos para las clases AB, AC, AF y AV de iAMP-2L_multiclass.

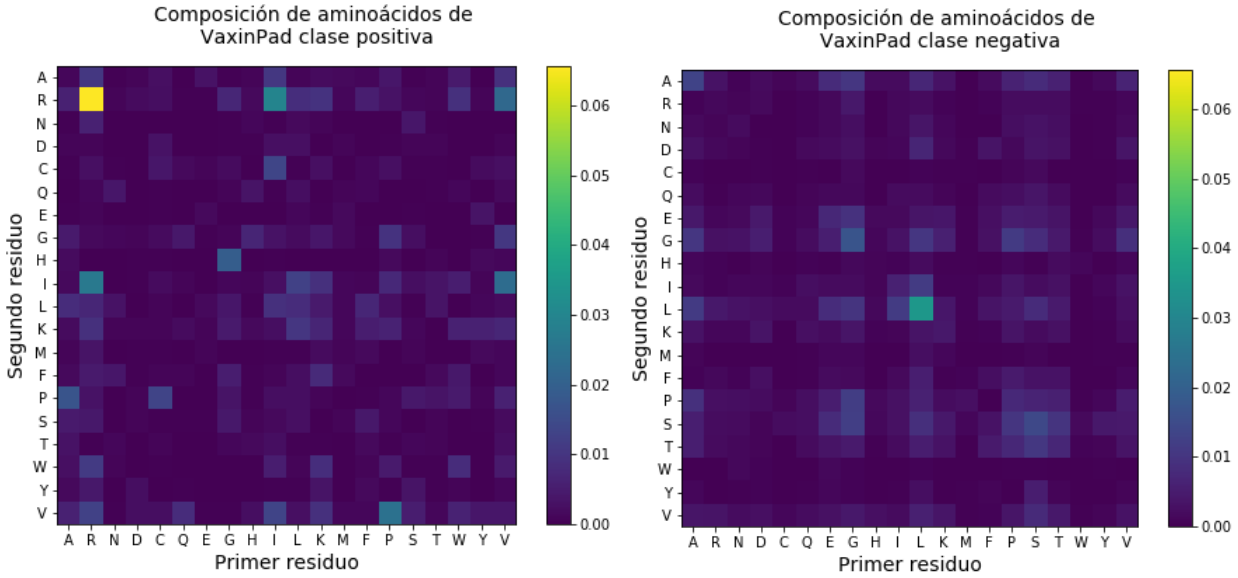


Figura 27: Composición de dipéptidos para las clases de VaxinPad.

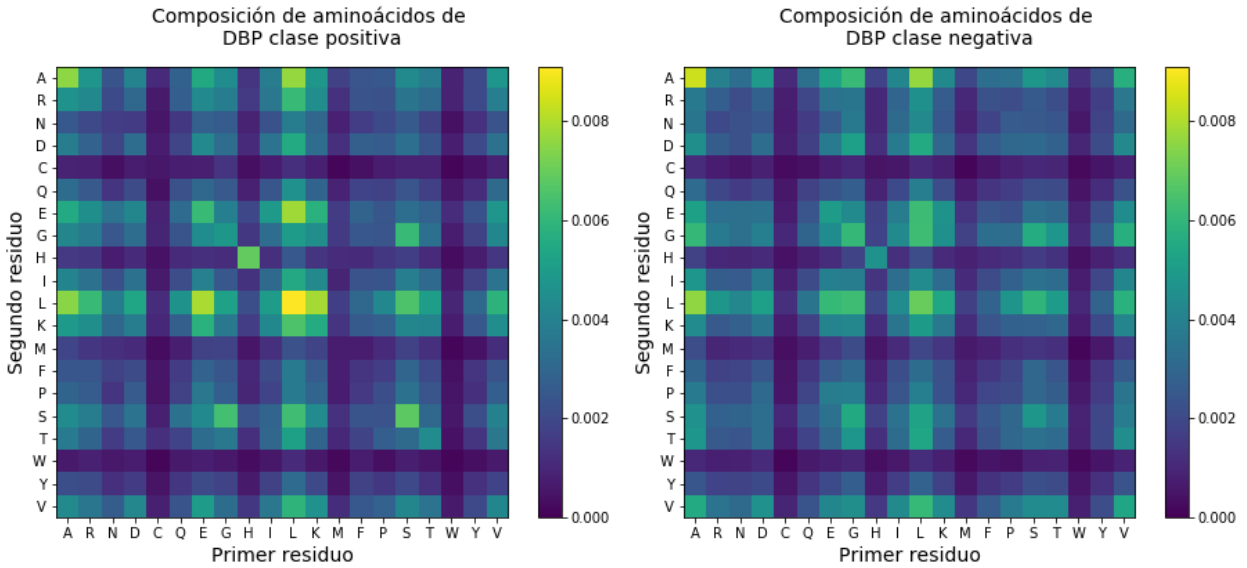


Figura 28: Composición de dipéptidos para las clases de DBP.

En las Figuras 25 y 26 (iAMP-2L_multiclass), es posible extraer información relevante de las clases AC, AV, AHIV y nonAMP. Las tres primeras se diferencian en la tendencia a presentar mayor composición de pares de LG, RC y CS, respectivamente. En cuanto a nonAMP, se observa una distribución relativamente uniforme a lo largo de todo el perfil de la matriz de composición dipeptídica. En contraste con la composición de residuos singulares, este formato es más informativo. Sin embargo, un obstáculo evidente es el perfil casi idéntico entre las clases AB y AF, lo cual introduciría mayores dificultades al momento de alimentar dicha codificación a los algoritmos de aprendizaje.

En cuanto a VaxinPad (Figura 27), se marca aún más la tendencia de la clase positiva a presentar residuos de arginina. En particular, se presenta una distintiva composición mayor de pares de RR. Sin embargo, es posible que este sea el único indicador útil para poder separar las clases, por lo que se introduciría un error de sesgo importante al emplear este formato de codificación.

Por último, la clase DBP (Figura 28) posee distribuciones más similares entre sus clases. En consecuencia, incurre en una peor calidad de entrega de información, similar a su composición aminoacídica singular.

5. Alineamiento

En esta sección se verifica si es posible extraer información relevante de los alineamientos, pues es bien sabido que este herramienta es ampliamente usada en el campo de la biología. Si el alineamiento no logra identificar correctamente las clases/funciones de proteína, implica que el uso de *machine learning* estaría bien fundamentado. Para poder realizar los alineamientos, se utilizó el software MEGA-X (*Molecular Evolutionary Genetics Analysis*). Se optó por el algoritmo MUSCLE para efectuar los alineamientos, pues suele ser más preciso que la opción ClustalW (ambas son las dos opciones que ofrece MEGA-X). La configuración de MUSCLE se dejó en sus parámetros por defecto. Posteriormente, se procedió a determinar las distancias evolutivas entre cada par de secuencia, mediante el modelo de Dayhoff. A continuación, se muestran las matrices de distancias de Dayhoff. Mientras más grande es el valor de la distancia, mayor es la probabilidad de que las dos secuencias sean completamente independientes y, por tanto, no tengan similitud evolutiva.

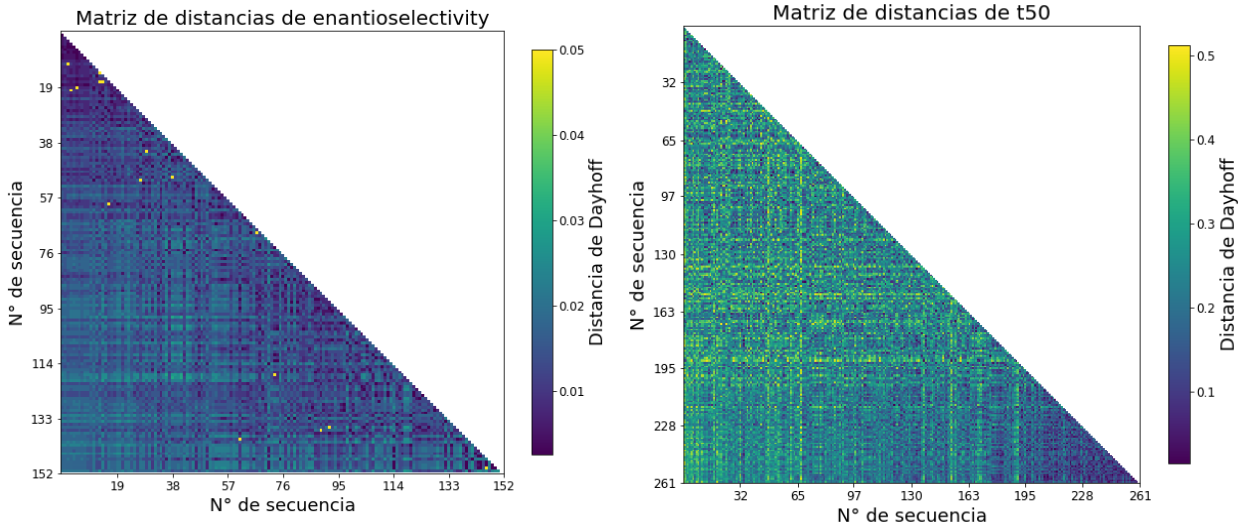


Figura 29: Matrices de distancias de Dayhoff para enantioselectivity y t50. Las secuencias están ordenadas según su respuesta en forma ascendente según su número.

En estos casos, no se observan cambios significativos a lo largo de la matriz, a excepción de la porción inferior-derecha de t50. Esto implicaría que las proteínas con t50 más alto son más similares en su alineamiento que sus demás variantes no-optimizadas. Dado a que no existe ningún patrón observable para el resto de la matriz de t50, la herramienta de alineamiento sólo podría ser útil para discernir cualitativamente entre un valor alto de t50 y uno bajo. Para el escenario de enantioselectivity, no se podría emplear este método. Esto se explicaría debido a que el conjunto de enantioselectivity corresponde a múltiples variantes de 1-8 mutaciones puntuales de una misma proteína, mientras que t50 corresponde a una recombinación quimérica de 3 proteínas. Por lo tanto, se desprende de estas matrices que es improbable encontrar utilidad en el uso de alineamiento para abordar problemas de regresión frente a variantes de proteínas.

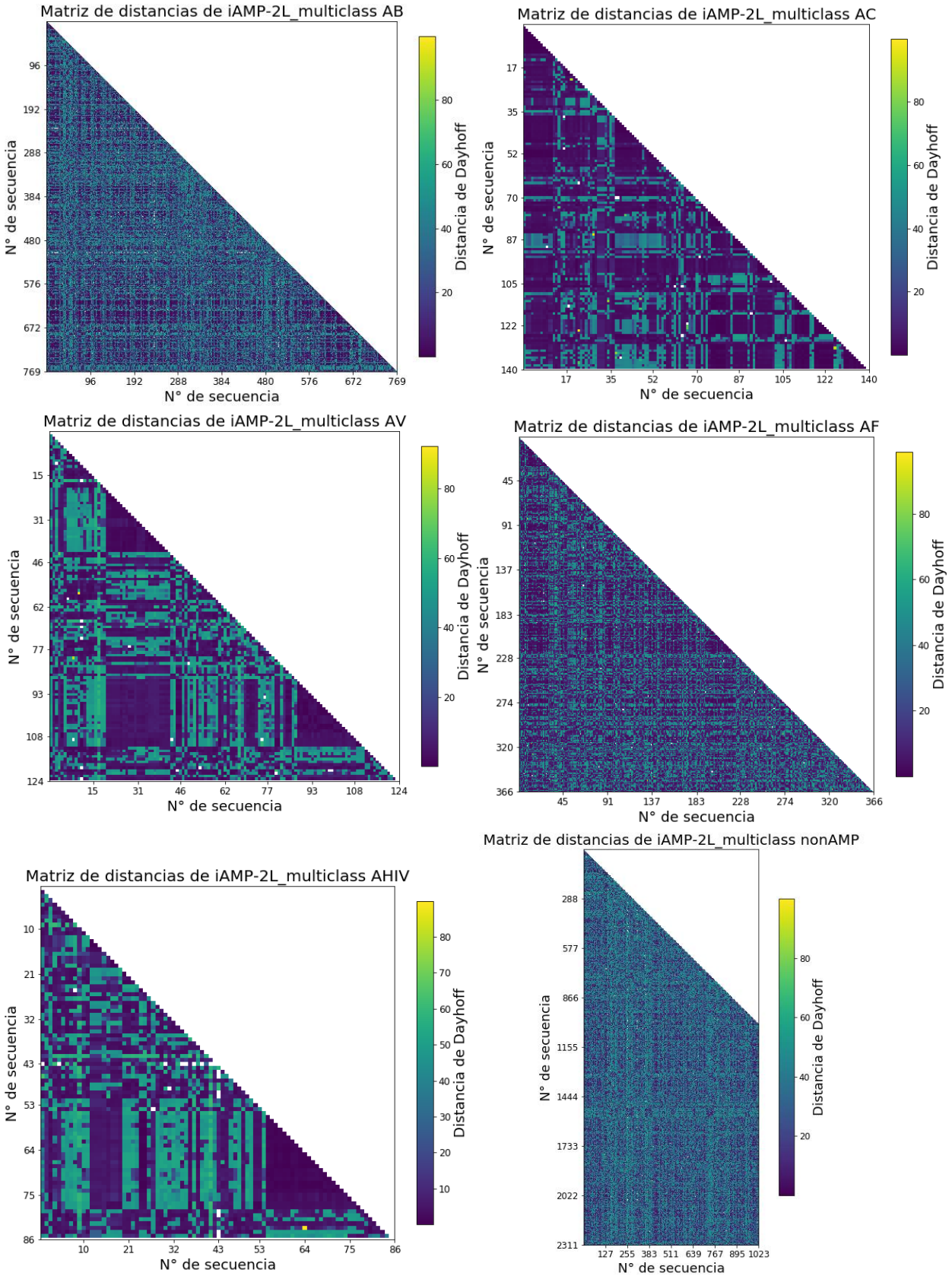


Figura 30: Matrices de distancia de Dayhoff de las clases de iAMP-2L_multiclass. La matriz de la clase nonAMP se muestra cortada debido a que MEGA-X no soporta un gran número de secuencias.

De las matrices de distancias de iAMP-2L_multiclass, sólo se observa cercanía evolutiva en algunas regiones de las clases AC, AV y AHIV. Sin embargo, sus áreas no son suficientes para compensar la confusión que generarían las demás regiones que indican una clara lejanía evolutiva entre otras secuencias. En el caso de AB, AF y nonAMP, la distribución de los valores de sus matrices pareciera dispuestos de manera aleatoria.

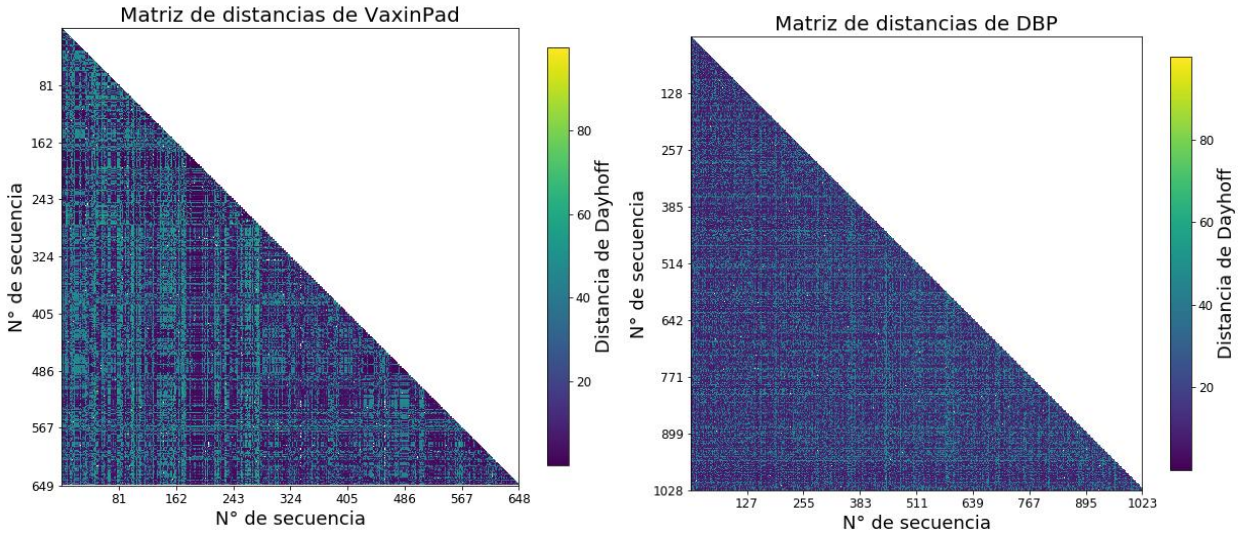


Figura 31: Matrices de distancia de Dayhoff de VaxinPad y DBP. Para ambos casos, la clase positiva se asignó a la primera mitad de las secuencias y la clase negativa para la segunda mitad.

En cuanto a los casos de VaxinPad y DBP, tampoco es posible extraer información a partir del alineamiento. En el caso de que hubiese similitud evolutiva, la porción superior a los triángulos se diferenciaría notoriamente de la porción inferior. Toda esta sección de alineamiento apunta a que es altamente complicado emplear alineamiento de secuencias múltiples para abordar problemáticas sobre conjuntos con poca homología entre sí.

Para complementar la discusión, es posible visualizar las secuencias de logo para cada conjunto. Sin embargo, esta herramienta es mucho más útil para identificar *motifs* presentes en un conjunto de secuencias en lugar de evaluar todo el largo de las cadenas de aminoácidos. Esto último es complicado de realizar en conjuntos heterogéneos de proteínas/péptidos, pero existen modelos de clusterización reportados para poder abordar este obstáculo [90]. De todas maneras, se optó por mostrar el logo de secuencias de VaxinPad a modo de ejemplo, pues este conjunto se forma por cadenas relativamente cortas.

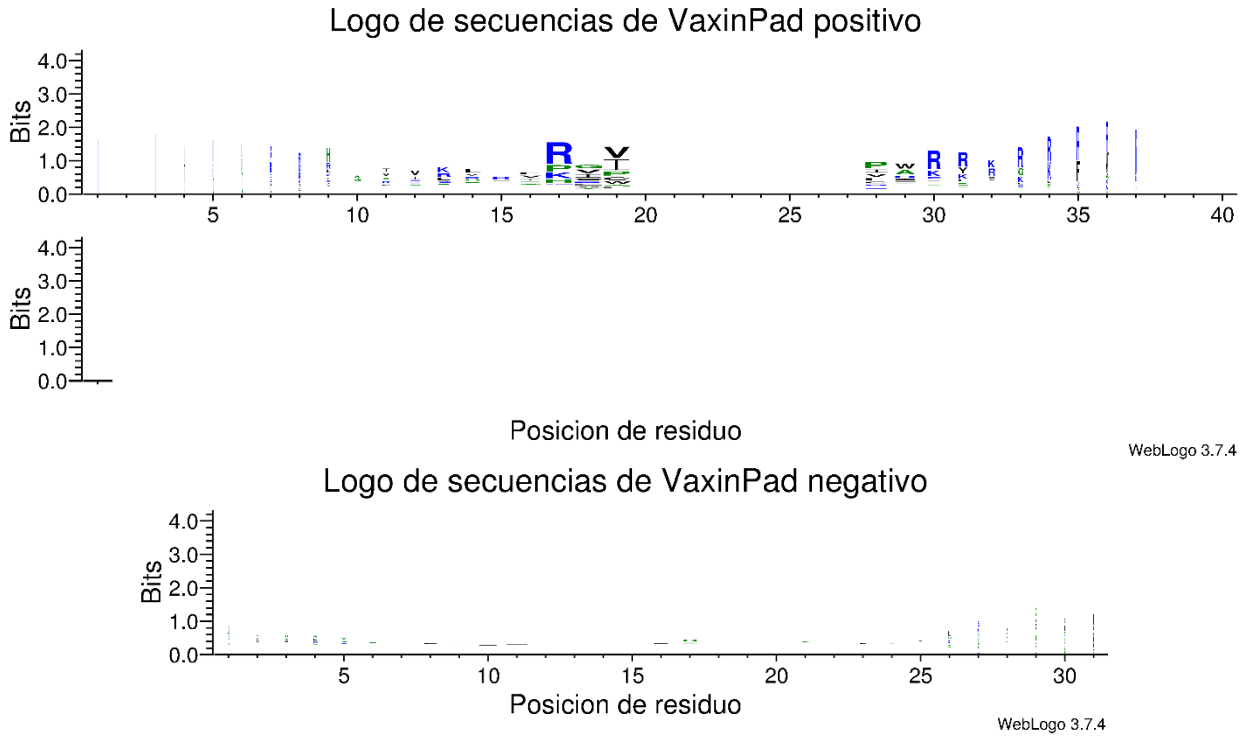


Figura 32: Logo de secuencias para las clases de VaxinPad. Se utilizó WebLogo 3.7.4 para su visualización [91].

De la Figura 32 se desprende que es posible identificar ciertos patrones de secuencias en la clase positiva. En particular, se vuelve a destacar la importante de arginina en la estructura de sus péptidos. En la referencia de donde se extrajo el conjunto de datos de VaxinPad ([83]), llevaron a cabo identificación de *motifs* para determinar las diferencias claves entre el conjunto positivo y el negativo, logrando distintivas apreciaciones para ambas clases. Esto implica que, dependiendo de la complejidad del conjunto de péptidos, es posible extraer información relevante del alineamiento que potencialmente sea suficiente para lograr una clasificación exitosa. Sin embargo, en conjuntos más complejos (proteínas de secuencias más largas), resulta altamente complicado determinar *motifs* y distancia evolutiva a causa de la poca homología existente. A raíz de esto último, resulta más llamativo el uso de algún otro recurso que compense la poca capacidad de los alineamientos y búsqueda de patrones de secuencias para llevar a cabo la labor de clasificación. En el caso de conjuntos destinados para problemas de regresión, el alineamiento queda casi obsoleto al tratar con mutaciones específicas, pues el propósito de éstas es justamente introducir desviaciones evolutivas marginales para lograr un mejor resultado, llámese estabilidad, velocidad de reacción, selectividad, entre otros.

6. Caracterización de espectros

La codificación de las secuencias de cada conjunto se realizó mediante el código en Python (elaboración propia), disponible en Anexo E. A continuación, se realiza un estudio preliminar cualitativo sobre la utilidad de los espectros para cada conjunto, empleando espectrogramas y espectros multicanales. Cabe destacar que se evaluó cada una de las propiedades fisicoquímicas para cada uno de los conjuntos presentes, pero sólo se muestra la propiedad visualmente más informativa en cada caso.

6.1. VaxinPad:

Se determinó que la propiedad fisicoquímica que entrega mejor indicadores visuales corresponde a JOND750101. En la Figura 33 se observa la diferencia entre la clase negativa (0) y positiva (1). Debido a que las secuencias de péptidos de VaxinPad no superan los 32 residuos, se obtienen largo de espectros de 16 [Hz]. La construcción de los espectro multicanales se realizó mediante la superposición de todos los espectros individuales de cada conjunto. De esta manera, se logra evidenciar una distribución de los *peaks* más relevantes y distintivos, si es que lo hubiese. En el caso de JOND750101 para VaxinPad, no se observan diferencias significativas. Sólo se aprecia una sutil tendencia de la clase 0 a presentar magnitudes mayores a frecuencias bajas.

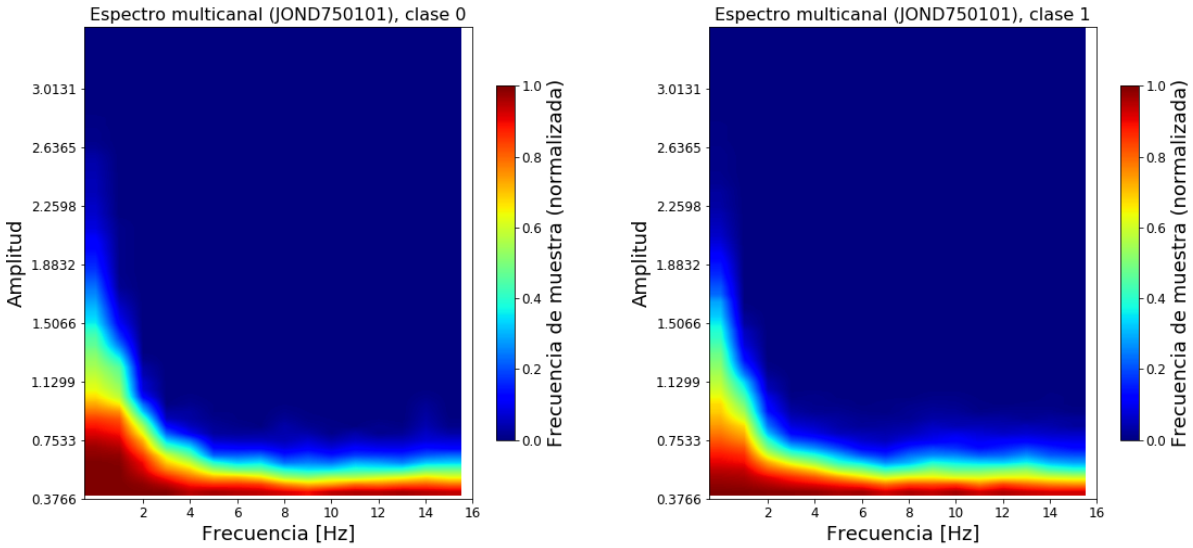


Figura 33: Espectros multicanales para las clases de VaxinPad.

Los espectrogramas se construyen disponiendo en un plano a las frecuencias de cada una de las muestras, similar a como se construyen los espectrogramas de audio, sólo que, en lugar de tener tiempo, se tiene número de secuencias.

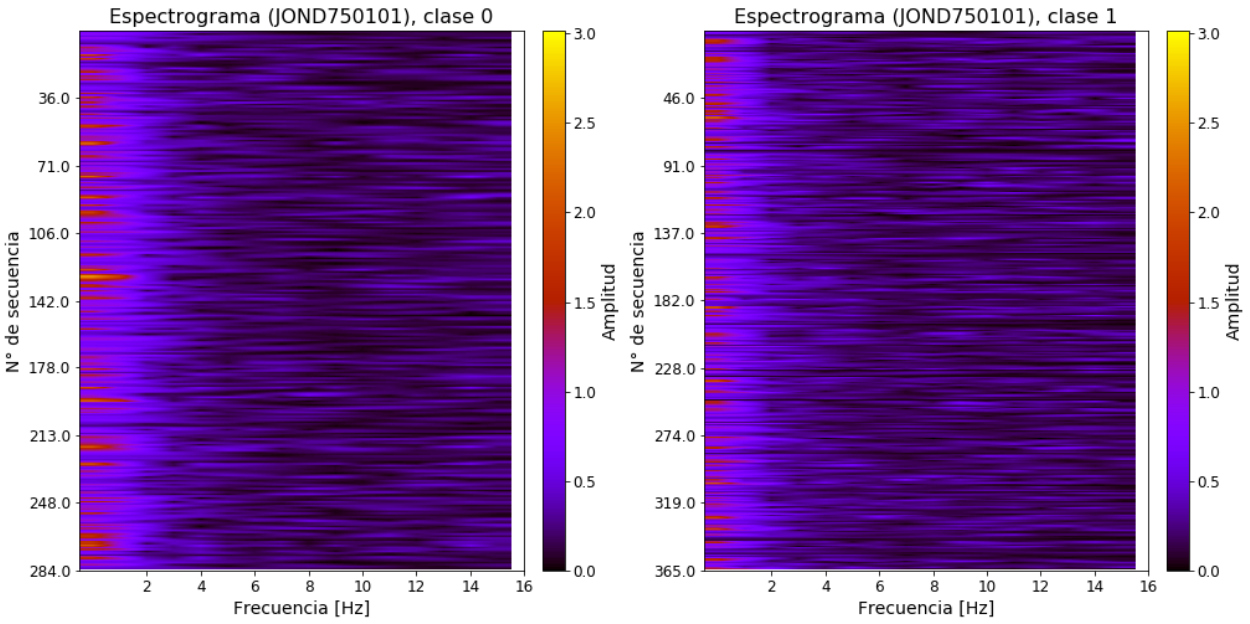


Figura 34: Espectrogramas para las clases de VaxinPad.

De la Figura 34, no es posible extraer información significativa para discernir entre las dos clases. Se podría argumentar que la clase 0 presenta más variaciones en sus amplitudes en los rangos inferiores a 3 [Hz], mientras que la clase 1 se estabiliza mucho mejor en 2 [Hz], pero dicha observación es muy susceptible a apreciaciones del ojo humano. Por lo tanto, no es posible determinar utilidad cualitativa a partir de los espectros de Fourier para VaxinPad.

6.2. DBP

Para el caso de DBP, se empleó la propiedad fisicoquímica COSI940101. Debido a que el largo de secuencia máxima de DBP es aproximadamente de 700 residuos, se tienen 512 frecuencias. Para poder evaluar mejor la calidad de información de los espectros, se decide observar por separado las zonas de baja frecuencia y alta frecuencia. Los gráficos de baja frecuencia se generan simplemente truncando todos los valores posterior a 16 [Hz], mientras que los de alta frecuencia se obtienen eliminando los peaks más grandes: se consideran *outliers* mediante el método IQR.

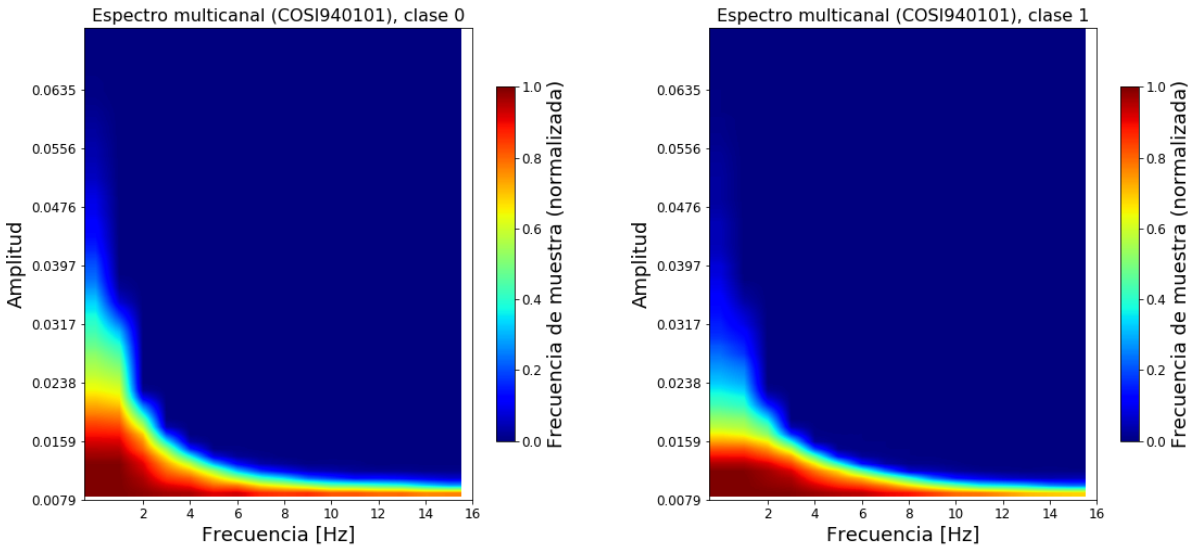


Figura 35: Espectros multicanalesde baja frecuencias para las clases de DBP.

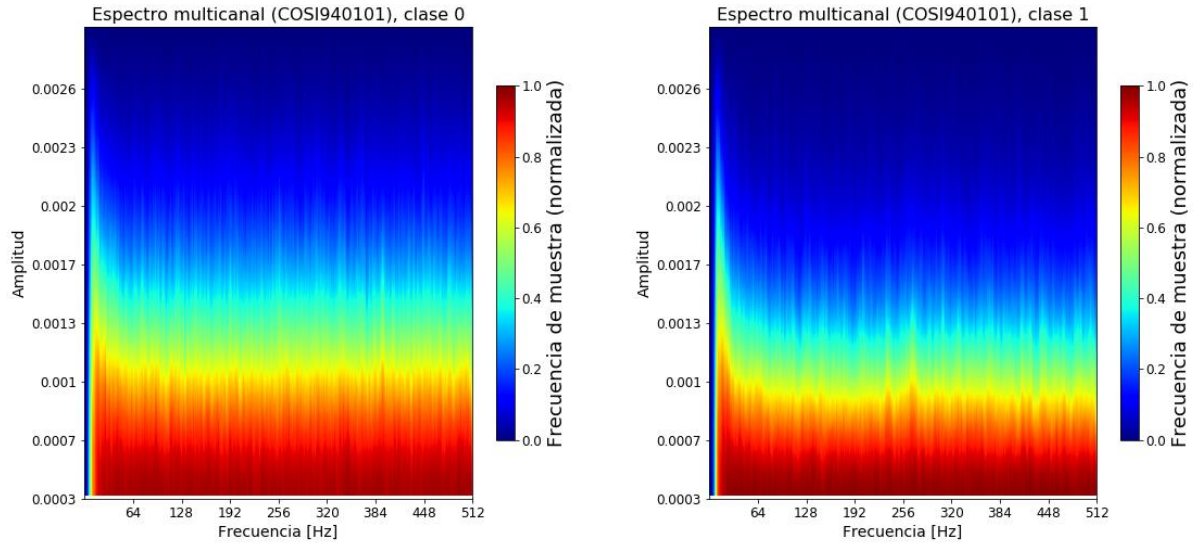


Figura 36: Espectros multicanales de alta frecuencias para las clases de DBP.

Es posible observar algo similar que en el caso de VaxinPad para las frecuencias bajas, pero en DBP se acentúa más esta diferencia de magnitudes de los *peaks* entre las dos clases. En cuanto a las frecuencias altas, el nivel basal de magnitud para la clase 1 es ligeramente menor que la clase 0. Si bien se nota una diferencia, sigue siendo leve y puede generar dificultades al momento del entrenamiento. Para los espectrogramas, sólo se muestran las regiones de baja frecuencias debido a que presentan una distinción más sustancial.

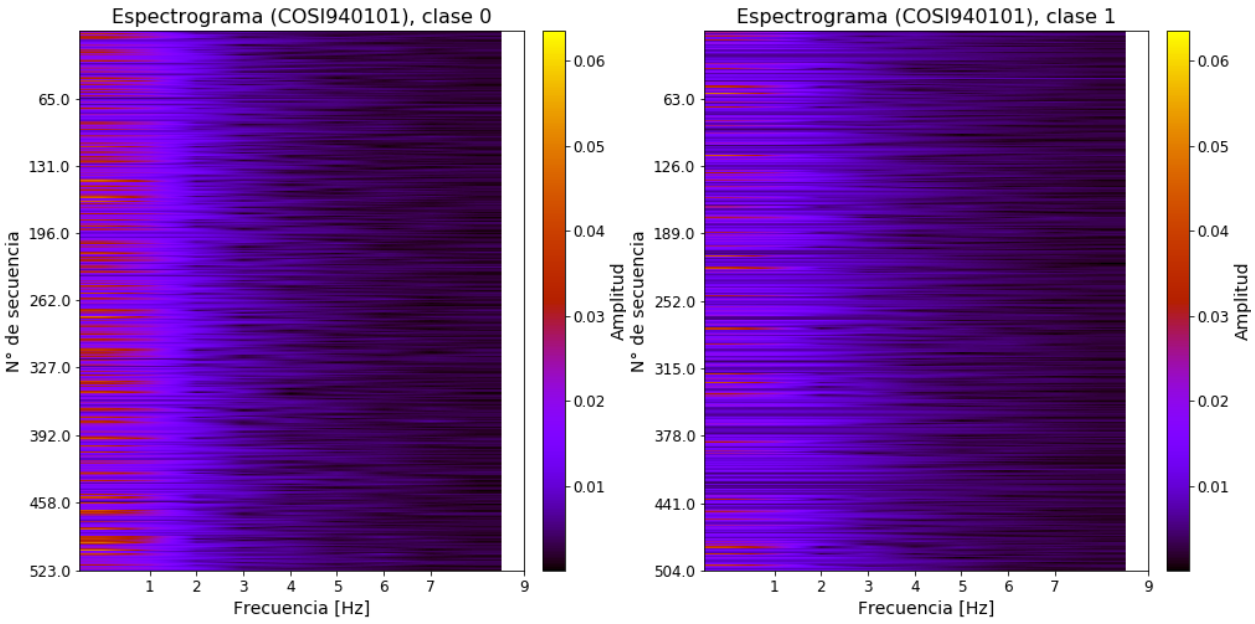


Figura 37: Espectrogramas de baja frecuencias para las clases de DBP.

En este caso, se aprecia una diferencia mucho más marcada, la cual corrobora el análisis del espectro multicanal: las amplitudes máximas en las zonas de frecuencias más bajas son menor en la clase 1.

6.3. iAMP-2L_multiclass

Se utilizó la propiedad HOPT810101 (hidrofobicidad) para este conjunto. No se observó diferencia alguna en las frecuencias bajas, por lo que se muestran los espectros en todo el rango de frecuencias. Se disponen en las siguientes dos páginas para facilitar la visualización.

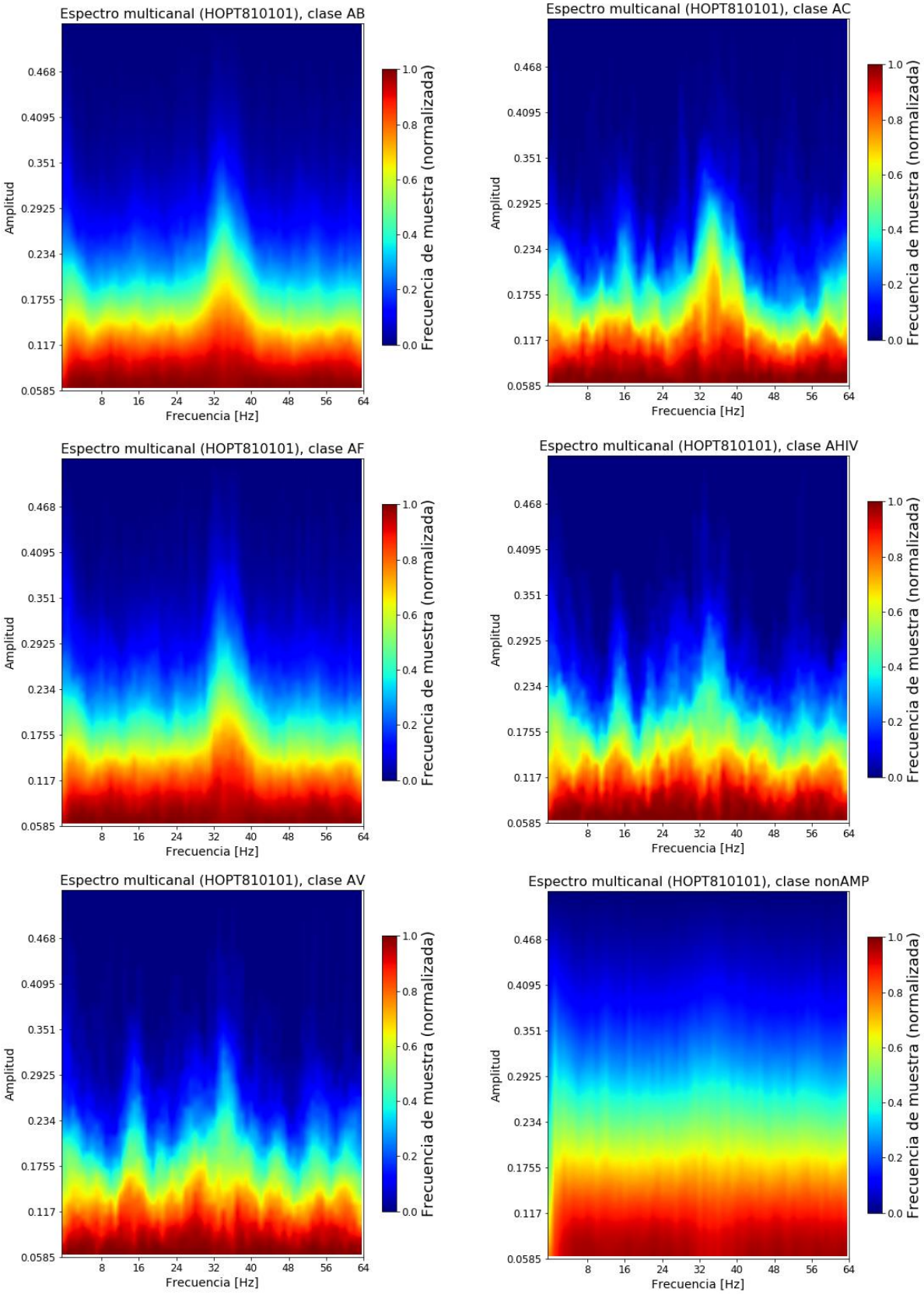


Figura 38: Espectros multicanales para las clases de iAMP-2L_multiclass.

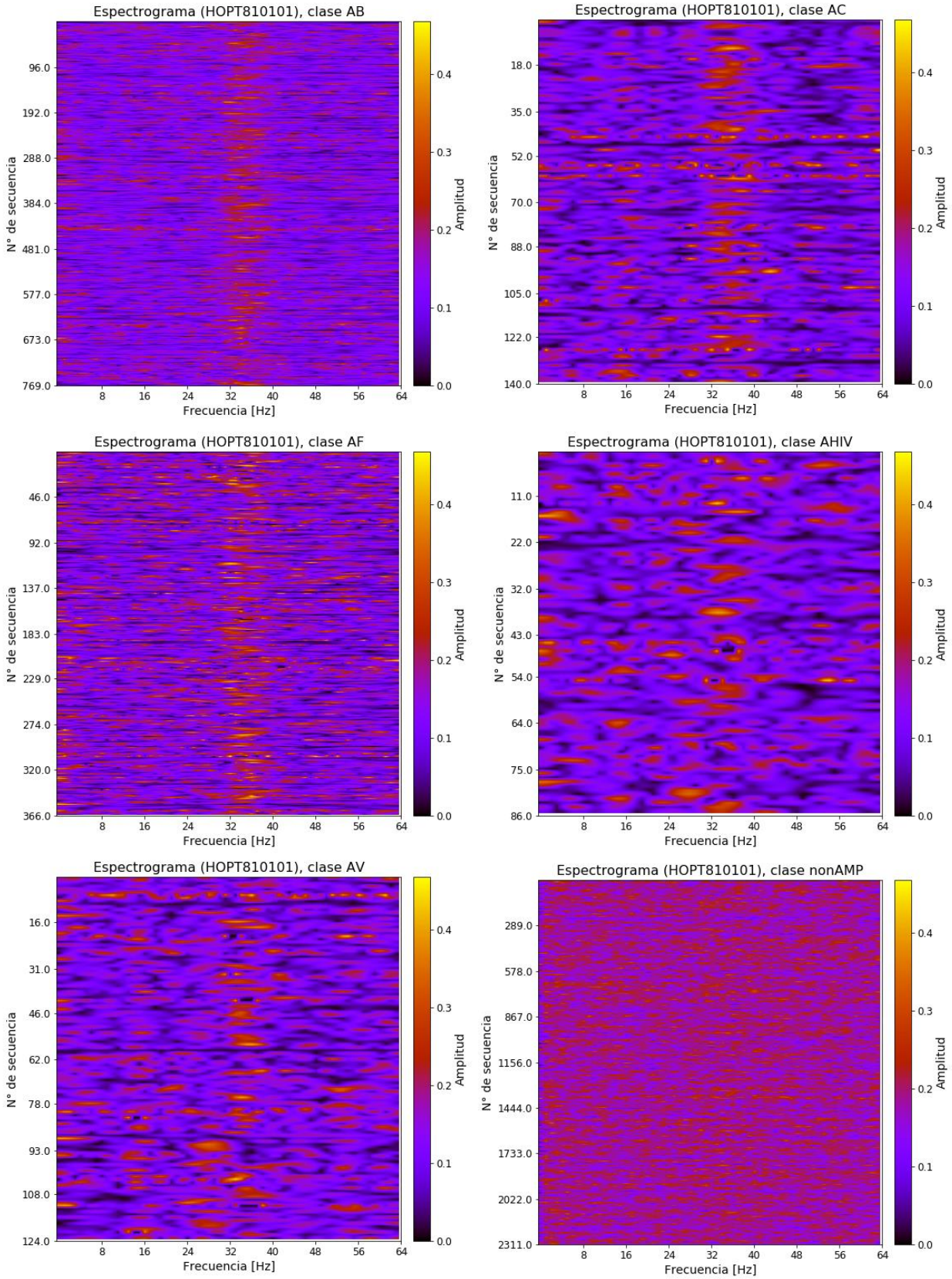


Figura 39: Espectrogramas de las clases de iAMP-2L_multiclass.

En la Figura 38 es posible identificar un *peak* distintivo para las clases AB, AC y AF. En cuanto a las clases AHIV y AV, no se aprecia el mismo *peak*, pero se nota un comportamiento similar entre estas dos clases, lo cual se podría explicar debido al tipo de péptido antimicrobiano que corresponden: AHIV es destinado a combatir el VIH, mientras que AV es un grupo de antivirales general. Para nonAMP, este *peak* se pierde por completo y la distribución de los espectros adquiere un comportamiento más uniforme, en particular, manifiesta una mayor varianza en sus niveles (explicado por el área difusa alrededor del nivel basal). En la Figura 39 se exacerba aún más lo observado anteriormente, y se corrobora la uniformidad de los espectros para el caso nonAMP. Esto implicaría que los espectros serían capaces de diferenciar entre algunas clases generales, pero presenta dificultades para discernir entre grupos específicos.

La evaluación de la propiedad de hidrofobicidad en este caso está fundamentada por la literatura, la cual indica que los péptidos antimicrobianos poseen, en general, un porcentaje importante de aminoácidos hidrofóbicos [92], los cuales entregan capacidad a los péptidos de adoptar estructuras anfipáticas para atravesar membranas lipídicas. Esto sustenta el hecho de que los espectros de nonAMP exhiban una varianza mucho mayor a las demás clases.

6.4. Enantioselectivity:

Se optó por graficar los espectros de la propiedad COSI940101. En la Figura 40 se aprecian los espectros multicanales en las frecuencias bajas y altas, respectivamente. Se debe recordar que enantioselectivity es un conjunto de variantes producto de mutaciones puntuales, por lo que los espectros tenderán a ser muy similares entre sí. Sin embargo, se puede capturar esta diferencia, lo cual se ve reflejado en lo difuso de los bordes de los espectros. Si bien esto indica que existe “algún” cambio, la magnitud puede no ser suficiente para el algoritmo de aprendizaje.

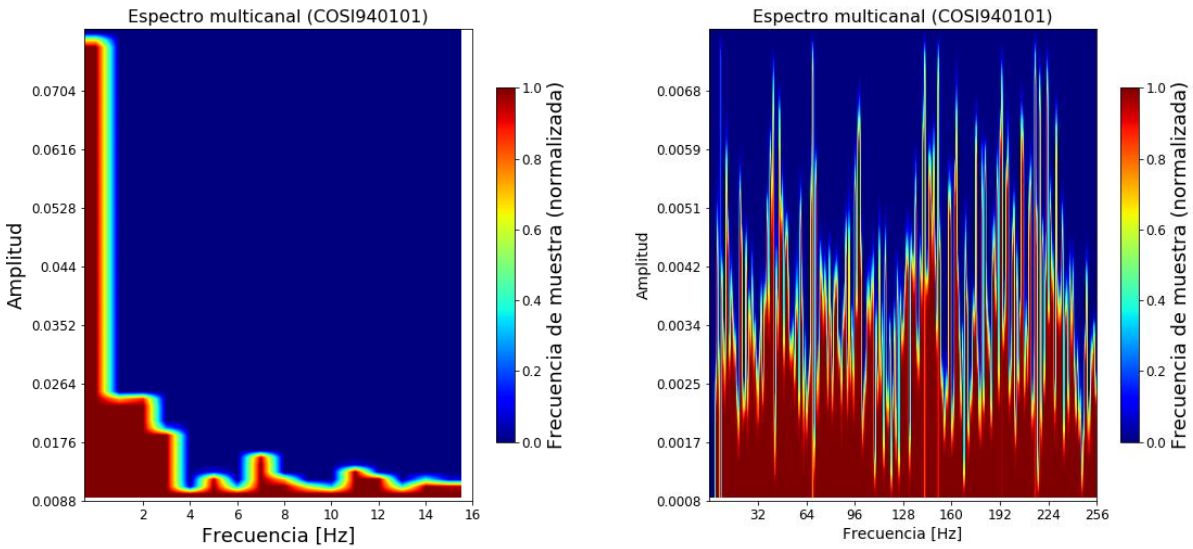


Figura 40: Espectros multicanales para enantioselectivity, para (izquierda) frecuencias bajas y (derecha) frecuencias altas.

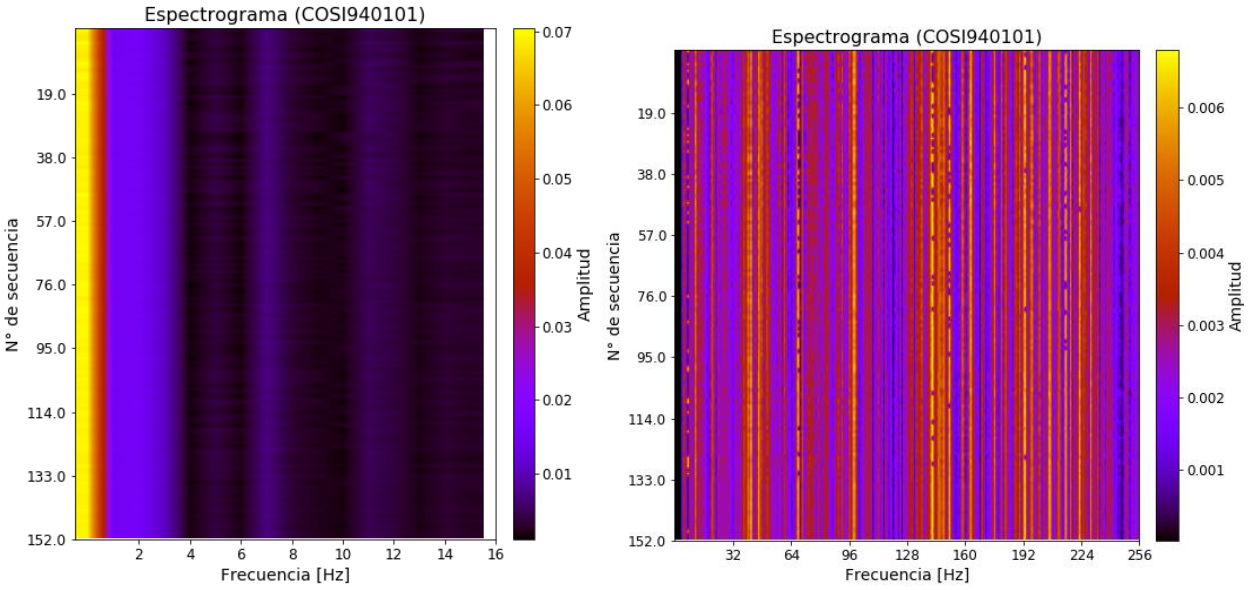


Figura 41: Espectrogramas de enantioselectivity para (izquierda) frecuencias bajas y (derecha) frecuencias altas.

La Figura 41 respalda lo dicho anteriormente: en el espectrograma de frecuencias altas existen ligeros cambios a lo largo del gráfico. Cabe destacar que para la construcción de espectrogramas de conjuntos de regresión, se ordenó a las secuencias para lograr un comportamiento ascendente de la respuesta según número de secuencia.

6.5. T50

Para T50, se escogió la propiedad HOPT810101. Esencialmente, se tienen las mismas conclusiones que en el caso de enantioselectividad, pero se puede notar mejor la dispersión de las magnitudes de los espectros a distintas frecuencias. Esto implica que sería más sencillo entrenar un algoritmo de aprendizaje para predecir las respuestas del conjunto t50 que las del conjunto de enantioselectivity.

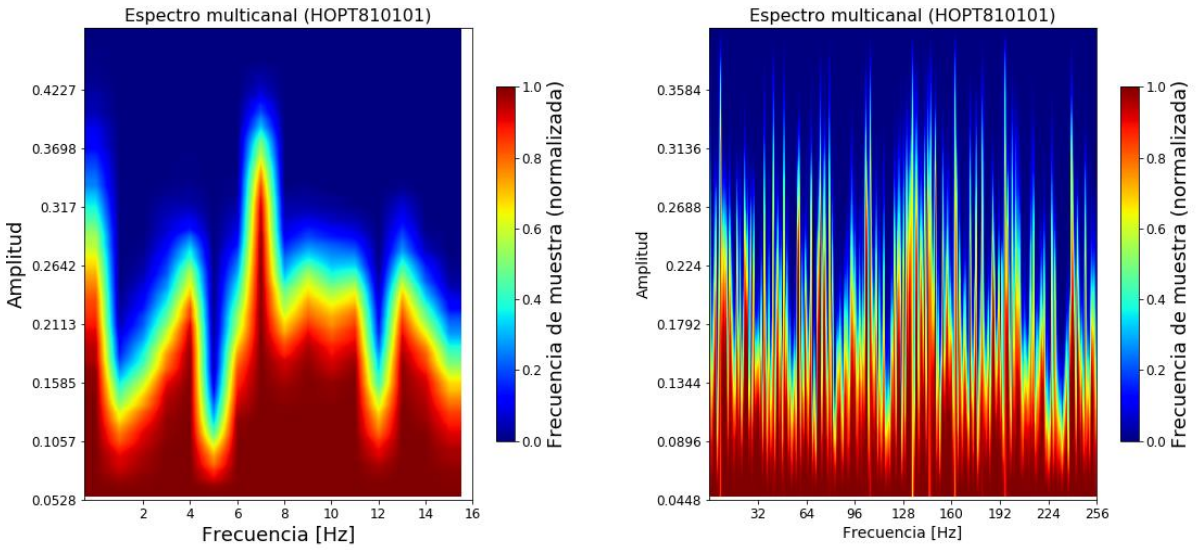


Figura 42: Espectros multicanales de t50, para (izquierda) frecuencias bajas y (derecha) frecuencias altas.

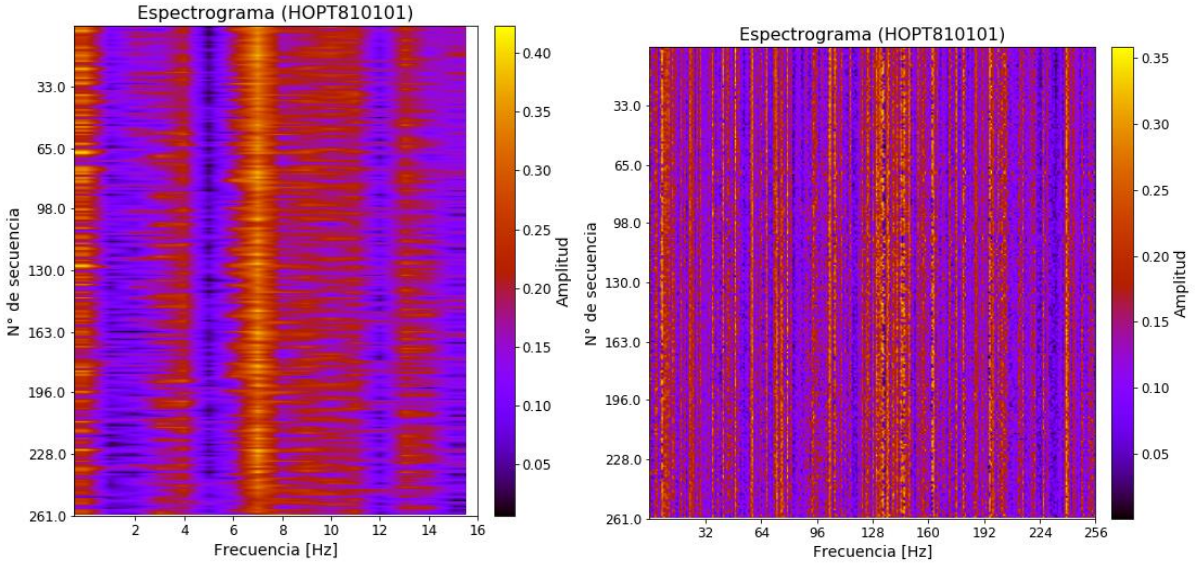


Figura 43: Espectrogramas de t50 para (izquierda) frecuencias bajas y (derecha) frecuencias altas.

7. Modelos predictivos

La última sección de este trabajo involucra probar si efectivamente los espectros de Fourier son válidos para construir un modelo competente. Para ello, se codificaron cada una de las secuencias de los conjuntos según *onehot*, ordinal (a cada aminoácido se le asigna un número único entero), composición aminoacídica, y composición de dipéptidos sin extracción de características. Primeramente, se determinó cuál es la propiedad fisicoquímica más informativa para cada conjunto mediante el uso de una red neuronal convolucional. Esta CNN consta de las siguientes capas:

- Capa de entrada
- Capa convolucional 1D con 256 filtro con kernel de 3 unidades, sin strider, con función de activación ReLU
- Capa *max-pooling* de 2 unidades, sin strider
- Capa *dropout* con una probabilidad de *dropout* del 20%
- Capa convolucional 1D de 128 filtros con kernel de 3 unidades, sin strider, con función de activación ReLU
- Capa *max-pooling* de 2 unidades, sin strider
- Capa *dropout* con una probabilidad de *dropout* del 20%
- Capa *flatten*
- Capa densa de 16 neuronas con función de activación ReLU
- Capa de salida

La arquitectura de esta red se basó en la referencia [74], con algunas modificaciones que se determinaron de manera iterativa empleando el conjunto DBP para la selección de hiperparámetros, mediante validación cruzada *5-fold*. Las propiedades fisicoquímicas más informativas se obtuvieron luego de probar la CNN con cada una de ellas usando validación cruzada *5-fold* para los conjuntos con más de 500 secuencias (VaxinPad, IAMP-2L_multiclass y DBP) dejando un 20% de datos aparte; y *10-fold* para los conjuntos con a lo más 500 secuencias (enantioselectivity y t50) dejando un 30% de datos aparte. Los resultados se aprecian en la Tabla5.

Tabla 5: Selección de propiedades fisicoquímicas más informativas para cada conjunto. El valor de cada celda es el Accuracy [%] y R2, para los conjuntos de clasificación y regresión, respectivamente. Las celdas destacadas corresponden a la propiedad fisicoquímica informativa del conjunto en cuestión.

	iAMP-2L_multiclass	VaxinPad	DBP	enantio.	t50
PRAM900102	68.91 ± 1.35	77.66 ± 2.25	68.08 ± 1.67	-0.089 ± 0.013	0.004 ± 0.003
PRAM900103	70.26 ± 0.98	78.22 ± 3.01	69.18 ± 1.15	-0.010 ± 0.014	0.001 ± 0.003
COSI940101	68.31 ± 1.63	71.69 ± 5.02	62.84 ± 11.16	0.003 ± 0.003	0.514 ± 0.360
HOPT810101	69.47 ± 1.67	74.37 ± 4.06	65.78 ± 2.75	0.110 ± 0.239	0.796 ± 0.096
JOND750101	68.51 ± 1.85	79.20 ± 2.26	67.48 ± 2.49	-0.024 ± 0.045	0.516 ± 0.341
RADA880106	68.79 ± 1.54	81.31 ± 2.82	66.04 ± 9.86	-0.007 ± 0.010	0.003 ± 0.006
GRAR740103	69.53 ± 1.78	83.43 ± 1.83	67.11 +- 2.60	-0.011 ± 0.013	0.005 ± 0.006
FASG760101	67.42 ± 1.13	79.13 ± 3.37	67.48 ± 3.35	-0.008 ± 0.017	0.008 ± 0.004

Si bien estas propiedades no corresponden a las mismas analizadas en la sección anterior, es necesario recordar que la caracterización de los espectros se realiza para prever potenciales resultados para esta etapa. Existen comportamientos y relaciones entre

variables que los humanos no son capaces de notar, y es aquí donde la propuesta del *machine learning* se vuelve relevante.

Después de identificar las propiedades fisicoquímicas a emplear en los conjuntos de datos, se procede a optar por un algoritmo de aprendizaje que sea útil para validar la digitalización. En la Tabla 5 se puede notar que los desempeños de los conjuntos de regresión son bajos, lo cual puede atribuirse a que una CNN podría no ser adecuado para el tipo de problema. Se disponen de otros cuatro algoritmos posibles: *Artificial Neural Network* (ANN), *Random Forest* (RF), *Support Vector Machine* (SVM) y *K-Nearest Neighbor*. La arquitectura de la ANN se compone de 3 capas ocultas de 256, 128 y 16 neuronas, todas con función de activación ReLU y separadas por una capa de *dropout* del 20%. Para determinar los parámetros de cada algoritmo, se emplea validación cruzada de la misma forma que fue descrita anteriormente. Los valores de los hiperparámetros se encuentran en Anexo D.

Tabla 6: Desempeño de cada modelo optimizado para cada conjunto de datos empleando la codificación por digitalización de la propiedad fisicoquímica más informativa. El valor de cada celda es el Accuracy [%] y R2.

	ANN	CNN	RF	SVM	KNN
iAMP-2L_multi	70.09 ± 1.28	70.26 ± 0.98	63.57 ± 0.86	61.92 ± 0.77	68.45 ± 2.62
VaxinPad	83.43 ± 1.83	83.02 ± 5.75	80.72 ± 3.45	80.36 ± 3.58	79.04 ± 2.96
DBP	69.43 ± 3.48	69.18 ± 1.15	67.35 ± 1.32	66.26 ± 3.94	68.45 ± 2.61
enantio.	0.156 ± 0.261	0.110 ± 0.239	0.603 ± 0.180	0.577 ± 0.149	0.645 ± 0.195
t50	0.752 ± 0.155	0.796 ± 0.096	0.556 ± 0.079	0.827 ± 0.089	0.509 ± 0.142

Se procede a optimizar cada algoritmo para cada una de las demás formas de codificación. Los detalles se encuentran en Anexo D. Luego, se decide el mejor modelo para cada formato:

Tabla 7: Selección del mejor modelo (optimizado) para cada tipo de codificación en cada conjunto de datos.

	Onehot	Ordinal	AAC	Dipéptidos	Digitalización
iAMP-2L_multi	ANN	ANN	ANN	ANN	CNN
VaxinPad	RF	CNN	RF	ANN	ANN
DBP	ANN	CNN	ANN	ANN	CNN
enantioselectivity	SVM	RF	SVM	RF	KNN
t50	SVM	SVM	SVM	SVM	SVM

Se procedió a determinar los desempeños en los conjuntos de prueba (se estableció en 300 las *epochs*, 32 el *batch size* y 0.001 el *learning rate* para las redes neuronales). Cabe destacar que en esta instancia, no se modificó la actualización de los pesos según balances de clases para el conjunto iAMP-2L_multiclass.

Tabla 8: Rendimiento para cada forma de codificación, empleando el mejor modelo encontrado. El valor de cada celda es el Accuracy [%] y R2.

	Onehot	Ordinal	AAC	Dipéptidos	Digitalización
iAMP-2L_multi	61.45	64.87	74.34	64.08	72.07
VaxinPad	93,85	76.15	93.85	96.15	81.54
DBP	65.53	61.65	73.57	72.33	72.82
enantioselectivity	0.721	0.716	0.678	0.789	0.786
t50	0.800	0.836	0.892	0.848	0.797

De aquí es posible señalar que la digitalización posee un desempeño intermedio a lo largo de las demás formas más comunes de digitalización. En los casos de iAMP-2L_multiclass, DBP y enantioselectivity, la digitalización logró estar a la par con el rendimiento más alto de los otros formatos. Sin embargo, para t50 y VaxinPad exhibe un rendimiento equiparable con los peores resultados obtenidos. A pesar de ello, es posible apreciar cierta robustez de la digitalización, pues no obtiene desempeños inferiores a 70% para los casos de clasificación, mientras que los demás modelos exhiben al menos una caída notoria de *accuracy* a lo largo de los distintos conjuntos. Para los casos de regresión, se mantiene en un margen cercano a los R^2 obtenidos.

Para analizar de mejor manera los resultados de los conjuntos de clasificación para la codificación por digitalización, se disponen de las matrices de confusión y sus respectivos indicadores globales de los grupos VaxinPad y DBP, pues iAMP-2L se abordará en detalle posteriormente:

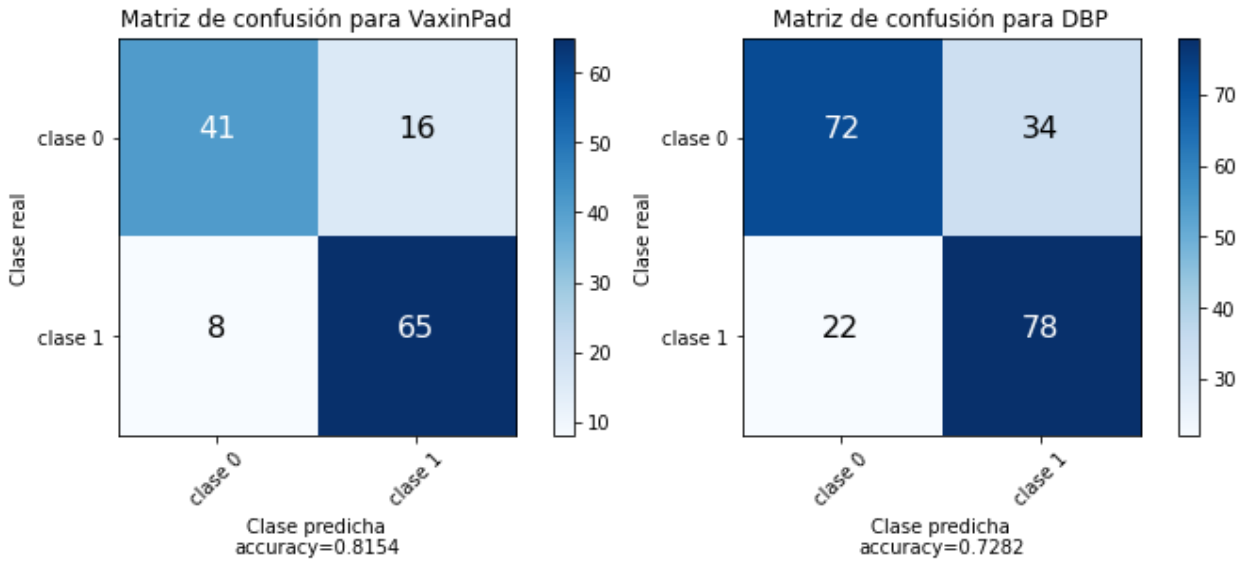


Figura 44: Matrices de confusión para el conjunto de prueba de (izquierda) VaxinPad y (derecha) DBP.

Luego, se presentan sus indicadores globales:

Tabla 9: Medidores de desempeño obtenidos para el conjunto de prueba de DBP y VaxnPad.

	Accuracy [%]	Precision [%]	Recall [%]	F-score [%]	MCC
VaxinPad	81.54	89.04	80.25	84.42	0.624
DBP	72.82	78.00	69.64	73.58	0.461

Tabla 10: Medidores de desempeño reportados en la referencia. Recall y F-score no fueron indicados.

	Accuracy [%]	Precision [%]	Recall	F-score [%]	MCC	Ref
VaxinPad	95.00	93.28	-	-	0.90	[83]
DBP	79.96	81.91	-	-	0.62	[47]

De la Figura 44 es posible observar que ambos conjuntos tienen una tendencia de identificar mejor las clases positivas. Sin embargo, el medidor relativamente bajo de *recall* (en comparación a sus demás indicadores) muestra que los modelos por codificación presentan un sesgo a clasificar como positivo a la mayor parte de las muestras. Esto se podría explicar por los gráficos de espectros multicanales y espectrogramas abordados en la sección anterior, donde en todos los casos las clases positivas exhibían menos dispersión en sus frecuencias que los grupos negativos. De las Tablas 9 y 10, se desprende que la codificación por digitalización no está a la altura de los desempeños logrados en los modelos de las referencias correspondientes. No obstante, es importante tomar en consideración que estos estudios construyeron modelos específicos para cada tarea, por lo que el esfuerzo en establecer mejores hiperparámetros fue mayor que en el presente trabajo. Además, ambos emplearon información más diversa para la extracción de características: en el caso de VaxinPad, se realizaron búsquedas de *motifs* relevantes y se armó un modelo híbrido en conjunto con la composición aminoacídica; en cuanto a DBP, el labor fue focalizado en la identificación de patrones de secuencias mediante el modelo n-gram, es decir, que se determinan interacciones entre distintos números de aminoácidos que estén separados por una distancia predeterminada. Para poder verificar la seguridad de la predicción realizada por los modelos presentes, es importante estudiar sus curvas ROC:

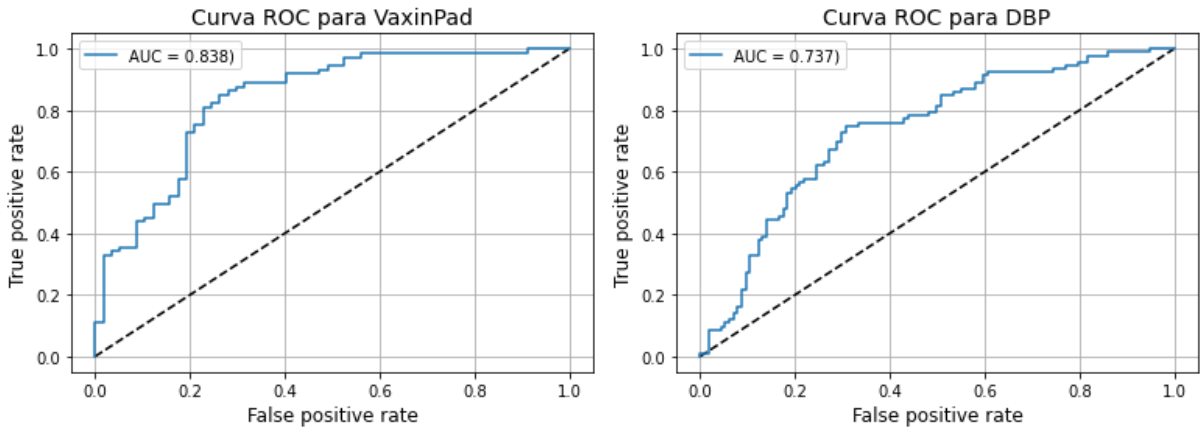


Figura 45 Curva ROC para los conjuntos de (izquierda) VaxinPad y (derecha) DBP.

Las curvas ROC se construyen mediante la relación existente entre la tasa de falsos positivos y la tasa de verdaderos positivos, a partir de la modificación del umbral que permite distinguir entre dos clases (que por lo general, se establece por defecto en 0.5 en el proceso de aprendizaje) [93]. Dado a que la curva es monótonamente creciente, si ésta presenta una pendiente muy pronunciada al inicio, implica que se logra una gran seguridad en la separación de las clases [93]. No obstante, este no es el caso para los gráficos que se muestran en la Figura 45. Se aprecia una pendiente relativamente baja, junto con un área distante de 1. Esto implica que la existe incertidumbre al momento de identificar los dos grupos, lo que corrobora los resultados previamente mostrados. Esto puede deberse a la distribución de los espectros según clases: si bien los espectros de las clases positivas suelen tener menor dispersión, su rango de magnitudes siguen formando parte del rango que admiten las clases negativas.

A continuación, se presenta la matriz de confusión para el conjunto iAMP-2L_multiclass.

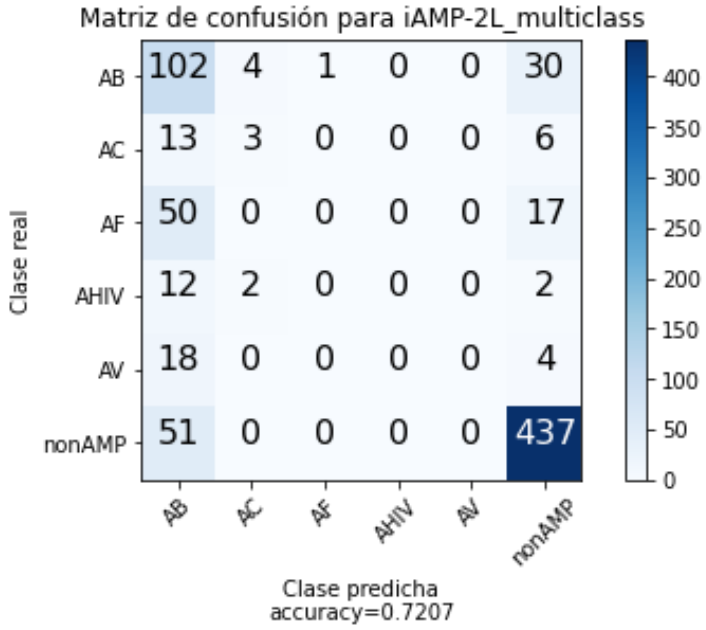


Figura 46: Matriz de confusión para el conjunto iAMP-2L_multiclass sin modificación de los pesos según distribución de clases.

Tabla 11: Medidores de desempeño para conjunto iAMP-2L_multiclass sin modificación de los pesos según distribución de clases.

	Precision [%]	Recall [%]	F-score [%]
AB	74.45	41.46	53.26
AC	13.63	33.33	19.34
AF	0.00	0.00	0.00
AHIV	0.00	0.00	0.00
AV	0.00	0.00	0.00
nonAMP	89.55	88.10	88.19

De la Figura 46 y Tabla 11 se puede observar un comportamiento completamente errático para las clases AC, AF, AHIV y AV, mientras. Más aun, el modelo tiende a "transformar" este problema de clasificación multicategórica a uno de clasificación binaria: entre AB y nonAMP. Esto se debería principalmente al desbalance de clases que existe, pues el grupo nonAMP presenta aproximadamente 20 veces más muestras que los demás grupos, a excepción de AB, al cual supera por 3 veces su tamaño. Por lo tanto, resulta interesante modificar el modelo para que tome en cuenta esta distribución desigual.

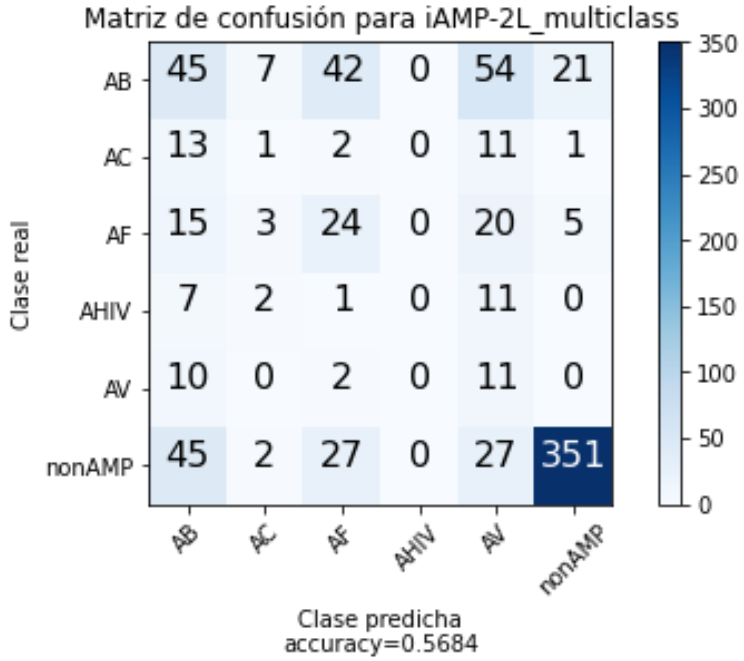


Figura 47: Matriz de confusión para el conjunto iAMP-2L_multiclass con modificación de los pesos según distribución de clases.

La *accuracy* disminuye considerablemente (aproximadamente un 20%), pues esta vez logra identificar menos ejemplares correctamente. Sin embargo, es importante destacar que el modelo logra mayor sensibilidad a los grupos subrepresentados. La poca capacidad de clasificar apropiadamente un péptido puede deberse a lo que se expuso en la sección anterior: los espectros suelen parecerse entre las clases antimicrobianas, pero logra una fuerte distinción con respecto a los péptidos no-antimicrobianos.

Tabla 12: Medidores de desempeño para conjunto iAMP-2L_multiclass sin modificación de los pesos según distribución de clases.

	Precision [%]	Recall [%]	F-score [%]
AB	26.62	33.33	29.60
AC	3.70	6.67	4.76
AF	35.82	24.49	29.09
AHIV	0.00	0.00	0.00
AV	47.83	8.21	14.01
nonAMP	77.65	92.86	84.58

Asimismo, se dispone de los resultados obtenidos a lo largo de todos los conjuntos de datos fuera de los casos de estudio. La Tabla 13 indica las propiedades fisicoquímicas más informativas y la Tabla 14 indica los desempeños logrados.

Tabla 13: Propiedades fisicoquímicas más informativas para los conjuntos fuera de los casos de estudio.

	Propiedad	Nombre
ACP-DL	PRAM900103	Frecuencia relativa en hojas beta
AntiTb_primary	GRAR740103	Volumen
AntitTb_secondary	GRAR740103	Volumen
iACP	JOND750101	Hidrofilicidad
iAMP-2L_binary	RADA880106	Área de superficie accesible
localization	HOPT810101	Hidrofobicidad
Pop_ara	N/A	N/A
Pop_chlamy	N/A	N/A
Pop_yeast	N/A	N/A
QSP	GRAR740103	Volumen
RT_hela	GRAR740103	Volumen
RT_yeast	JOND750101	Hidrofilicidad
Solub	JOND750101	Hidrofilicidad

Tabla 14: Medidores de desempeño logrados para los conjuntos que no forman parte de los casos de prueba. Se destacaron los valores mejor o similares que en la referencia (en un margen de 5%). Para cada uno se utilizó CNN.

	Desempeño logrado (accuracy [%] o R^2)	Desempeño reportado en la referencia	Ref.
ACP-DL	67.55	83.45	[24]
AntiTb_primary	72.46	73.20	[23]
AntitTb_secondary	78.99	75.62	[23]
iACP	83.06	93.72	[27]
iAMP-2L_binary	97.00	92.23	[33]
localization	0.447	-	[85]
Pop_ara	N/A	0.62	[80]
Pop_chlamy	N/A	0.62	[80]
Pop_yeast	N/A	0.62	[80]
QSP	92.36	92.50	[29]
RT_hela	0.681	0.971	[79]
RT_yeast	0.643	0.987	[79]
Solub	0.376	0.42	[84]

Es posible evidenciar que, en general, el método de codificación por digitalización logra resultados favorables o incluso superiores para los conjuntos de clasificación, mientras que es notoriamente peor en los casos de regresión. Se debe recordar que en este trabajo no se realizaron modelos híbridos (i.e. que tomen en consideración múltiples formas de codificación o que se usen algoritmos de aprendizaje supervisado combinados con otros), mientras que la mayoría de las referencias se basan en la extracción de características diversas en la construcción de su metodología. Asimismo, los casos presentados llevaron a cabo modelos específicos para cada conjunto de datos, mientras que en el presente labor se mantuvo una generalización para proponer una metodología de diseño de modelos predictivos que empleen la digitalización como formato de codificación de secuencias. Los conjuntos correspondientes a las divisiones de Pop no pueden ser reportados porque nunca se logró un coeficiente de determinación significativamente distinto de cero, lo cual indica completa aleatoriedad en la predicción, o un sesgo importante. En el caso de Localization, no se reporta un desempeño de referencia porque se determinaron otros indicadores que no se vieron en este trabajo. Sin embargo, 0,447 es un bajo valor de R^2 debido a que logra explicar menos de la mitad de la varianza que exhiben las respuestas de los péptidos.

En general, el desempeño de la digitalización apunta a más utilizable en los problemas de clasificación binaria, pues los espectros de dos péptidos suficientemente distintivos entre sí logran denotar comportamientos diferentes, ya sea la identificación de *peaks* o una dispersión particular de las magnitudes de sus frecuencias. En los casos de regresión, se obtienen mejores desempeños para los casos de enantioselectividad y t50 que para el resto de los casos principalmente debido posiblemente a dos razones: primero, que se disponen de modelos más optimizados para los primeros porque corresponden a casos de estudio, segundo, porque los espectros de Fourier para secuencias marginalmente distintas (i. e. mutaciones puntuales o recombinación local) pueden exhibir aumentos o disminuciones de *peaks* lo suficiente como para explicar un cambio en un valor de respuesta. En cambio, para los demás casos de regresión, los conjuntos son altamente heterogéneos, lo cual genera una altísima dispersión en las magnitudes que se deben procesar, lo cual generaría que la sensibilidad de la codificación por digitalización esté relacionada con distribuciones de cada frecuencia a lo largo de todo el conjunto de dato. Para corroborar dicha hipótesis, se propone realizar un estudio estadístico sobre cambios controlados en los espectros de Fourier de tal manera que presenten distintas distribuciones probabilísticas a lo largo de un mismo conjunto de características globalmente similares. Dicho estudio no necesariamente debe realizarse a partir de proteínas, sino que es posible generar datos artificiales para lograr una conclusión al respecto.

Un aspecto importante de abordar es que utilizar espectros de Fourier de una única propiedad fisicoquímica implica introducir un error de sesgo en el modelo debido a que se limita la información máxima a la que puede acceder. Es bien sabido que los péptidos/proteínas funcionan en base a interacciones fisicoquímicas complejas, ya sea adoptando estructuras distintas para mecanismos enzimático, penetrar membranas celulares mediante un grupo de aminoácidos terminales, lograr aglomerados de proteínas mediante la exposición de aminoácidos hidrofóbicos, entre otros. Por lo tanto, es necesario mantener información íntegra al momento de realizar una codificación. El mal desempeño de la digitalización en algunos caso podría deberse a, por ejemplo, que se esté solamente evaluando el perfil hidrofóbico de un conjunto de péptidos antimicrobianos, pero que no se esté considerando los aminoácidos hidrofílicos que requieren para

introducir mecanismos de acción específicos. O en el caso de las *DBP* (*DNA binding protein*), se tomó en cuenta la estructura de hélice alfa para el modelo, pero es altamente probable que el perfil de contacto iónico cumpla un rol importante para identificar este conjunto. Debido a esto, se propone encontrar un método que logra integrar de manera compacta (para evitar el efecto negativo de la *sparsity* de onehot, por ejemplo) las distintas propiedades fisicoquímicas más informativas de AAindex, que serían las ocho mostradas. Una forma de llevar a cabo esto podría ser mediante el uso de imágenes de Fourier en lugar de un espectro unidimensional, para ello, se puede realizar transformaciones en una matriz de N propiedades fisicoquímicas. Otra forma de abordar este obstáculo podría ser el uso de una CNN que reciba espectros en diversos canales, ya que en el presente trabajo no se trabajó con más de un único canal en la capa de entrada de la red convolucional.

A pesar de las múltiples dificultades que la DFT puede presentar, al ser más robusto que las demás formas de codificación, figura como una buena alternativa para realizar modelos predictivos preliminares y pivotear desde ese punto a modelos mejores. Ya se mencionó que la mayoría de los modelos realizados en las referencias abordan sus problemas mediante el uso de codificaciones híbridas. Por lo tanto, se propone que para un trabajo futuro se lleve a cabo un estudio que corrobore la efectividad de integrar otras codificaciones a la digitalización. Asimismo, siempre es posible iterar sobre las arquitecturas de las redes neuronales y probar otras combinaciones de hiperparámetros para optimizar la información disponible en los espectros de Fourier.

8. Conclusiones

En el presente trabajo de Memoria tuvo como objetivo general plantear una metodología para el diseño y de modelos predictivos sobre un conjunto de datos de péptidos/proteínas empleando la codificación por digitalización de propiedades fisicoquímicas como su descriptor único. Asimismo, se requirió de una validación de dicha metodología para el formato de codificación propuesto. Para ello, se trabajó con 18 conjuntos independientes conformados por secuencias distintas, englobando problemas de clasificación binaria, regresión y clasificación multicategórica. Se centró el estudio en cinco conjuntos para establecer un caso de estudio significativo. La validación se basó en la comparación del presente método con cuatro formatos de codificación ampliamente usados: *onehot*, *ordinal*, composición aminoacídica, y composición de dipéptidos. También se contrastó la calidad de información extraíble de los espectros de Fourier y del alineamiento de las secuencias.

La metodología propuesta consta de cinco puntos relevantes: caracterización de los conjuntos de datos, análisis de los espectros de Fourier, identificación de la propiedad fisicoquímica más informativa, construcción de un modelo con parámetros optimizados y validación de dicho modelo. A lo largo de la presente Memoria, se aplicó dicha metodología a los conjuntos: DBP (identificación de *DNA binding proteins*), iAMP-2L_multiclass (clasificación de péptidos en distintos grupos antimicrobianos), VaxinPad (identificación de péptidos inmunomoduladores), enantioselectivity (regresión de enantioselectividad de proteínas producto de mutaciones puntuales de una única pariente), y t50 (determinación de la temperatura t50 de recombinaciones quiméricas de tres proteínas parientes). Los aspectos destacables fueron los siguientes:

- La caracterización de los conjuntos de datos permitió tener una estimación del desbalance de clases y de la distribución desigual de los largos de secuencia. A raíz de esto, se tomó en consideración ponderaciones distintas para las diferentes clases de iAMP-2L_multiclass al momento de realizar el entrenamiento del modelo. Se removió los outliers de largo de secuencia para todos los conjuntos.
- El alineamiento no logra diferenciar entre dos grupos globalmente distintos al estar formados por un conjunto altamente heterogéneo de péptidos/proteínas. Sin embargo, se extrae potencial información relevante a partir de la búsqueda de *motifs* en conjuntos de secuencias más pequeñas.
- Los espectros de Fourier logran principalmente aportar con información referente a la dispersión de las magnitudes de las frecuencias a lo largo de distintas secuencias, y a la identificación de *peaks* de relevancia que permiten diferenciar dos grupos diferentes de péptidos/proteínas. Desde la apreciación humana, los espectros de Fourier fallan al intentar exhibir diferencias de respuestas para los conjuntos enantioselectivity y t50, los cuales corresponden a conjuntos de proteínas con alta similitud.
- Se probó la codificación por digitalización y las cuatro formas comunes en cinco diferentes algoritmos/arquitecturas: *K-Neareast Neighbor*, *Support Vector Machine, Random Forest*, *Artificial Neural Network*, y *Convolutional Neural Network*. Se logró apreciar que la digitalización presenta robustez para cada uno de los algoritmos y para cada conjunto de datos, logrando desempeños similares o superiores al obtenido por el mejor entre las cuatro formas comunes.

- Los modelos de digitalización tienden a clasificar péptidos como positivos (indican bajo *recall*), potencialmente debido a que los espectros de Fourier de los grupos positivos se encuentran dentro del margen de dispersión de las magnitudes de frecuencia de las clases negativas
- Para el caso de iAMP-2L_multiclass, el modelo de digitalización es insensible a las clases subrepresentadas si no se asignan pesos a dichas clases según su distribución en el conjunto de datos. Más aun, al introducir este cambio, la capacidad predictiva del modelo empeora, lo cual puede deberse a la dificultad que presentan los espectros de Fourier para diferenciar dos clases similares.
- Para los modelos de regresión de conjuntos heterogéneos, la digitalización exhibe rendimientos bajos, potencialmente debido a la gran dispersión de las magnitudes de las frecuencias de Fourier.
- Al comparar los modelos obtenidos con la literatura, se aprecia que se logran resultados similares o mejores para los casos de clasificación, pero mantiene grandes dificultades para abordar conjuntos de regresión.

A raíz de esto, los objetivos específicos se logran parcialmente, pues se lleva a cabo exitosamente la creación y validación de los modelos para conjuntos de clasificación, pero se obtienen desempeños bajos para casos de regresión. Sin embargo, el objetivo general se cumple, ya que se dispone de una metodología para implementar la digitalización por propiedades fisicoquímicas, y de esta forma se tiene un punto de pivoteo robusto para lograr modelos mejores.

Como propuesta de trabajo futura, se plantea la incorporación de múltiples propiedades fisicoquímicas para incrementar el poder predictivo de los modelos previamente construidos. Asimismo, es posible integrar la codificación por digitalización con otras formas para así construir un modelo híbrido que tome en consideración información más completa de las secuencias.

9. Bibliografía

[1] K. Mau Goh, G. Poh Hong, N. H. C. @ Pearly Ng, C. Kian Piaw, and R. N. Z. Raja Abdul Rahman, "Trends and Tips in Protein Engineering, A Review," *J. Teknol.*, vol. 59, no. 1, Mar. 2013.

[2] S. Lutz and S. M. Iamurri, "Protein Engineering: Past, Present, and Future," in *Methods in Molecular Biology*, 2018, pp. 1–12.

[3] J. Lei *et al.*, "The antimicrobial peptides and their potential clinical applications," *American Journal of Translational Research*, vol. 11, no. 7. E-Century Publishing Corporation, pp. 3919–3931, 2019.

[4] R. Seyfi *et al.*, "Antimicrobial Peptides (AMPs): Roles, Functions and Mechanism of Action," *International Journal of Peptide Research and Therapeutics*, vol. 26, no. 3. Springer, pp. 1451–1463, 01-Sep-2020.

[5] M. Y. Galperin, D. R. Walker, and E. V Koonin, "Analogous enzymes: independent inventions in enzyme evolution.," *Genome Res.*, vol. 8, no. 8, pp. 779–90, Aug. 1998.

[6] A. E. Todd, C. A. Orengo, and J. M. Thornton, "Sequence and Structural Differences between Enzyme and Nonenzyme Homologs," *Structure*, vol. 10, no. 10, pp. 1435–1451, Oct. 2002.

[7] "protein engineering | Encyclopedia.com." [Online]. Available: https://www.encyclopedia.com/science/dictionaries-thesauruses-pictures-and-press-releases/protein-engineering. [Accessed: 01-Jul-2019].

[8] Z. Chen and H. Zhao, "Protein Design."

[9] M. S. Packer and D. R. Liu, "Methods for the directed evolution of proteins," *Nature Reviews Genetics*, vol. 16, no. 7. Nature Publishing Group, pp. 379–394, 19-Jul-2015.

[10] K. Bozek, T. Lengauer, S. Sierra, R. Kaiser, and F. S. Domingues, "Analysis of Physicochemical and Structural Properties Determining HIV-1 Coreceptor Usage," *PLoS Comput. Biol.*, vol. 9, no. 3, 2013.

[11] J. Rodriguez, S. Rath, J. Francis-Landau, Y. Demirci, B. B. Üstündağ, and M. Sarikaya, "A Generalized Similarity Metric for Predicting Peptide Binding Affinity," *A Gen. Similarity Metr. Predict. Pept. Bind. Affin.*, p. 654913, Aug. 2019.

[12] A. Jain and D. Kihara, "NNTox: Gene Ontology-Based Protein Toxicity Prediction Using Neural Network," *Sci. Rep.*, vol. 9, no. 1, p. 17923, Nov. 2019.

[13] C. Wu, R. Gao, Y. Zhang, and Y. De Marinis, "PTPD: Predicting therapeutic peptides by deep learning and word2vec," *BMC Bioinformatics*, vol. 20, no. 1, p. 456, Sep. 2019.

[14] L. Wang, H. F. Wang, S. R. Liu, X. Yan, and K. J. Song, "Predicting Protein-Protein Interactions from Matrix-Based Protein Sequence Using Convolution Neural Network and Feature-Selective Rotation Forest," *Sci. Rep.*, vol. 9, no. 1, pp. 1–12, Dec. 2019.

[15] H. Fukuda and K. Tomii, "DeepECA: An end-to-end learning framework for protein contact prediction from a multiple sequence alignment," *BMC Bioinformatics*, vol. 21, no. 1, p. 10, Jan. 2020.

[16] C. Recio, F. Maione, A. J. Iqbal, N. Mascolo, and V. De Feo, "The Potential Therapeutic Application of Peptides and Peptidomimetics in Cardiovascular Disease," *Front. Pharmacol.*, vol. 7, p. 526, Jan. 2017.

[17] B. Yavari, R. Mahjub, M. Saidijam, M. Raigani, and M. Soleimani, “The Potential Use of Peptides in Cancer Treatment,” *Curr. Protein Pept. Sci.*, vol. 19, no. 8, pp. 759–770, Jun. 2018.
[18] A. Pfalzgraff, K. Brandenburg, and G. Weindl, “Antimicrobial Peptides and Their Therapeutic Potential for Bacterial Skin Infections and Wounds,” *Front. Pharmacol.*, vol. 9, p. 281, Mar. 2018.
[19] “Problems and solutions in protein engineering-towards rational design,” 1994.
[20] T. Schwede, “Protein modeling: what happened to the "protein structure gap"?,” *Structure*, vol. 21, no. 9, pp. 1531–40, Sep. 2013.
[21] K. K. Yang, Z. Wu, and F. H. Arnold, “Machine learning-guided directed evolution for protein engineering,” Nov. 2018.
[22] S. Spänig and D. Heider, “Encodings and models for antimicrobial peptide classification for multi-resistant pathogens,” *BioData Min.*, vol. 12, no. 1, pp. 1–29, 2019.
[23] S. S. Usmani, S. Bhalla, and G. P. S. Raghava, “Prediction of antitubercular peptides from sequence information using ensemble classifier and hybrid features,” *Front. Pharmacol.*, vol. 9, no. AUG, pp. 1–11, 2018.
[24] H. C. Yi *et al.*, “ACP-DL: A Deep Learning Long Short-Term Memory Model to Predict Anticancer Peptides Using High-Efficiency Feature Representation,” *Mol. Ther. - Nucleic Acids*, vol. 17, no. September, pp. 1–9, 2019.
[25] P. K. Meher, T. K. Sahu, V. Saini, and A. R. Rao, “Predicting antimicrobial peptides with improved accuracy by incorporating the compositional, physico-chemical and structural features into Chou’s general PseAAC,” *Sci. Rep.*, vol. 7, no. February, 2017.
[26] Y. H. Li *et al.*, “SVM-prot 2016: A web-server for machine learning prediction of protein functional families from sequence irrespective of similarity,” *PLoS One*, vol. 11, no. 8, pp. 1–14, 2016.
[27] W. Chen, H. Ding, P. Feng, H. Lin, and K. C. Chou, “iACP: A sequence-based tool for identifying anticancer peptides,” *Oncotarget*, vol. 7, no. 13, pp. 16895–16909, 2016.
[28] D. Sarkar, T. Jana, and S. Saha, “LMDIPred: A web-server for prediction of linear peptide sequences binding to SH3, WW and PDZ domains,” *PLoS One*, vol. 13, no. 7, pp. 1–14, 2018.
[29] A. Rajput, A. K. Gupta, and M. Kumar, “Prediction and analysis of quorum sensing peptides based on sequence features,” *PLoS One*, vol. 10, no. 3, pp. 1–16, 2015.
[30] B. Yu, C. Chen, Z. Yu, A. Ma, B. Liu, and Q. Ma, “Prediction of protein-protein interactions based on elastic net and deep forest,” *bioRxiv*, p. 2020.04.23.058644, 2020.
[31] C. Zhou, H. Yu, Y. Ding, F. Guo, and X. J. Gong, “Multi-scale encoding of amino acid sequences for predicting protein interactions using gradient boosting decision tree,” *PLoS One*, vol. 12, no. 8, pp. 1–18, 2017.
[32] K. C. Chou, “Prediction of protein cellular attributes using pseudo-amino acid composition,” *Proteins Struct. Funct. Genet.*, vol. 43, no. 3, pp. 246–255, 2001.
[33] X. Xiao, P. Wang, W. Z. Lin, J. H. Jia, and K. C. Chou, “IAMP-2L: A two-level multi-label classifier for identifying antimicrobial peptides and their functional types,” *Anal. Biochem.*, vol. 436, no. 2, pp. 168–177, 2013.
[34] Z. Khan, “PREDICTION OF ACIDIC AND ALKALINE ENZYME USING PSEUDO AMINO ACID PREDICTION OF ACIDIC AND ALKALINE ENZYME USING PSEUDO AMINO ACID COMPOSITION IN COMBINATION WITH MACHINE

LEARNING APROACHES BY Synopsis Submitted to the Abdul Wali Khan University Mardan in ," no. August, 2016.
[35] D. Veltri, U. Kamath, and A. Shehu, "Improving Recognition of Antimicrobial Peptides and Target Selectivity through Machine Learning and Genetic Programming," *IEEE/ACM Trans. Comput. Biol. Bioinforma.*, vol. 14, no. 2, pp. 300–313, 2017.
[36] P. Bhadra, J. Yan, J. Li, S. Fong, and S. W. I. Siu, "AmPEP: Sequence-based prediction of antimicrobial peptides using distribution patterns of amino acid properties and random forest," *Sci. Rep.*, vol. 8, no. 1, pp. 1–10, 2018.
[37] T. Sun, B. Zhou, L. Lai, and J. Pei, "Sequence-based prediction of protein protein interaction using a deep-learning algorithm," *BMC Bioinformatics*, vol. 18, no. 1, pp. 1–8, 2017.
[38] S. Giguère *et al.*, "Machine Learning Assisted Design of Highly Active Peptides for Drug Discovery," *PLoS Comput. Biol.*, vol. 11, no. 4, pp. 1–21, 2015.
[39] K. Boone, K. Camarda, P. Spencer, and C. Tamerler, "Antimicrobial peptide similarity and classification through rough set theory using physicochemical boundaries," *BMC Bioinformatics*, vol. 19, no. 1, pp. 1–10, 2018.
[40] F. Cadet *et al.*, "A machine learning approach for reliable prediction of amino acid interactions and its application in the directed evolution of enantioselective enzymes," *Sci. Rep.*, vol. 8, no. 1, pp. 1–15, 2018.
[41] F. Cadet *et al.*, "Application of fourier transform and proteochemometrics principles to protein engineering," *BMC Bioinformatics*, vol. 19, no. 1, p. 382, Dec. 2018.
[42] K. H. Chen, T. F. Wang, and Y. J. Hu, "Protein-protein interaction prediction using a hybrid feature representation and a stacked generalization scheme," *BMC Bioinformatics*, vol. 20, no. 1, pp. 1–17, 2019.
[43] P. Charoenkwan, N. Schaduangrat, C. Nantasenamat, T. Piacham, and W. Shoombuatong, "IQSP: A sequence-based tool for the prediction and analysis of quorum sensing peptides via chou's 5-steps rule and informative physicochemical properties," *Int. J. Mol. Sci.*, vol. 21, no. 1, 2020.
[44] "AAindex: Amino acid index database." [Online]. Available: https://www.genome.jp/aaindex/. [Accessed: 25-May-2020].
[45] V. V. Kleandrova, J. M. Ruso, A. Speck-Planche, and M. N. Dias Soeiro Cordeiro, "Enabling the Discovery and Virtual Screening of Potent and Safe Antimicrobial Peptides. Simultaneous Prediction of Antibacterial Activity and Cytotoxicity," *ACS Comb. Sci.*, vol. 18, no. 8, pp. 490–498, 2016.
[46] X. Zhou, R. Yin, C. K. Kwoh, and J. Zheng, "A context-free encoding scheme of protein sequences for predicting antigenicity of diverse influenza A viruses," *BMC Genomics*, vol. 19, no. Suppl 10, 2018.
[47] F. Ali *et al.*, "DBPPred-PDSD: Machine learning approach for prediction of DNA-binding proteins using Discrete Wavelet Transform and optimized integrated features space," *Chemom. Intell. Lab. Syst.*, vol. 182, pp. 21–30, Nov. 2018.
[48] J. Y. An, Z. H. You, F. R. Meng, S. J. Xu, and Y. Wang, "RVMAB: Using the relevance vector machine model combined with average blocks to predict the interactions of proteins from protein sequences," *Int. J. Mol. Sci.*, vol. 17, no. 5, 2016.
[49] L. Wang, H. F. Wang, S. R. Liu, X. Yan, and K. J. Song, "Predicting Protein-Protein Interactions from Matrix-Based Protein Sequence Using Convolution Neural Network and Feature-Selective Rotation Forest," *Sci. Rep.*, vol. 9, no. 1, pp. 1–12,

2019.
[50] Z. W. Li *et al.*, "Accurate prediction of protein-protein interactions by integrating potential evolutionary information embedded in PSSM profile and discriminative vector machine classifier," *Oncotarget*, vol. 8, no. 14, pp. 23638–23649, 2017.
[51] G. H. Liu, H. Bin Shen, and D. J. Yu, "Prediction of Protein–Protein Interaction Sites with Machine-Learning-Based Data-Cleaning and Post-Filtering Procedures," *J. Membr. Biol.*, vol. 249, no. 1–2, pp. 141–153, 2016.
[52] Y. Wang, Z. You, X. Li, X. Chen, T. Jiang, and J. Zhang, "PCVMZM: Using the probabilistic classification vector machines model combined with a Zernike moments descriptor to predict protein-protein interactions from protein sequences," *Int. J. Mol. Sci.*, vol. 18, no. 5, pp. 1–13, 2017.
[53] Z.-H. Chen *et al.*, "Prediction of Self-Interacting Proteins from Protein Sequence Information Based on Random Projection Model and Fast Fourier Transform," *Int. J. Mol. Sci.*, vol. 20, no. 4, p. 930, Feb. 2019.
[54] C. Yin and S. S. T. Yau, "A coevolution analysis for identifying proteinprotein interactions by Fourier transform," *PLoS One*, vol. 12, no. 4, pp. 1–19, 2017.
[55] L. Al Shalabi, Z. Shaaban, and B. Kasasbeh, "Data Mining: A Preprocessing Engine," *J. Comput. Sci.*, vol. 2, no. 9, pp. 735–739, 2006.
[56] A. Estabrooks, T. Jo, and N. Japkowicz, "A multiple resampling method for learning from imbalanced data sets," *Comput. Intell.*, vol. 20, no. 1, pp. 18–36, Feb. 2004.
[57] A. Saleem, K. H. Asif, A. Ali, S. M. Awan, and M. A. Alghamdi, "Pre-processing methods of data mining," in *Proceedings - 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, UCC 2014*, 2014, pp. 451–456.
[58] Y. C. Chen, "A tutorial on kernel density estimation and recent advances," *Biostat. Epidemiol.*, vol. 1, no. 1, pp. 161–187, Jan. 2017.
[59] L. J. Latecki, A. Lazarevic, and D. Pokrajac, "Outlier detection with kernel density functions," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2007, vol. 4571 LNAI, pp. 61–75.
[60] D. W. Scott, "Scott's rule," *Wiley Interdiscip. Rev. Comput. Stat.*, vol. 2, no. 4, pp. 497–502, Jul. 2010.
[61] S. Sinai, E. Kelsic, G. M. Church, and M. A. Nowak, "Variational auto-encoding of protein sequences."
[62] F. Fabris, J. P. de Magalhães, and A. A. Freitas, "A review of supervised machine learning applied to ageing research," *Biogerontology*, vol. 18, no. 2, pp. 171–188, Apr. 2017.
[63] D. Medina-Ortiz *et al.*, "Combination of digital signal processing and assembled predictive models facilitates the rational design of proteins," Oct. 2020.
[64] D. Sarkar and S. Saha, "Machine-learning techniques for the prediction of protein–protein interactions," *J. Biosci.*, vol. 44, no. 4, 2019.
[65] "Understanding Support Vector Machine Regression - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/help/stats/understanding-support-vector-machine-regression.html. [Accessed: 26-May-2020].
[66] A. Sikandar *et al.*, "Decision Tree Based Approaches for Detecting Protein Complex in Protein Protein Interaction Network (PPI) via Link and Sequence Analysis," *IEEE Access*, vol. 6, pp. 22108–22120, 2018.
[67] B. S. Satpute and R. Yadav, "Decision tree classifier for classification of proteins using the protein data bank," in *Studies in Computational Intelligence*, vol. 771,

Springer Verlag, 2019, pp. 71–78.
[68] H. Li, X. J. Gong, H. Yu, and C. Zhou, “Deep neural network based predictions of protein interactions using primary sequences,” *Molecules*, vol. 23, no. 8, pp. 1–16, 2018.
[69] S. K. Sarvepalli, S. Sarat, and K. Sarvepalli, “Deep Learning in Neural Networks: The science behind an Artificial Brain,” 2015.
[70] S. Vieira, W. H. L. Pinaya, and A. Mechelli, “Using deep learning to investigate the neuroimaging correlates of psychiatric and neurological disorders: Methods and applications,” *Neurosci. Biobehav. Rev.*, vol. 74, no. January, pp. 58–75, 2017.
[71] A. M. Fred Agarap, “Deep Learning using Rectified Linear Units (ReLU).”
[72] K. Eckle and J. Schmidt-Hieber, “A comparison of deep networks with ReLU activation function and linear spline-type methods,” *Neural Networks*, vol. 110, pp. 232–242, Feb. 2019.
[73] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, “Dying ReLU and Initialization: Theory and Numerical Examples,” vol. 107, pp. 1–32, Mar. 2019.
[74] J. Yan *et al.*, “Deep-AmPEP30: Improve Short Antimicrobial Peptides Prediction with Deep Learning,” *Mol. Ther. - Nucleic Acids*, vol. 20, pp. 882–894, Jun. 2020.
[75] A. Bhande, “What is underfitting and overfitting in machine learning and how to deal with it.,” 2018. [Online]. Available: https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76. [Accessed: 01-Jul-2019].
[76] F. Cadet *et al.*, “A machine learning approach for reliable prediction of amino acid interactions and its application in the directed evolution of enantioselective enzymes.,” *Sci. Rep.*, vol. 8, no. 1, p. 16757, Nov. 2018.
[77] “A Detailed Introduction To Cross-Validation in Machine Learning -.” [Online]. Available: http://thatdatatho.com/2018/10/11/detailed-introduction-cross-validation-machine-learning/. [Accessed: 01-Jul-2019].
[78] “What is a ROC Curve and How to Interpret It | Displayr.” [Online]. Available: https://www.displayr.com/what-is-a-roc-curve-how-to-interpret-it/. [Accessed: 25-May-2020].
[79] C. Ma, Y. Ren, J. Yang, Z. Ren, H. Yang, and S. Liu, “Improved Peptide Retention Time Prediction in Liquid Chromatography through Deep Learning,” *Anal. Chem.*, vol. 90, no. 18, pp. 10881–10888, 2018.
[80] D. Zimmer, K. Schneider, F. Sommer, M. Schroda, and T. Mühlhaus, “Artificial intelligence understands peptide observability and assists with absolute protein quantification,” *Front. Plant Sci.*, vol. 871, no. November, pp. 1–12, 2018.
[81] M. Nei and J. Zhang, “Evolutionary Distance: Estimation.”
[82] “Distance Estimation --MEGA manual.” [Online]. Available: https://www.megasoftware.net/mega1_manual/Distance.html. [Accessed: 07-Sep-2020].
[83] G. Nagpal, K. Chaudhary, P. Agrawal, and G. P. S. Raghava, “Computer-aided prediction of antigen presenting cell modulators for designing peptide-based vaccine adjuvants,” *J. Transl. Med.*, vol. 16, no. 1, pp. 1–15, 2018.
[84] X. Han, X. Wang, K. Zhou, and A. Valencia, “Develop machine learning-based regression predictive models for engineering protein solubility,” *Bioinformatics*, vol. 35, no. 22, pp. 4640–4646, 2019.
[85] K. K. Yang, Z. Wu, C. N. Bedbrook, and F. H. Arnold, “Learned protein embeddings for machine learning,” *Bioinformatics*, vol. 34, no. 15, pp. 2642–2648, 2018.

[86] L. Zhang, R. Yang, and C. Zhang, “Using a Classifier Fusion Strategy to Identify Anti-angiogenic Peptides,” *Sci. Rep.*, vol. 8, no. 1, pp. 1–12, Dec. 2018.
[87] Q. Yang, X. Cai, M. Huang, and S. Wang, “A specific peptide with immunomodulatory activity from Pseudostellaria heterophylla and the action mechanism,” *J. Funct. Foods*, vol. 68, p. 103887, May 2020.
[88] E. Elayaraja, K. Thangavel, B. Ramya, and M. Chitralegha, “Extraction of Motif Patterns from Protein Sequence Using Rough- K-Means Algorithm,” *Procedia Eng.*, vol. 30, pp. 814–820, 2012.
[89] Y. Ding, Y. Cai, G. Zhang, and W. Xu, “The influence of dipeptide composition on protein thermostability,” *FEBS Lett.*, vol. 569, no. 1–3, pp. 284–288, Jul. 2004.
[90] M. Andreatta, B. Alvarez, and M. Nielsen, “GibbsCluster: Unsupervised clustering and alignment of peptide sequences,” *Nucleic Acids Res.*, vol. 45, no. W1, pp. W458–W463, Jul. 2017.
[91] “WebLogo - Create Sequence Logos.” [Online]. Available: https://weblogo.berkeley.edu/logo.cgi. [Accessed: 08-Sep-2020].
[92] O. N. Silva *et al.*, “An anti-infective synthetic peptide with dual antimicrobial and immunomodulatory activities,” *Sci. Rep.*, vol. 6, no. 1, pp. 1–11, Nov. 2016.
[93] D. K. McClish, “Analyzing a Portion of the ROC Curve,” *Med. Decis. Mak.*, vol. 9, no. 3, pp. 190–195, Aug. 1989.
[94] “Antimicrobial Peptide Database - DBAASP.” [Online]. Available: https://dbaasp.org/. [Accessed: 28-May-2020].
[95] “The Antimicrobial Peptide Database APD3.” [Online]. Available: http://aps.unmc.edu/AP/main.php. [Accessed: 28-May-2020].
[96] “UniProt.” [Online]. Available: https://www.uniprot.org/. [Accessed: 28-May-2020].
[97] “UniProtKB.” [Online]. Available: https://www.uniprot.org/help/uniprotkb. [Accessed: 28-May-2020].
[98] “Quorumpeps Home | Quorumpeps.” [Online]. Available: http://quorumpeps.ugent.be/. [Accessed: 28-May-2020].
[99] “RCSB PDB.” [Online]. Available: https://www.rcsb.org/pdb/static.do?p=general_information/about_pdb/index.html. [Accessed: 28-May-2020].
[100] “eSOL(Solubility database of all E.coli proteins).” [Online]. Available: http://www.tanpaku.org/tp-esol/index.php?lang=en. [Accessed: 28-May-2020].

Anexos

Anexo A: Descripción detallada de los conjuntos de datos

A continuación, se describen los conjuntos de datos a emplear en el trabajo presente. Cada repositorio posee únicamente secuencias de péptidos/proteínas con aminoácidos naturales. Los nombres de los conjuntos de datos se decidieron en base a uno de dos criterios: (1) nombre del modelo de la referencia de la que fueron extraídas, (2) alguna sigla que pueda dar una idea global del conjunto de datos.

AntiTb_primary

Extraído desde [23]. El problema que se apuntó a resolver fue poder clasificar secuencias en péptidos antituberculares y no-antituberculares. El conjunto de datos se compone de dos clases:

Positiva: Péptidos antituberculares. Corresponde a una base de datos actualizada de manera manual y experimentalmente verificada. La mayoría de los péptidos contienen modificaciones no-naturales. Efectivos contra *Mycobacterium*. Este grupo figura como conjunto de clasificación positiva.

Negativa: Péptidos antibacterianos. Los datos fueron extraídos de DBAASP (una base de datos de péptidos antimicrobianos) [94]. Eficaces contra bacterias Gram positivas y Gram negativas.
Cada uno de ellos posee 246 secuencias únicas que varían en un largo entre 5 – 61 aminoácidos. Asimismo, se mantuvo una distribución de tamaño similar mediante la generación de diferentes compartimentos (5 – 14, 15 – 24, etc.).

AntiTb_secondary

Similar a AntiTb_primary, pero con una clase negativa distinta. Se aseguró que no existiera intersección entre la clase negativa de AntiTb_primary y la presente.

Negativa: Péptidos no-antituberculares. Las secuencias fueron aleatoriamente generadas desde la base de datos Swiss-Prot. Este grupo figura como un conjunto de clasificación negativa.

ACP-DL

Extraído desde [24], cuyo objetivo fue la clasificación de péptidos en anticancerígenos o no-anticancerígenos. Existen dos subconjuntos independientes:

ACP-DL_740: Corresponden a ejemplares seleccionados de los estudios de [27]. El conjunto consta de una mezcla de péptidos anticancerígenos y no-anticancerígenos. La selección en fue hecha en base a la extracción de ejemplares de la base de datos de péptidos antimicrobianos [95] y de literatura disponible; asimismo las secuencias de clasificación negativa se obtuvieron de Universal Protein Source [96]. También se extrajeron los

péptidos de UniProtKB [97]. En total, el conjunto de datos ACP-DL_740 posee 740 secuencias que varían en largo de 11 – 97 residuos.

ACP-DL_240: Posee características de selección similares a ACP-DL_740. El conjunto consta de 240 ejemplares, de los cuales 129 son péptidos anticancerígenos experimentalmente validados y 111 son AMPs sin actividad anticancerígena. Las secuencias varían en una longitud entre 11 – 207 aminoácidos.

No existe intersección entre los dos subconjuntos. Además, se verificó en cada uno que no existiera similitud mayor a un 90% entre péptidos.

iACP

Extraído de [27]. El problema a resolver es la identificación de péptidos anticancerígenos. Se formularon dos subconjuntos independientes:

iACP_benchmark: Consta de 138 péptidos anticancerígenos y 206 péptidos no-anticancerígenos, extraídos desde la base de datos Universal Protein Source [96]. El tamaño de las secuencias varía entre 11 – 138 aminoácidos.

iACP_independent: Recoge a 150 péptidos anticancerígenos y 150 péptidos no-anticancerígenos. No se detalla el método de construcción del grupo. El largo de las cadenas varía entre 11 – 128 aminoácidos.
No existe intersección entre los dos subconjuntos. Asimismo, se garantizó que no hubiera homología mayor a 90% entre péptidos.

QSP

Extraído de [29] y reportado adicionalmente en [43]. El propósito del trabajo fue identificar péptidos percebidores de cuórum (QSP). Se tienen dos clases:

Positiva: Corresponde al conjunto clasificado como QSP. Fue obtenido de la base de datos Quorumpeps [98]. El largo de las secuencias incluye largo entre 5 – 48 residuos.

Negativa: Engloba a los péptidos clasificados como no-QSP. Se construyó mediante búsqueda en UniProt [96]. Figuran como péptidos de bacterias Gram positivas sin capacidad de percibir cuórums. El tamaño de las cadenas se extiende entre 7 – 60 aminoácidos.
Ambos subconjuntos constan de 220 péptidos únicos, sin intersecciones entre sí.

iAMP-2L_multiclass

Extraído de [33]. El modelo de la referencia tuvo el objetivo de clasificar secuencias aminoacídicas en péptidos antibacterianos, anticancerígenos/antitumorales, antifúngicos, anti-VIH, antivirales, o no-antimicrobianos. Se tienen 6 subconjuntos que conforman las distintas clases:

iAMP-2L_AB: 769 péptidos antibacterianos. Largo varía entre 5 – 107 residuos.

iAMP-2L_AC: 140 péptidos anticancerígenos. Largo varía entre 5 – 98 residuos.

iAMP-2L_AF: 366 péptidos antifúngicos. Largo varía entre 6 – 105 residuos.

iAMP-2L_AHIV: 86 péptidos anti-VIH. Largo varía entre 10 – 46 residuos.

iAMP-2L_AV: 124 péptidos antivirales. Largo varía entre 5 – 98 residuos.

Todos los cinco subconjuntos recién mencionados fueron armados a partir de Antimicrobial Peptide Database. La referencia [33] indica que se realizó un filtro a estos subconjuntos para asegurar homología menor a un 40% entre los péptidos de un mismo subconjunto, sin embargo, dicho proceso se llevó a cabo únicamente a los grupos con más de 150 péptidos previo la eliminación. La referencia no indica cuáles subconjuntos presentan reducción de similitud. Existe intersección entre los grupos, del total de 878 péptidos reales: 454 pertenecen a una única clase, 296 a dos, 85 a tres, 30 a cuatro, y 13 a cinco.

iAMP-2L_b_nonAMP: 2405 péptidos sin actividad antimicrobiana. Se extrajeron de UniProt y se aseguró homología menor a 40%. El largo varía de 5 – 100 residuos.

iAMP-2L_binary

El conjunto se construyó a partir de los péptidos eliminados de los subconjuntos de iAMP-2L_multiclass para asegurar un máximo porcentaje de homología. El tamaño de las cadenas se extiende entre 5 – 131 aminoácidos. Consta de dos clases:

Positiva: 920 péptidos antimicrobianos no clasificados en algún atributo específico.
Negativa: 920 péptidos sin actividad antimicrobiana.

DBP

Extraído de [47]. El propósito del trabajo de la referencia fue clasificar la capacidad de proteínas de unirse a moléculas de ADN (*DNA-binding protein*, DBP). Se tienen dos clases:

Positiva: 524 DBPs. Se recopilaron de la base de datos Protein Data Bank [99]. Las secuencias varían en largo de 50 – 2226 aminoácidos.

Negativa: 550 no-DBPs. Se recopilaron aleatoriamente de la base de datos Protein Data Bank. Las secuencias varían en largo de 62 – 718 residuos.

En ambos conjuntos de datos se aseguró que no hubiese homología mayor al 25%.

Pop
Extraído de [80]. La idea central de la referencia fue predecir la observabilidad de péptidos en distintos organismos. Con "observabilidad", se hace referencia a la presencia de dichas cadenas por digestión de proteínas u otros péptidos. El estimador de observabilidad empleado fue el área de los cromatorgramas de iones, por lo que un ranking (valor) más bajo indica una alta abundancia del péptido. Se tienen tres subconjuntos independientes:

Pop_ara: 60288 péptidos de *Arabidopsis thaliana*. Poseen largos de 6 – 315 residuos.

Pop_chlamy: 97757 péptidos de *Chlamydomonas reinhardtii.* Presentan cadenas de 7 – 2984 aminoácidos.

Pop_yeast: 94320 péptidos de *Saccharomyces cerevisiae*. Poseen largos de 7 – 356 residuos.

VaxinPad
Extraído de [83], el cual tenía el objetivo de clasificar distintas secuencias cortas como péptidos inmunomoduladores. Consta de dos clases:

Positiva: 304 péptidos inmunomoduladores experimentalmente validados en diversas patentes. El tamaño de las secuencias varía entre 3 – 30 aminoácidos.

Negativa: 385 péptidos asumidos como no-inmunomoduladores. Específicamente, estas secuencias corresponden a péptidos de suero endógeno humano. Poseen largos entre 4 – 30 residuos.

Solub
Extraído de [84]. El problema por resolver es la predicción de la solubilidad de proteínas. Con "solubilidad" se hace referencia a la razón entre proteína soluble y proteína total. Se obtuvieron las secuencias de la base de datos eSOL [100]. El conjunto consta de 3148 proteínas que poseen largo variante entre 29 – 1367 residuos.

Enantioselectivity
Extraído de [85]. El trabajo de la referencia fue predecir la enantioselectividad de epóxido hidrolasa (EH) de *Aspergillus niger* para mejorar su preferencia por el (S)-enantiómero de glicidil fenil éter. El conjunto de datos se obtuvo a partir de 1 – 8 mutaciones puntuales en el sitio de unión de EH, logrando así un total de 152 secuencias de 389 residuos de largo.

Localization
Extraído de [85]. La tarea de la referencia fue predecir la localización de membrana de la proteína ChR (canalrodopsina) en células embriónicas humanas de riñón. Se construyeron variaciones quiméricas de ChR a partir de tres proteínas parentales:

CheRiff, C1C2 y CsChrimsonR. El conjunto consta de 248 secuencias que varían en largo entre 329 – 361 residuos, junto con su valor de localización de membrana.

T50

Extraído de [85]. El objetivo fue predecir la temperatura T50 de la proteína citocromo P450 en *Escherichia coli*. T50 se define como la temperatura a la cual la mitad de las proteínas son irreversiblemente denaturadas luego de 10 minutos de incubación. El conjunto de datos se formó mediante la variación quimérica de citocromo P450 a partir de tres proteínas parentales: CYP102A1, CYP102A2 y CYP102A3. De esta forma, se logró 242 secuencias. Adicionalmente, se generaron 19 citocromo P450s quiméricos de otras proteínas parentales. En total, se dispone de 261 secuencias de 466 aminoácidos de largo.

RT

Extraído de [79], cuyo objetivo fue la determinación del tiempo de retención en cromatografías de péptidos. Se disponen de 2 subconjuntos independientes provenientes de distintas especies/líneas celulares, o determinados en distintas columnas de cromatografía.

RT_Yeast: Péptidos de levadura. 14361 secuencias. Realizado en condiciones de RPLC (*Reversed phase liquid chromatography*). El largo de las cadenas varía entre 6 – 38 residuos.

RT_Hela: Péptidos de la línea células HeLa. 3412 secuencias. Realizado en condiciones de RPLC. Presenta modificaciones de algunos aminoácidos. El largo de las cadenas varía entre 7 – 50 residuos.

Anexo B: Histogramas de largo de secuencias para los conjuntos fuera de los casos de estudio

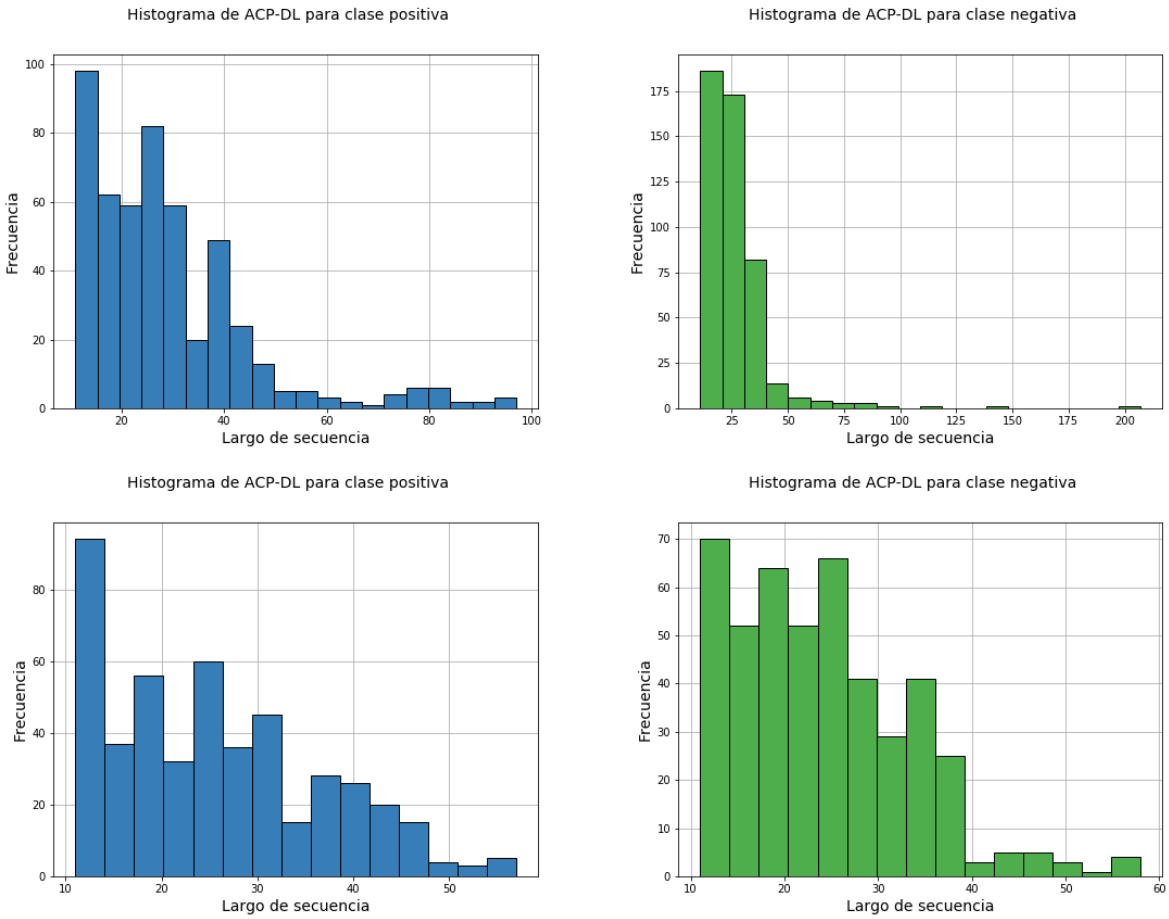


Figura 48: Histogramas de largo de secuencias para el conjunto ACP-DL. Arriba: previo a eliminar outliers.

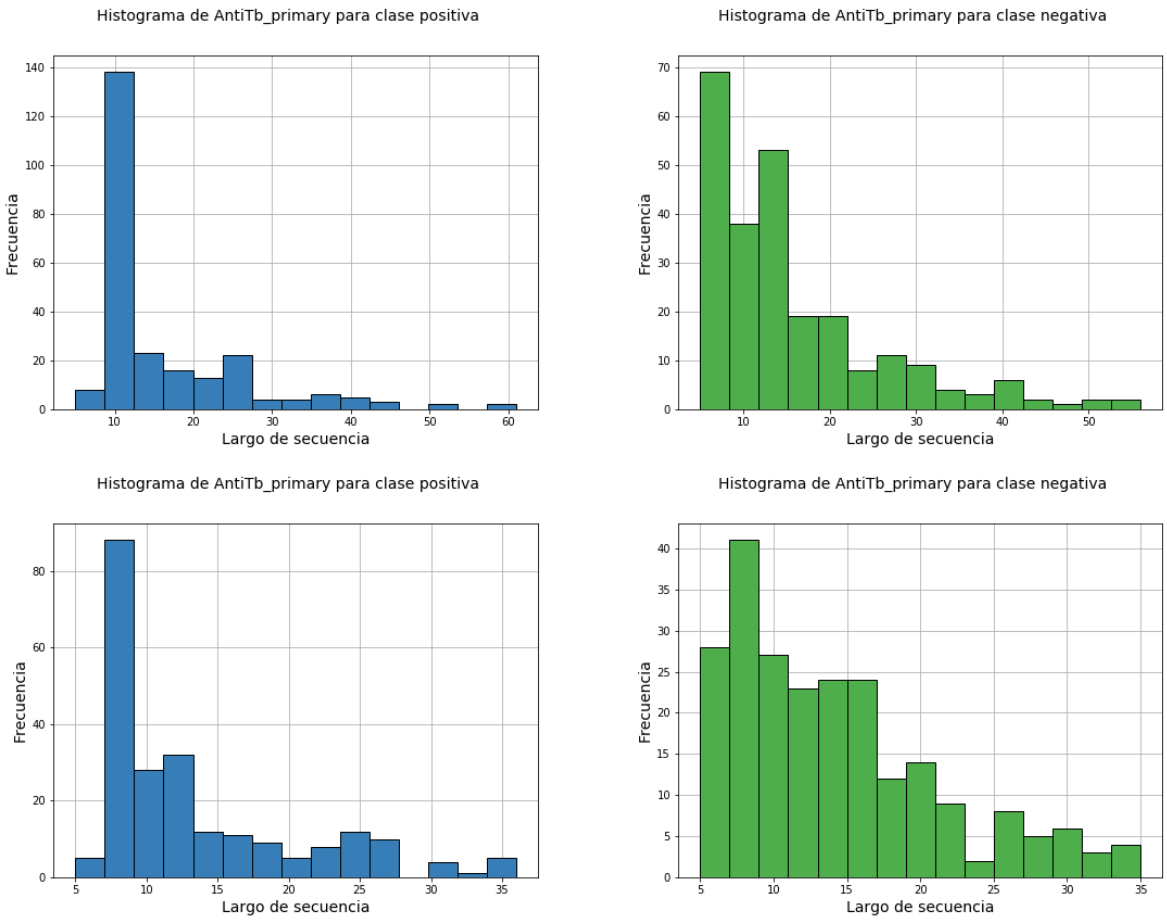
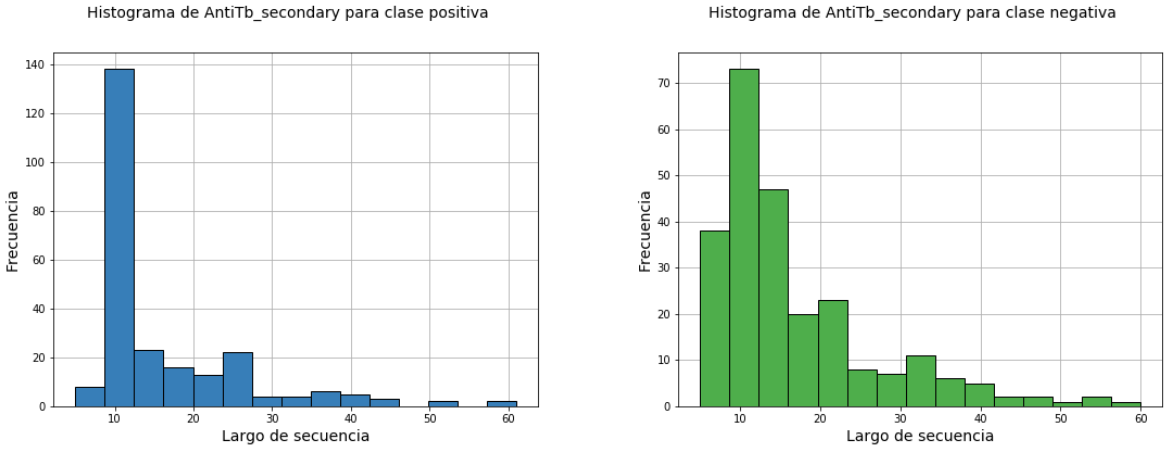


Figura 49: Histogramas de largo de secuencias para el conjunto AntiTb_primary. Arriba: previo a eliminar outliers.



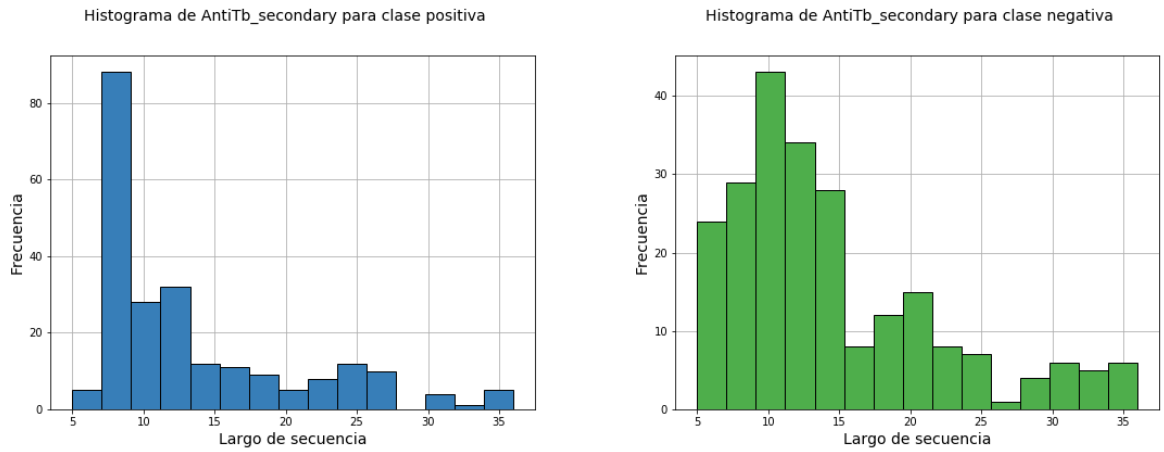


Figura 50: Histogramas de largo de secuencias para el conjunto AntiTb_secondary. Arriba: previo a eliminar outliers.

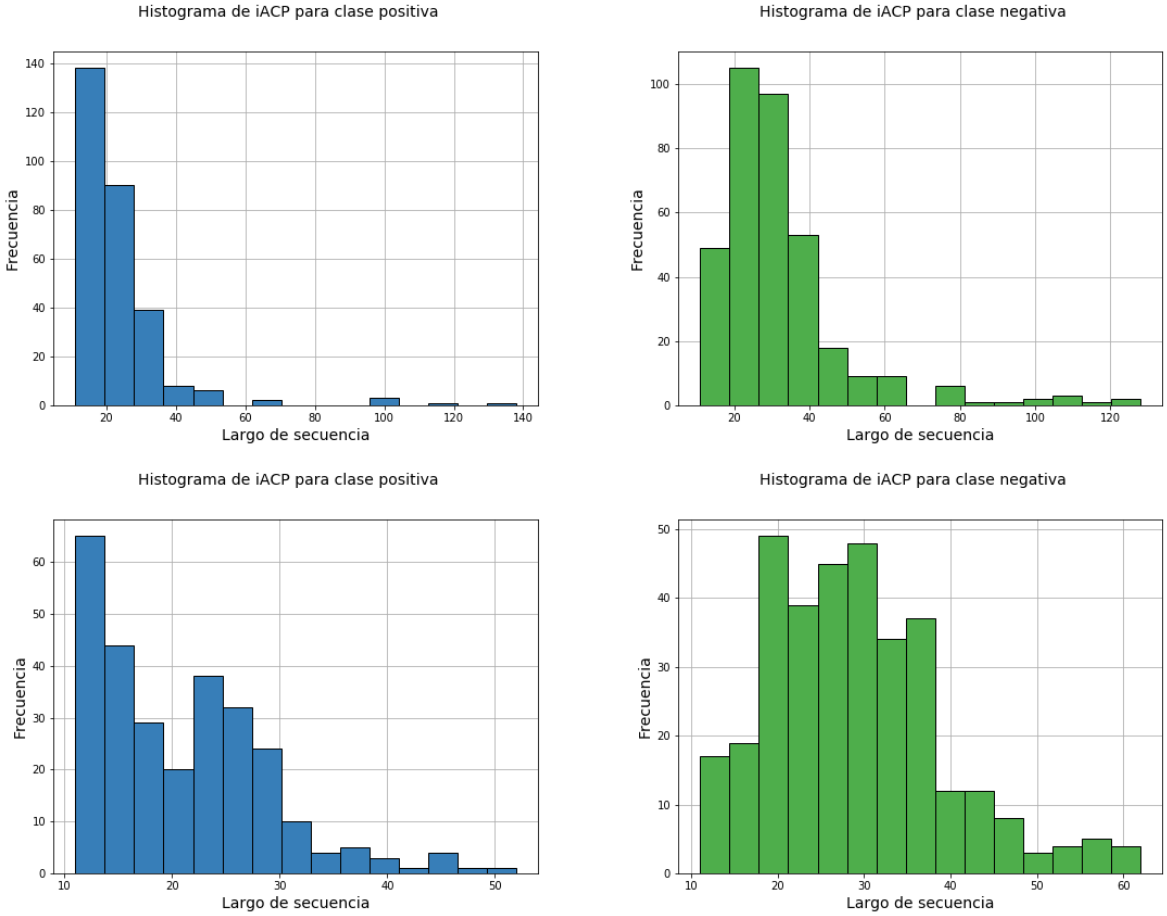


Figura 51: Histogramas de largo de secuencias para el conjunto iACP. Arriba: previo a eliminar outliers.

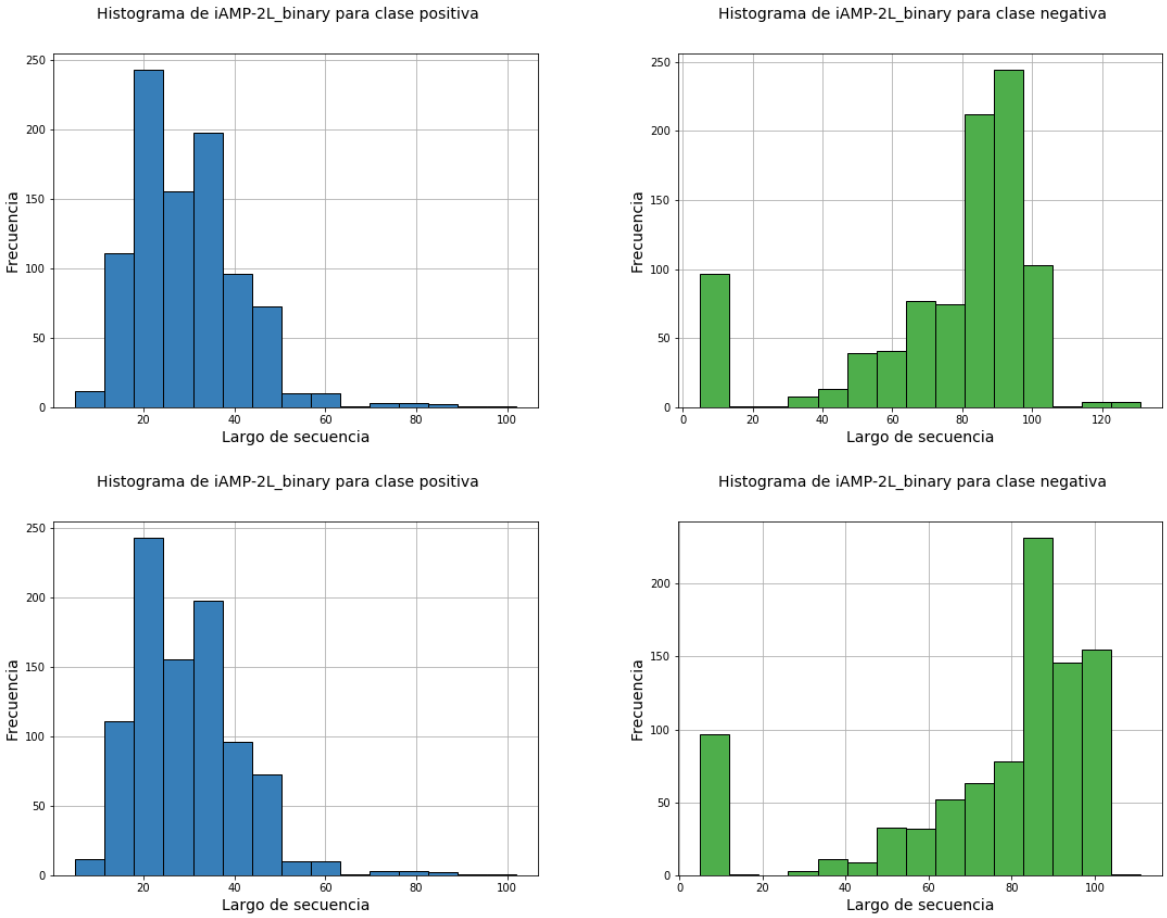


Figura 52: Histogramas de largo de secuencias para el conjunto iAMP-2L_binary. Arriba: previo a eliminar outliers.

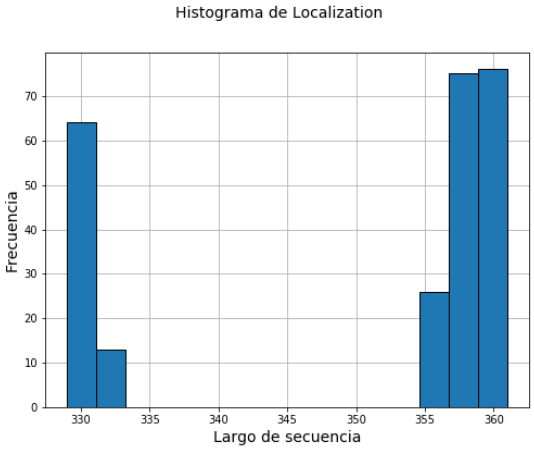


Figura 53: Histograma de largo de secuencias para el conjunto localization.

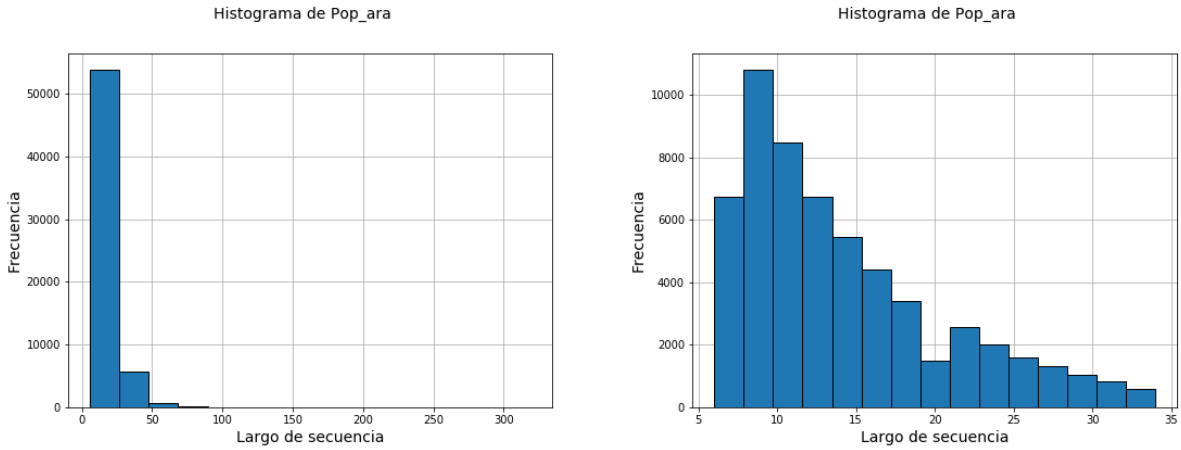


Figura 54: Histogramas de largo de secuencias para Pop_ara. Izquierda: previo a eliminar outliers.

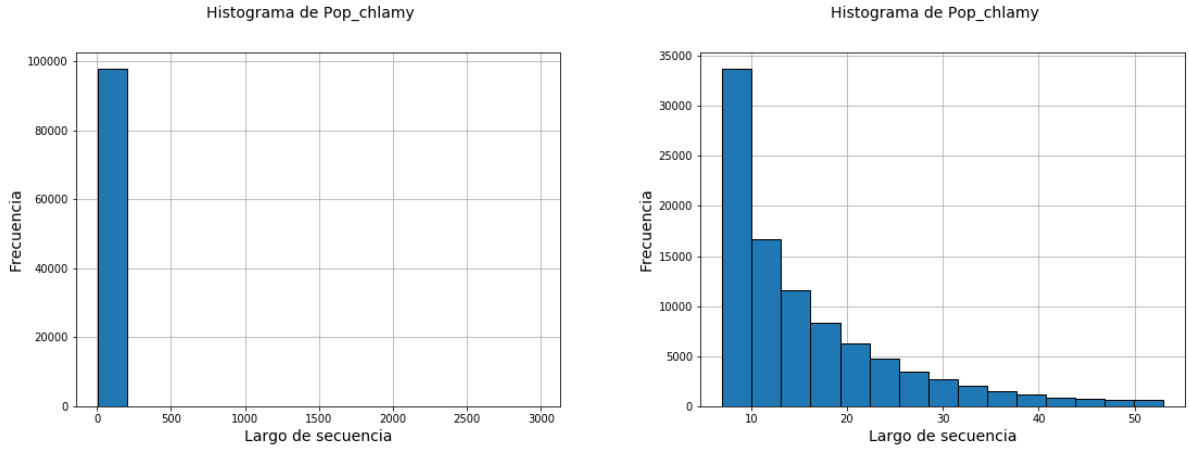


Figura 55: Histogramas de largo de secuencias para Pop_chlamy. Izquierda: previo a eliminar outliers.

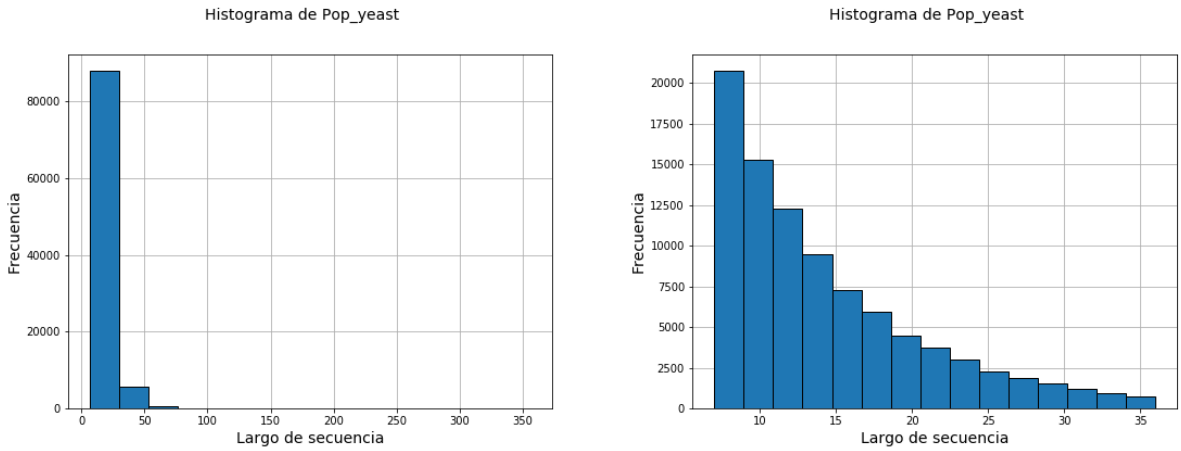


Figura 56: Histogramas de largo de secuencias para Pop_yeast. Izquierda: previo a eliminar outliers.

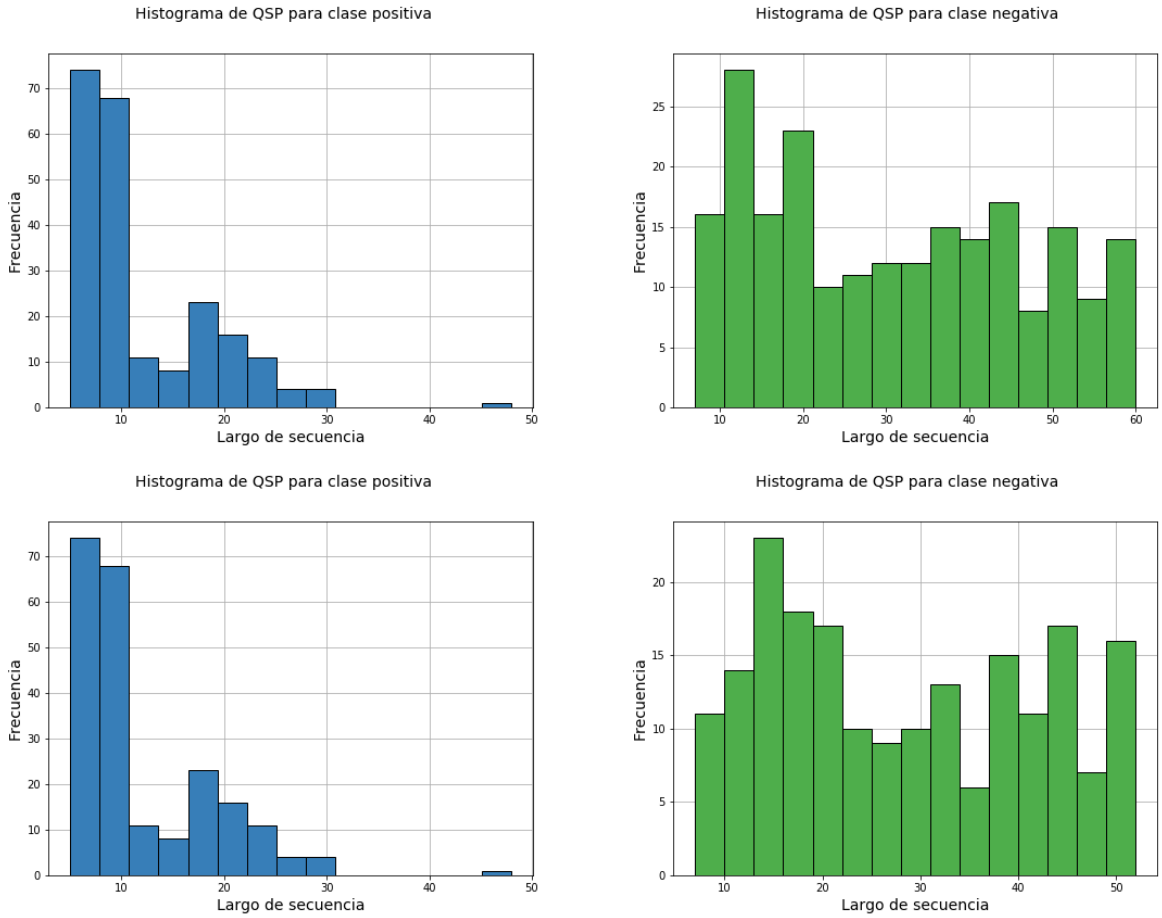


Figura 57: Histogramas de largo de secuencias para el conjunto QSP. Arriba: previo a eliminar outliers.

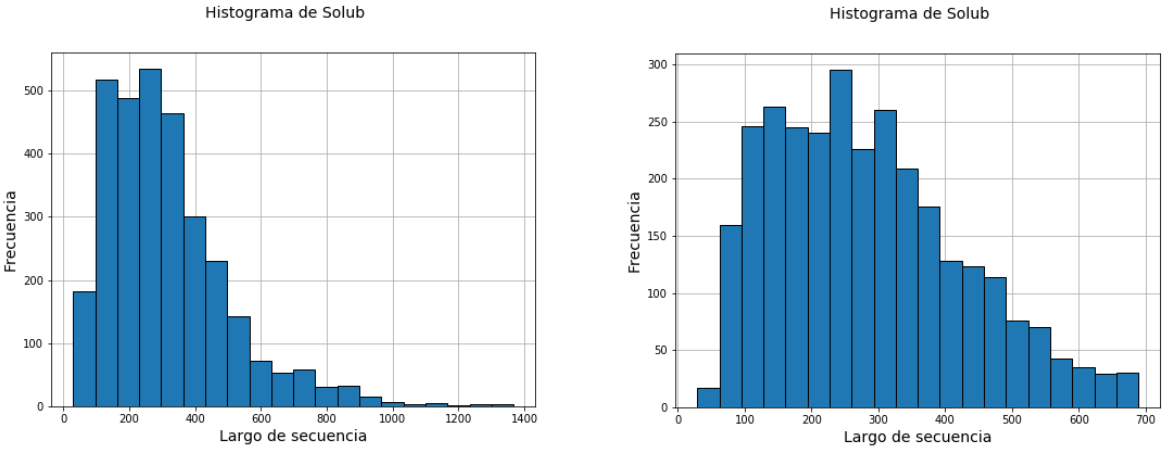


Figura 58: Histogramas de largo de secuencias para Solub. Izquierda: previo a eliminar outliers.

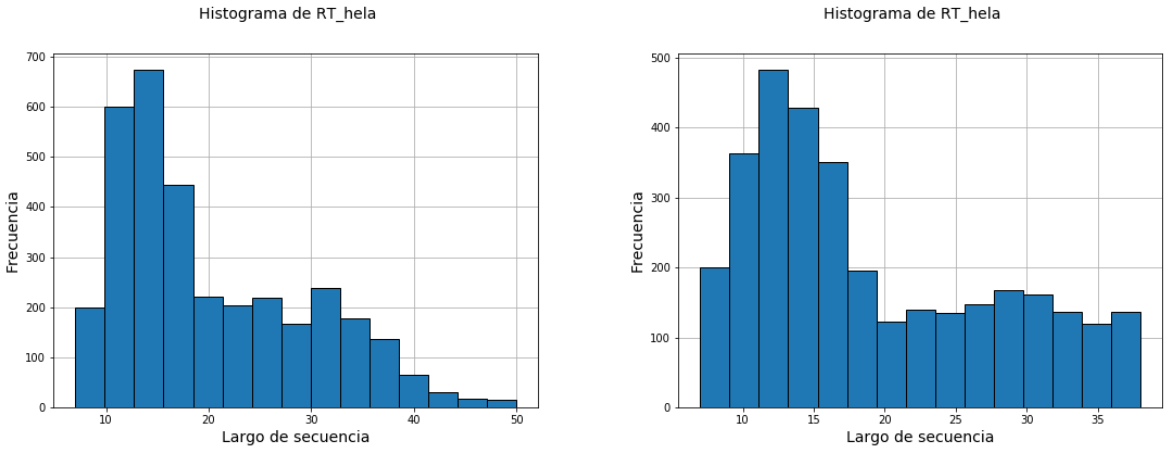


Figura 59: Histogramas de largo de secuencias para el conjunto RT_hela. Izquierda: previo a eliminar outliers.

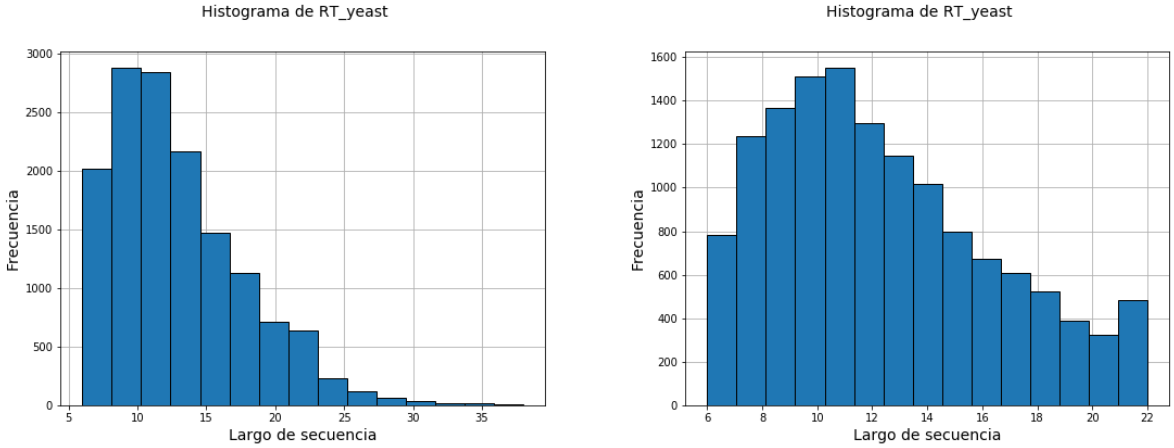


Figura 60: Histogramas de largo de secuencias para el conjunto RT_yeast. Izquierda: previo a eliminar outliers.

Anexo C: Histogramas de respuestas para los conjuntos de regresión fuera de los casos de estudio

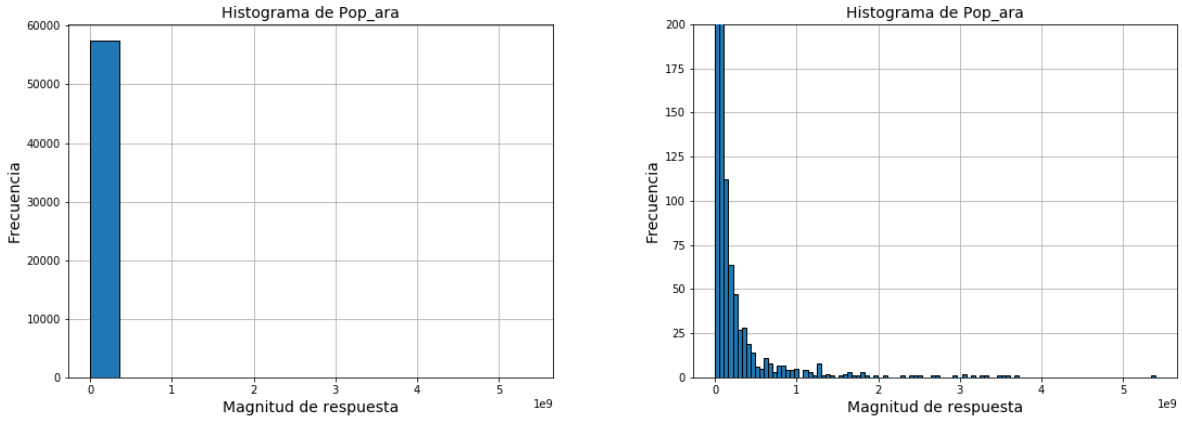


Figura 61: Histogramas de la magnitud de respuestas para el conjunto Pop_ara. Derecha: corresponde a un acercamiento al gráfico de la izquierda.

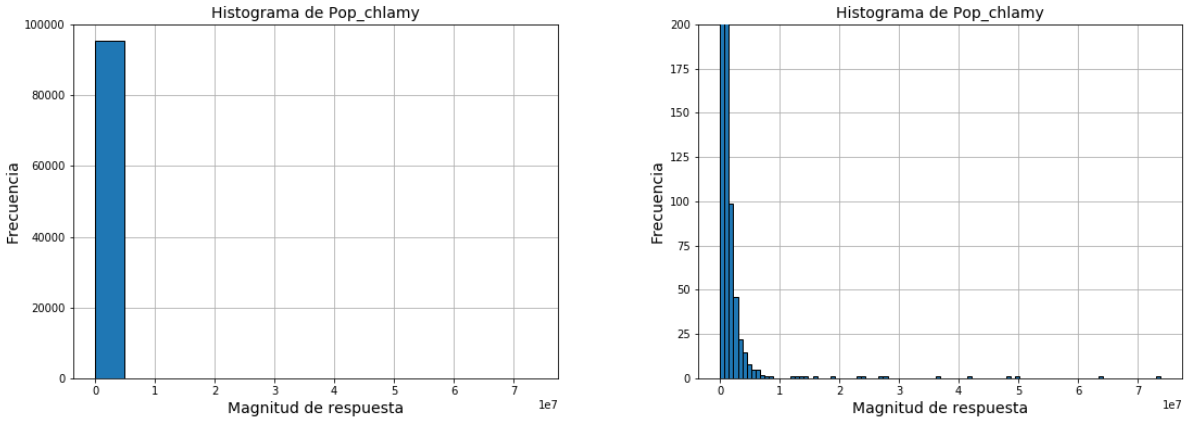


Figura 62; Histogramas de la magnitud de respuestas para el conjunto Pop_chlamy. Derecha: corresponde a un acercamiento al gráfico de la izquierda.

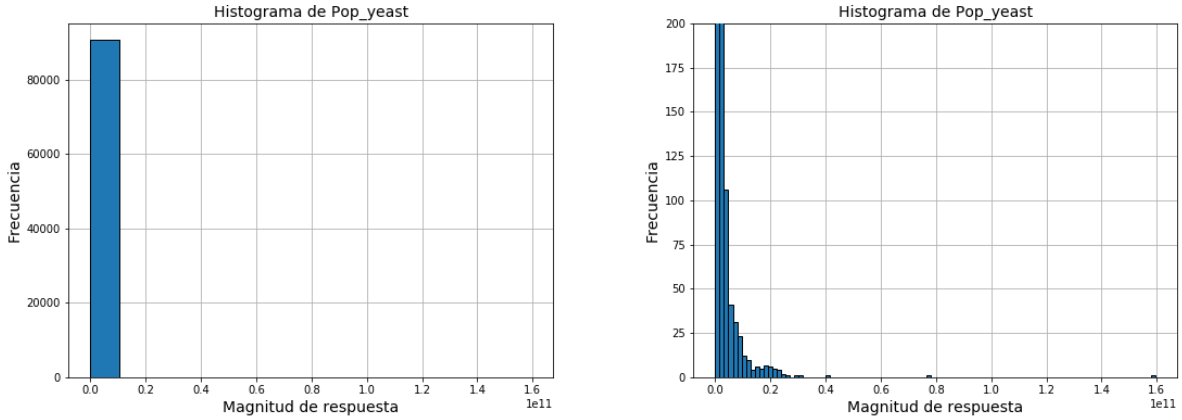


Figura 63: Histogramas de la magnitud de respuestas para el conjunto Pop_yeast. Derecha: corresponde a un acercamiento al gráfico de la izquierda.

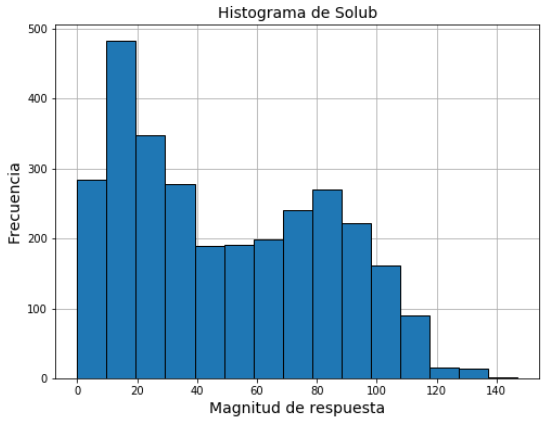


Figura 64: Histogramas de la magnitud de respuestas para el conjunto Solub.

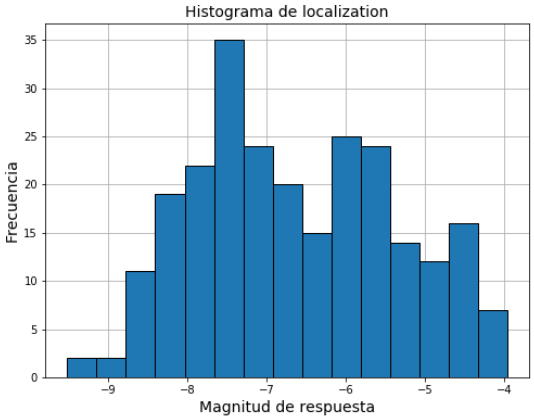


Figura 65: Histogramas de la magnitud de respuestas para el conjunto localization.

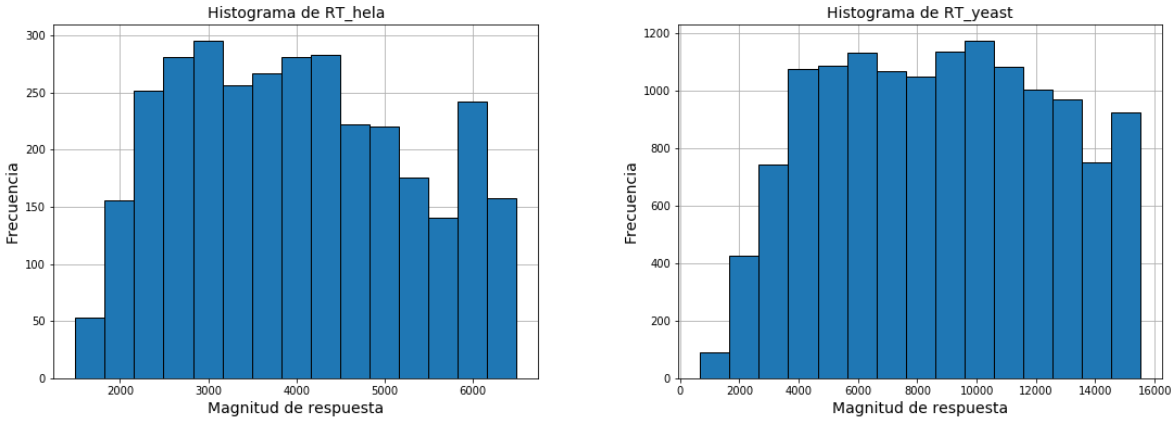


Figura 66: Histogramas de la magnitud de respuestas para los conjuntos (izquierda) RT_hela y (derecha) RT_yeast.

Anexo D: Hiperparámetros estimados para cada modelo

Random Forest

Para determinar los parámetros, se empleó una búsqueda por grilla (*grid search*), determinando 4 parámetros para la función *RandomForestClassifier* y *RandomForestRegressor* de *sklearn*. Los parámetros a determinar y sus respectivas grillas son (en formato [mínimo:paso:máximo]):

- n_estimators: [25:25:400]
- max_features: [1:1:30]
- min_sample_leag: [1:1:10]
- min_sample_split: [2:1:20]

Tabla 15: Hiperparámetros para Random Forest utilizando codificación por digitalización.

Conjunto	estimators	max features	min sample leaf	min samples split
iAMP-2L	375	8	1	5
VaxinPad	100	11	3	18
DBP	300	18	5	13
enantioselectivity	50	7	1	2
t50	125	13	3	8

Tabla 16: Hiperparámetros para Random Forest utilizando codificación Onehot.

Conjunto	estimators	max features	min sample leaf	min samples split
iAMP-2L	400	15	1	3
VaxinPad	350	7	1	3
DBP	400	21	8	12
enantioselectivity	375	7	1	3
t50	350	14	2	4

Tabla 17: Hiperparámetros para Random Forest utilizando codificación Ordinal.

Conjunto	estimators	max features	min sample leaf	min samples split
iAMP-2L	225	27	1	4
VaxinPad	125	24	1	3
DBP	275	12	9	6
enantioselectivity	100	17	2	5
t50	325	29	1	3

Tabla 18: Hiperparámetros para Random Forest utilizando codificación AAC.

Conjunto	estimators	max features	min sample leaf	min samples split
iAMP-2L	325	14	2	4
VaxinPad	175	2	3	2
DBP	325	11	8	6
enantioselectivity	250	15	5	13
t50	100	8	1	3

Tabla 19: Hiperparámetros para Random Forest utilizando codificación por composición de dipéptidos.

Conjunto	estimators	max features	min sample leaf	min samples split
iAMP-2L	175	13	1	7
VaxinPad	300	6	1	3
DBP	300	28	3	3
enantioselectivity	50	29	1	2
t50	375	20	1	3

Support Vector Machine

Para determinar los parámetros, se empleó una búsqueda por grilla (*grid search*), determinando 4 parámetros para la función *SVC* y *SVR* de *sklearn*. Los parámetros a determinar y sus respectivas grillas son:

- C, utilizando los valores 1e-4, 1e-3, 1e-2, 1, 1e+1, 1e+2, 1e+3 y 1e+4
- Función kernel, pudiendo ser entre *linear, rbf, poly* o *sigmoid*.
- Gamma para el caso de kernels no-lineales: 1e-4, 1e-3, 1e-2, 1e-1, 1 1e+1, 1e+2, 1e+3 y 1e+4.

Tabla 20: Hiperparámetros para Support Vector Machine utilizando codificación por digitalización.

Conjunto	C	kernel	gamma
iAMP-2L	1	rbf	1000
VaxinPad	100	rbf	100
DBP	10	rbf	100
enantioselectivity	30	linear	-
t50	10	poly	0.1

Tabla 21: Hiperparámetros para Support Vector Machine utilizando codificación Onehot.

Conjunto	C	kernel	gamma
iAMP-2L	10	linear	-
VaxinPad	10	rbf	0.1
DBP	10	rbf	0.01
enantioselectivity	100	poly	0.01
t50	0.1	poly	0.01

Tabla 22: Hiperparámetros para Support Vector Machine utilizando codificación Ordinal.

Conjunto	C	kernel	gamma
iAMP-2L	1000	rbf	10
VaxinPad	10	rbf	0.1
DBP	0.1	rbf	0.01
enantioselectivity	100	poly	0.1
t50	0.1	poly	0.1

Tabla 23: Hiperparámetros para Support Vector Machine utilizando codificación AAC.

Conjunto	C	kernel	gamma
iAMP-2L	0.1	poly	10
VaxinPad	10000	rbf	10
DBP	10	rbf	10
enantioselectivity	100	poly	10
t50	1	poly	10

Tabla 24: Hiperparámetros para Support Vector Machine utilizando codificación por composición de dipéptidos.

Conjunto	C	kernel	gamma
iAMP-2L	1	sigmoid	1
VaxinPad	10000	rbf	10
DBP	1	rbf	100
enantioselectivity	100	linear	-
t50	10	poly	0.1

K-Nearest Neighbor

Para determinar el parámetro de K en las funciones *KNeighborsClassifier* y *KNeighborsRegressir*, se itera desde K=1 hasta K=40 y se registra aquel valor que maximice el desempeño en la etapa de validación cruzada.

Tabla 25: Valores de K para el algoritmo KNN, para cada codificación.

Conjunto	Onehot	Ordinal	AAC	Comp. dipéptidos	PC
iAMP-2L	18	16	4	3	17
VaxinPad	3	3	7	9	1
DBP	2	11	5	1	10
enantio	8	3	11	2	2
t50	4	5	8	6	5

Selección del algoritmo

Una vez se tienen los parámetros que logren mejores desempeños en cada algoritmo, éstos se comparan entre sí para determinar el mejor candidato de prueba para cada codificación.

Tabla 26: Rendimientos de los mejores modelos para la codificación Onehot. Se muestra la Accuracy [%] para los casos de clasificación y el R2 para los de regresión.

Conjunto	ANN	CNN	RF	SVM	KNN
iAMP-2L	73.35 ± 1.75	71.61 ± 0.83	66.01 ± 0.91	60.70 ± 1.41	21.21 ± 1.31
VaxinPad	90.75 ± 3.57	89.02 ± 3.06	94.42 ± 1.05	93.07 ± 1.11	84.97 ± 3.32
DBP	69.18 ± 4.91	66.27 ± 4.03	66.38 ± 1.83	66.50 ± 4.01	49.69 ± 0.90
enantioselectivity	0.780 ± 0.093	0.718 ± 0.193	0.622 ± 0.139	0.830 ± 0.099	0.630 ± 0.185
t50	0.813 ± 0.108	0.803 ± 0.083	0.584 ± 0.095	0.857 ± 0.038	0.609 ± 0.113

Tabla 27: Rendimientos de los mejores modelos para la codificación Ordinal. Se muestra la Accuracy [%] para los casos de clasificación y el R2 para los de regresión.

Conjunto	ANN	CNN	RF	SVM	KNN
iAMP-2L	68.12 ± 1.67	67.69 ± 2.16	64.85 ± 1.17	61.30 ± 0.22	66.07 ± 0.28
VaxinPad	79.39 ± 2.84	80.73 ± 3.95	86.90 ± 1.57	72.26 ± 3.61	72.45 ± 4.46
DBP	67.11 ± 2.15	68.21 ± 2.42	66.51 ± 2.56	64.43 ± 2.11	60.78 ± 3.09
enantioselectivity	0.211 ± 0.276	0.494 ± 0.288	0.617 ± 0.157	0.337 ± 0.222	0.193 ± 0.300
t50	0.802 ± 0.083	0.759 ± 0.108	0.726 ± 0.126	0.831 ± 0.054	0.452 ± 0.196

Tabla 28: Rendimientos de los mejores modelos para la codificación AAC. Se muestra la Accuracy [%] para los casos de clasificación y el R2 para los de regresión.

Conjunto	ANN	CNN	RF	SVM	KNN
iAMP-2L	72.92 ± 0.66	72.79 ± 0.95	66.30 ± 0.83	63.01 ± 1.49	64.56 ± 0.85
VaxinPad	94.22 ± 1.07	93.87 ± 1.97	94.61 ± 1.30	93.64 ± 2.22	92.29 ± 2.43
DBP	79.04 ± 2.01	77.34 ± 0.72	75.28 ± 4.18	75.77 ± 3.58	68.58 ± 2.83
enantioselectivity	-0.002 ± 0.109	-0.003 ± 0.007	0.664 ± 0.142	0.676 ± 0.141	0.487 ± 0.155
t50	0.228 ± 0.221	0.443 ± 0.114	0.673 ± 0.072	0.823 ± 0.095	0.468 ± 0.300

Tabla 29: Rendimientos de los mejores modelos para la codificación por composición de dipéptidos. Se muestra la Accuracy [%] para los casos de clasificación y el R2 para los de regresión.

Conjunto	ANN	CNN	RF	SVM	KNN
iAMP-2L	72.89 ± 0.82	71.51 ± 0.68	65.61 ± 0.97	67.23 ± 1.52	65.09 ± 1.30
VaxinPad	95.18 ± 2.38	91.71 ± 2.19	96.34 ± 2.05	93.25 ± 2.86	89.22 ± 2.66
DBP	73.98 ± 1.92	73.92 ± 1.88	71.02 ± 5.05	72.83 ± 4.76	59.08 ± 3.71
enantioselectivity	0.685 ± 0.185	0.563 ± 0.317	0.748 ± 0.173	0.667 ± 0.127	0.626 ± 0.483
t50	0.769 ± 0.173	0.755 ± 0.107	0.742 ± 0.107	0.837 ± 0.095	0.561 ± 0.144

Anexo E: Código

Formas de codificación

```
import pandas as pd
import numpy as np
import os
import math as mt
import scipy as sc
import matplotlib.pyplot as plt

def onehot(seq_list,alphabet):
    length_seq = []
    for i in range(0,len(seq_list)):
        length_seq += [len(seq_list.iloc[i])]
    max_len = max(length_seq)
    onehot_seq = []
    for i in range(0,len(seq_list)):
        seq_str = seq_list.iloc[i]
        seq_str = seq_str + '-'*(max_len - len(seq_str))
        onehot_matrix = np.zeros((len(alphabet),max_len), dtype=int)
        for j in range(0,len(seq_str)):
            ind = alphabet.find(seq_str[j])
            if ind!=-1:
                onehot_matrix[ind,j] = 1
        onehot_matrix = onehot_matrix.transpose()
        onehot_list = []
        for k in range(0,len(seq_str)):
            for j in range(0,len(alphabet)):
                onehot_list += [onehot_matrix[k,j]]
        onehot_seq.append(onehot_list)
    return onehot_seq

def ordinal(seq_list,alphabet):
    length_seq = []
    for i in range(0,len(seq_list)):
        length_seq += [len(seq_list.iloc[i])]
    max_len = max(length_seq)
    ordinal_seq = []
    for i in range(0,len(seq_list)):
        seq_str = seq_list.iloc[i]
        seq_str = seq_str + '-'*(max_len - len(seq_str))
        ordinal_vec = []
        for j in range(0,len(seq_str)):
            ind = alphabet.find(seq_str[j])
            if ind!=-1:
                ordinal_vec += [ind+1]
            else:
                ordinal_vec += [0]
        ordinal_seq.append(ordinal_vec)
    return ordinal_seq


def composition(seq_list,alphabet):
    AAC_list = []
    if type(seq_list)==str:
        n=1
    else:
        n=len(seq_list)
    for i in range(0,n):
        AAC_vec = [0]*len(alphabet)
```

```
        if type(seq_list)==str:
            seq_str = seq_list
        else:
            seq_str = seq_list.iloc[i]
        for j in range(0,len(seq_str)):
            ind = alphabet.find(seq_str[j])
            if ind!=-1:
                AAC_vec[ind] = AAC_vec[ind]+1
        N = len(seq_str)
        AAC_vec = [AAC_vec[i]/N for i in range(0,len(AAC_vec))]
        if type(seq_list)==str:
            for k in range(0,len(AAC_vec)):
                AAC_list += [AAC_vec[k]]
        else:
            AAC_list.append(AAC_vec)
    return AAC_list


def PC_prop(seq_list,alphabet,AAindex_loc,PCprop):
    AAindex = pd.read_csv(AAindex_loc)
    PCrow = AAindex.loc[AAindex['Property']==PCprop]
    PC_list = []
    for i in range(0,len(seq_list)):
        PC_vec = []
        seq_str = seq_list.iloc[i]
        for j in range(0,len(seq_str)):
            ind = alphabet.find(seq_str[j])
            if ind!=-1:
                PC_vec += PCrow[alphabet[ind]].tolist()
        PC_list.append(PC_vec)
    return PC_list

def dipeptide_comp(seq_list,alphabet):
    seq = seq_list
    dipept_list = []
    for k in range(0,len(seq)):
        dipept_comp = np.zeros((len(alphabet),len(alphabet)),dtype=int)   ## rows=first
    aa, columns=second aa
        dipept_quant = len(seq.iloc[k])-1
        for j in range(0,len(seq.iloc[k])-1):
            ind_first = alphabet.find(seq.iloc[k][j])
            ind_second = alphabet.find(seq.iloc[k][j+1])
            if ind_first!=-1 and ind_second!=-1:
                        dipept_comp[ind_first,ind_second] += 1
               dipept_comp = dipept_comp/dipept_quant
               dipept_comp = np.reshape(dipept_comp, len(alphabet)**2)
               dipept_list.append(dipept_comp)
           return dipept_list

       def digitalize(PC_list):
           length_seq = []
           for i in range(0,len(PC_list)):
               length_seq += [len(PC_list[i])]
           max_len = max(length_seq)
           pad_power = mt.ceil(mt.log(max_len, 2))
           pad_len = 2**pad_power
           nyq_lim = pad_len/2
           for i in range(0,len(PC_list)):
               PC_list[i] = PC_list[i] + [0]*(pad_len - len(PC_list[i]))
           digi_list = []
           for i in range(0,len(PC_list)):
               digi_vec = np.abs(sc.fft(PC_list[i]))
               digi_vec = digi_vec[0:int(nyq_lim)]
```

```
                digi_vec = [2*elem/pad_len for elem in digi_vec]
                digi_list.append(digi_vec)
            return digi_list


        class encoding:

            alphabet = 'ARNDCQEGHILKMFPSTWYV'

            def __init__(self,dataset_path):
                if dataset_path.endswith('.csv'):
                    self.datasets = pd.read_csv(dataset_path)
                    self.subsetnames = os.path.basename(dataset_path)[0:-4]
                    self.setname = self.subsetnames[0].split(' ', 1)[0]
                else:
                    print('ARCHIVO NO APLICABLE')

            def remove_outliers(self):
                col_names=list(self.datasets.columns.values)
                if col_names[2]=='is_outlier':
                    outliers = self.datasets['is_outlier']
                    self.datasets = self.datasets[outliers != 1]
                    self.datasets.reset_index(drop=True, inplace=True)

            def one_hot_encoding(self,exportpath):
                seq = self.datasets['sequence']
                onehot_list = onehot(seq,self.alphabet)
                col_names = list(self.datasets.columns.values)
                columns = ['p'+str(k) for k in range(0,len(onehot_list[0]))]
                rows = [k for k in range(0,len(seq))]
                datacells = np.zeros((len(seq),len(onehot_list[0])))
                for k in range(0,len(seq)):
                    for j in range(0,len(onehot_list[0])):
                        datacells[k,j] = onehot_list[k][j]
                df = pd.DataFrame(datacells,index=rows,columns=columns)
                df.insert(0,col_names[1],self.datasets[col_names[1]])
                if exportpath!='':
                    outfilename = exportpath + 'One_hot ' + self.subsetnames + '.csv'
                    outfile = open(outfilename, 'wb')
                    df.to_csv(outfilename, index = False, header = True, sep = ',', enco
    ding = 'utf-8')
                    outfile.close()
                else:
                    self.onehot_enc = df
                    return self.onehot_enc

            def ordinal_encoding(self,exportpath):
                seq = self.datasets['sequence']
                ordinal_list = ordinal(seq,self.alphabet)
                columns = ['p'+str(k) for k in range(0,len(ordinal_list[0]))]
                col_names = list(self.datasets.columns.values)
                rows = [k for k in range(0,len(seq))]
                datacells = np.zeros((len(seq),len(ordinal_list[0])))
                for k in range(0,len(seq)):
                    for j in range(0,len(ordinal_list[0])):
                        datacells[k,j] = ordinal_list[k][j]
                df = pd.DataFrame(datacells,index=rows,columns=columns)
                df.insert(0,col_names[1],self.datasets[col_names[1]])
                if exportpath!='':
                    outfilename = exportpath + 'Ordinal ' + self.subsetnames + '.csv'
                    outfile = open(outfilename, 'wb')
```

```
                    df.to_csv(outfilename, index = False, header = True, sep = ',', enco
    ding = 'utf-8')
                    outfile.close()
                else:
                    self.ordinal_enc = df
                    return self.ordinal_enc

            def AAC_encoding(self,exportpath):
                seq = self.datasets['sequence']
                ACC_list = composition(seq,self.alphabet)
                columns = [letter for letter in self.alphabet]
                col_names = list(self.datasets.columns.values)
                rows = [k for k in range(0,len(seq))]
                datacells = np.zeros((len(seq),len(ACC_list[0])))
                for k in range(0,len(seq)):
                    for j in range(0,len(ACC_list[0])):
                        datacells[k,j] = ACC_list[k][j]
                df = pd.DataFrame(datacells,index=rows,columns=columns)
                df.insert(0,col_names[1],self.datasets[col_names[1]])
                if exportpath!='':
                    outfilename = exportpath + 'AAC ' + self.subsetnames + '.csv'
                    outfile = open(outfilename, 'wb')
                    df.to_csv(outfilename, index = False, header = True, sep = ',', enco
    ding = 'utf-8')
                    outfile.close()
                else:
                    self.AAC_enc = df
                    return self.AAC_enc

            def dipept_encoding(self,exportpath):
                seq = self.datasets['sequence']
                dipept_list = dipeptide_comp(seq,self.alphabet)
                columns = []
                for i in range(0,len(self.alphabet)):
                    for j in range(0,len(self.alphabet)):
                        dipeptide = self.alphabet[i] + self.alphabet[j]
                        columns += [dipeptide]
                col_names = list(self.datasets.columns.values)
                rows = [k for k in range(0,len(seq))]
                datacells = np.zeros((len(seq),len(dipept_list[0])))
                for k in range(0,len(seq)):
                    for j in range(0,len(dipept_list[0])):
                        datacells[k,j] = dipept_list[k][j]
                df = pd.DataFrame(datacells,index=rows,columns=columns)
                df.insert(0,col_names[1],self.datasets[col_names[1]])
                if exportpath!='':
                    outfilename = exportpath + 'dipeptide_comp ' + self.subsetnames + '.
    csv'
                    outfile = open(outfilename, 'wb')
                    df.to_csv(outfilename, index = False, header = True, sep = ',', enco
    ding = 'utf-8')
                    outfile.close()
                else:
                    self.dipept_enc = df
                    return self.dipept_enc

            def digit_encoding(self,exportpath,AAindex_loc,PCprop):
                seq = self.datasets['sequence']
                PC_list = PC_prop(seq,self.alphabet,AAindex_loc,PCprop)
                digit_list = digitalize(PC_list)
                columns = ['p'+str(k) for k in range(0,len(digit_list[0]))]
                col_names = list(self.datasets.columns.values)
                rows = [k for k in range(0,len(seq))]
```

```
                datacells = np.zeros((len(seq),len(digit_list[0])))
                for k in range(0,len(seq)):
                    for j in range(0,len(digit_list[0])):
                        datacells[k,j] = digit_list[k][j]
                df = pd.DataFrame(datacells,index=rows,columns=columns)
                df.insert(0,col_names[1],self.datasets[col_names[1]])
                if exportpath!='':
                    outfilename = exportpath + 'digitalization ' + PCprop + ' ' + self.subsetnames + '.csv'
                    outfile = open(outfilename, 'wb')
                    df.to_csv(outfilename, index = False, header = True, sep = ',', encoding = 'utf-8')
                    outfile.close()
                else:
                    return df


            def fasta_conv(self,exportpath):
                df = self.datasets
                fasta_list = []
                prob_type = self.datasets.columns[1]
                if prob_type == 'class':
                    subsettypes = set(self.datasets['class'])
                    subsettypes = list(subsettypes)
                    for subset in subsettypes:
                        fasta_list = []
                        for i in range(0,len(df['sequence'])):
                            if df['class'].iloc[i]==subset:
                                fasta_list += ['>']
                                fasta_list += [df['sequence'].iloc[i]]
                        rows = [i for i in range(0,len(fasta_list))]
                        fasta_df = pd.DataFrame(fasta_list,index=rows,columns=['sequence'])
                        outfilename = exportpath + 'fasta ' + str(subset) + ' ' + self.subsetnames + '.fasta'
                        with open(outfilename, 'w') as outfile:
                            for row in fasta_df['sequence']:
                                outfile.write("".join(row)+'\n')
                        outfile.close()
                if prob_type == 'response':
                    for i in range(0,len(df['sequence'])):
                        fasta_list += ['>']
                        fasta_list += [df['sequence'].iloc[i]]
                    rows = [i for i in range(0,len(fasta_list))]
                    fasta_df = pd.DataFrame(fasta_list,index=rows,columns=['sequence'])

                    outfilename = exportpath + 'fasta ' + self.subsetnames + '.fasta'
                    with open(outfilename, 'w') as outfile:
                        for row in fasta_df['sequence']:
                            outfile.write("".join(row)+'\n')
                    outfile.close()

        # El presente script puede ejecutarse por sí solo utilizando las funciones y métodos definidos.
        # Input: Archivo que se desea codificar, debe ser un .csv con 3 columnas: sequence, class o response, is_outlier.
        # Output: Archivo codificado según el método especificado, en formato .csv. Contiene class o response en la primera columna.

        ## Instrucciones de uso:
        # 1. Declarar el archivo que se desea codificar
        # 2. Declarar la carpeta donde se desea guardar el archivo codificado
```

```
294.       # 3. En el caso de la digitalización, declarar el archivo que corresponde al AAi
    ndex, en .csv.
295.       # 4. Declarar el código de la propiedad fisicoquímica que se pretende emplear.
296.       # 5. Definir el conjunto de datos mediante una instancia de la clase "encoding"

297.       # 6. Emplear el método "remove_outliers()" para eliminar los outliers
298.       # 7. Utilizar los distintos métodos de codificación
```

Modelos predictivos

```
1.  %tensorflow_version 2.x
2.  import tensorflow as tf
3.
4.  !pip install -q sklearn
5.  !pip install bayesian-optimization
6.  !pip install PyDrive
7.  from pydrive.auth import GoogleAuth
8.  from pydrive.drive import GoogleDrive
9.  from google.colab import auth
10. from oauth2client.client import GoogleCredentials
11. import os
12. import sys
13. import numpy as np
14. import pandas as pd
15. import matplotlib.pyplot as plt
16. import math as mt
17. import time
18. import tensorflow.keras
19. import tensorflow.keras.initializers
20. from sklearn.model_selection import train_test_split, cross_validate
21. from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
22. from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
23. from sklearn.svm import SVC, SVR
24. from sklearn.metrics import classification_report, confusion_matrix
25. from sklearn.preprocessing import LabelEncoder
26. from tensorflow.keras.losses import sparse_categorical_crossentropy
27. from tensorflow.keras.optimizers import Adam, Adamax, Adagrad, Adadelta
28. from sklearn.model_selection import KFold
29. from tensorflow.keras.layers import LeakyReLU,PReLU, ELU
30. from tensorflow.keras.callbacks import EarlyStopping
31. from tensorflow.keras.models import Sequential, Model
32. from tensorflow.keras.layers import Dense, Activation, Dropout, InputLayer, BatchNormal
    ization, concatenate, Input, Conv1D, MaxPooling1D, Flatten, LSTM, Bidirectional, TimeDi
    stributed, Reshape, GlobalMaxPooling1D, GlobalMaxPool1D, AvgPool1D
33. from tensorflow.keras import regularizers
34. import time
35. import warnings
36.
37. documents_id = '1Z4mHE2N8Jy7Zjd4jZ973HtPHu8-JhhlQ'
38. documents_name = 'google_drive_documents_ids.csv'
39.
40. def r_square(y_true, y_pred):
41.   from keras import backend as K
42.   SS_res = K.sum(K.square(y_pred - y_true))
43.   SS_tot = K.sum(K.square(y_true - K.mean(y_true)))
44.   R2 = 1 - SS_res/(SS_tot + K.epsilon())
45.   return R2
46.
47. from keras import backend as K
48.
49. def recall_m(y_true, y_pred):
50.     true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
```

```
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    recall = true_positives / (possible_positives + K.epsilon())
    return recall

def precision_m(y_true, y_pred):
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
    return precision

def f1_m(y_true, y_pred):
    precision = precision_m(y_true, y_pred)
    recall = recall_m(y_true, y_pred)
    return 2*((precision*recall)/(precision+recall+K.epsilon()))

def matthews_correlation(y_true, y_pred):
    y_pred_pos = K.round(K.clip(y_pred, 0, 1))
    y_pred_neg = 1 - y_pred_pos

    y_pos = K.round(K.clip(y_true, 0, 1))
    y_neg = 1 - y_pos

    tp = K.sum(y_pos * y_pred_pos)
    tn = K.sum(y_neg * y_pred_neg)

    fp = K.sum(y_neg * y_pred_pos)
    fn = K.sum(y_pos * y_pred_neg)

    numerator = (tp * tn - fp * fn)
    denominator = K.sqrt((tp + fp) * (tp + fn) * (tn + fp) * (tn + fn))

    return numerator / (denominator + K.epsilon())

def get_datasets(set_name,encoding,df):
  ids=[]
  filenames=[]
  PCprops = ['PRAM900102','PRAM900103','COSI940101','HOPT810101','JOND750101','RADA880106','GRAR740103','FASG760101']
  if encoding=='onehot':
    ids = [df[df['Unnamed: 0']==set_name]['One_hot'].iloc[0]]
    filenames = ['One_hot ' + set_name + '.csv']
  elif encoding=='ordinal':
    ids = [df[df['Unnamed: 0']==set_name]['Ordinal'].iloc[0]]
    filenames = ['Ordinal ' + set_name + '.csv']
  elif encoding=='AAC':
    ids = [df[df['Unnamed: 0']==set_name]['AAC'].iloc[0]]
    filenames = ['AAC ' + set_name + '.csv']
  elif encoding=='dipeptide_comp':
    ids = [df[df['Unnamed: 0']==set_name]['dipeptide_comp'].iloc[0]]
    filenames = ['dipeptide_comp ' + set_name + '.csv']
        elif encoding=='all digitalizations':
          for prop in PCprops:
            ids += [df[df['Unnamed: 0']==set_name]['digitalization ' + prop].iloc[0]]
            filenames += ['digitalization ' + prop + ' ' + set_name + '.csv']
        elif PCprops.count(encoding)==1:
          ids = [df[df['Unnamed: 0']==set_name]['digitalization ' + encoding].iloc[0]]
          filenames = ['digitalization ' + encoding + ' ' + set_name + '.csv']
        for i in range(0,len(ids)):
          documents_file = drive.CreateFile({'id':ids[i]})
          documents_file.GetContentFile(filenames[i])
        return ids, filenames
```

```
111.
112.
113.       def read_dataframes(filenames):
114.         dfs = []
115.         for filename in filenames:
116.           df = pd.read_csv(filename)
117.           dfs += [df]
118.         return dfs
119.
120.       def fftpredict(dataframe, modeltype, problemtype, kfold_CV=True, dropout=0.2, lr
    =1e-4, loss_class='sparse_categorical_crossentropy',
121.                      loss_reg='mse', stopmetric='val_loss', patience=200, epochs=1000,
     batch_size=32,
122.                      optimizer=Adam, metrics=['accuracy'], min_delta=1e-3):
123.
124.         df = dataframe
125.         if problemtype=='regression':
126.             df = df[df['response'].notna()]
127.         df = df.sample(frac=1).reset_index(drop=True)
128.
129.         verbosity = 1
130.         columns = df.columns.tolist()
131.         samples = len(df[columns[1]])
132.
133.         if df.columns[0]=='class':
134.           no_outputs = len(df['class'].unique())
135.         elif df.columns[0]=='response':
136.           no_outputs = 1
137.
138.         if problemtype=='multiclass':
139.           LE = LabelEncoder()
140.           numerics=LE.fit(df['class'].tolist())
141.           transformed = LE.transform(df['class'].tolist())
142.           df['class'] = transformed
143.
144.         # Split training and external validation set, establish number of k-
    folds for internal validation
145.         if samples > 100 and samples <= 500:
146.           dtrain, deval = train_test_split(df, test_size=0.3)
147.           num_folds = 10
148.         elif samples > 500:
149.           dtrain, deval = train_test_split(df, test_size=0.2)
150.           num_folds = 5
151.         elif samples <= 100:
152.           dtrain = df
153.           deval = []
154.           num_folds = samples
155.
156.         # Determine shape of the data
157.         input_shape = (len(columns)-1, )
158.
159.         # Define per-fold score containers
160.         acc_per_fold = []
161.         loss_per_fold = []
162.
163.         # Define the K-fold Cross Validator
164.         kfold = KFold(n_splits=num_folds, shuffle=True)
165.
166.         # Seperate inputs and target values
167.         target_train = dtrain.pop(columns[0])
168.         input_train = dtrain.astype('float64')/dtrain.max().max()
169.         target_eval = deval.pop(columns[0])
170.         input_eval = deval.astype('float64')/dtrain.max().max()
```

```
171.
172.
173.         def generate_model(dropout,modeltype,problemtype):
174.           if modeltype=='ANN':
175.             inp = Input(shape=input_shape)
176.             model = Dense(256, activation='relu')(inp)
177.             model = Dropout(dropout)(model)
178.             model = Dense(128, activation='relu')(model)
179.             model = Dense(16, activation='relu')(model)
180.             model = Dropout(dropout)(model)
181.           elif modeltype=='LSTM':
182.             inp = Input(shape=(dtrain.shape[1],1))
183.             model = Bidirectional(LSTM(64, return_sequences =True, dropout=dropout))(i
    np)
184.             model = Dense(16, activation='relu')(model)
185.             model = Dropout(dropout)(model)
186.           elif modeltype=='CNN':
187.             inp = Input(shape=(dtrain.shape[1],1))
188.             model = Conv1D(filters=256, kernel_size=3, activation='relu')(inp)
189.             model = MaxPooling1D(pool_size=2)(model)
190.             model = Dropout(dropout)(model)
191.             model = Conv1D(filters=128, kernel_size=3, activation='relu')(model)
192.             model = MaxPooling1D(pool_size=2)(model)
193.             model = Dropout(dropout)(model)
194.             model = Flatten()(model)
195.             model = Dense(16, activation='relu')(model)
196.           if problemtype=='binary_class':
197.             model = Dense(2)(model)
198.             out = Activation('softmax')(model)
199.           elif problemtype=='multiclass':
200.             model = Dense(6)(model)
201.             out = Activation('softmax')(model)
202.           elif problemtype=='regression':
203.             out = Dense(1)(model)
204.           final = Model(inputs=[inp], outputs=[out])
205.           return final
206.
207.         def evaluate_complete_model(modeltype, problemtype, kfold_CV, dropout, lr,
208.                                     loss_class, loss_reg, stopmetric, patience, epochs
    , batch_size, optimizer, metrics, min_delta):
209.           batch_size = batch_size
210.           epochs_needed = []
211.           num = 0
212.           acc_per_fold = [] #acc or R2
213.           loss_per_fold = []
214.           start_time = time.time()
215.
216.           if kfold_CV==True:
217.             # K-fold Cross Validation model procedure
218.             fold_no = 1
219.             for train, internal_test in kfold.split(input_train, target_train):
220.               num+=1
221.
222.               # Model architect and hyperparameters establishment
223.               model = generate_model(dropout,modeltype,problemtype)
224.               if problemtype=='binary_class' or problemtype=='multiclass':
225.                 loss=loss_class
226.               elif problemtype=='regression':
227.                 loss=loss_reg
228.               if stopmetric=='val_r_square':
229.                 mode = 'max'
230.               else:
231.                 mode = 'auto'
```

```
232.                 model.compile(loss=loss, optimizer=optimizer(lr=lr), metrics=metrics)
233.                 monitor = EarlyStopping(monitor=stopmetric, min_delta=min_delta,
234.                 patience=patience, verbose=0, mode=mode, restore_best_weights=True)
235.
236.                 # Define training and validation subset within specific kfold set
237.                 x_train = input_train.iloc[train]
238.                 y_train = target_train.iloc[train]
239.                 x_test = input_train.iloc[internal_test]
240.                 y_test = target_train.iloc[internal_test]
241.
242.                 # Generate a print
243.                 print('------------------------------------------------------------------------
    -------')
244.                 print(f'Training for fold {fold_no} ...')
245.
246.                 # Train on the kfold sample
247.                 model.fit(x_train,y_train,validation_data=(x_test,y_test), callbacks=[mo
    nitor], epochs=epochs, batch_size=batch_size, verbose=verbosity)
248.                 epochs_end = monitor.stopped_epoch
249.                 epochs_needed.append(epochs_end)
250.
251.                 # Generate generalization metrics
252.                 scores = model.evaluate(x_test, y_test, verbose=0)
253.                 if problemtype=='regression':
254.                   print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[
    0]}; {model.metrics_names[1]} of {scores[1]}')
255.                   C = 1
256.                 else:
257.                   print(f'Score for fold {fold_no}: {model.metrics_names[0]} of {scores[
    0]}; {model.metrics_names[1]} of {scores[1]*100}%')
258.                   C = 100
259.                 acc_per_fold.append(scores[1] * C)
260.                 loss_per_fold.append(scores[0])
261.
262.                 # Increase fold number
263.                 fold_no = fold_no + 1
264.
265.             time_took = time.time() - start_time
266.             print(f'time took for kfold_CV: {time_took}')
267.
268.             # Provide average scores
269.             m1 = np.mean(loss_per_fold)
270.             m2 = np.mean(acc_per_fold)
271.             m3 = np.mean(epochs_needed)
272.             m1dev = np.std(loss_per_fold)
273.             m2dev = np.std(acc_per_fold)
274.             m3dev = np.std(epochs_needed)
275.             scores2 = model.evaluate(input_eval,target_eval)
276.             if problemtype=='regression':
277.               print(f'loss: {m1} +- {m1dev}, error: {m2} +- {m2dev}, epochs: {m3} +- {
    m3dev}, test error: {scores2[1]}')
278.             else:
279.               print(f'loss: {m1} +- {m1dev}, accuracy: {m2} +- {m2dev}, epochs: {m3} +
    - {m3dev}, test acc: {scores2[1]*100}')
280.             tensorflow.keras.backend.clear_session()
281.             stat = [m1,m1dev,m2,m2dev,m3,m3dev]
282.             return stat
283.
284.           elif kfold_CV==False:
285.             start_time = time.time()
286.             model = generate_model(dropout,modeltype,problemtype)
287.             if problemtype=='binary_class' or problemtype=='multiclass':
288.               loss=loss_class
```

```
            elif problemtype=='regression':
              loss=loss_reg
            model.compile(loss=loss, optimizer=optimizer(lr=lr), metrics=metrics)
            monitor = EarlyStopping(monitor=stopmetric, min_delta=min_delta,
                                    patience=patience, verbose=1, mode='auto', restore
_best_weights=True)
            x_train = input_train
            y_train = target_train
            x_test = input_eval
            y_test = target_eval

            history = model.fit(x=x_train,y=y_train,validation_data=(x_test,y_test), c
allbacks=[monitor], epochs=epochs, batch_size=batch_size, verbose=verbosity)

            time_took = time.time() - start_time
            print(f'time took for training on whole set: {time_took}')

        print(evaluate_complete_model(modeltype, problemtype, kfold_CV, dropout, lr,

                                      loss_class, loss_reg, stopmetric, patience, epochs, ba
tch_size, optimizer, metrics, min_delta))


      def knn_mod(dataframe, problemtype, kfold_CV=True, kmin=1, kmax=20, verb=1, K=1,
 target_names=['clase 0','clase 1']):
        df = dataframe
        if problemtype=='regression':
            df = df[df['response'].notna()]
        df = df.sample(frac=1).reset_index(drop=True)
        #samples = len(df['p0'])

        verbosity = 1
        columns = df.columns.tolist()
        samples = len(df[columns[1]])

        if problemtype=='multiclass':
          LE = LabelEncoder()
          numerics=LE.fit(df['class'].tolist())
          transformed = LE.transform(df['class'].tolist())
          df['class'] = transformed

        if problemtype=='binary_class' and isinstance(df['class'].iloc[0],str):
          LE = LabelEncoder()
          numerics=LE.fit(df['class'].tolist())
          transformed = LE.transform(df['class'].tolist())
          df['class'] = transformed

        # Split training and external validation set, establish number of k-
folds for internal validation
        if samples > 100 and samples <= 500:
          dtrain, deval = train_test_split(df, test_size=0.3)
          num_folds = 10
        elif samples > 500:
          dtrain, deval = train_test_split(df, test_size=0.2)
          num_folds = 5
        elif samples <= 100:
          dtrain = df
          deval = []
          num_folds = samples

        # Determine shape of the data
        input_shape = (len(columns)-1, )

```

```
        # Define per-fold score containers
        acc_per_fold = []
        loss_per_fold = []

        # Define the K-fold Cross Validator
        kfold = KFold(n_splits=num_folds, shuffle=True)

        # Seperate inputs and target values
        target_train = dtrain.pop(columns[0])
        input_train = dtrain.astype('float64')/dtrain.max().max()
        target_test = deval.pop(columns[0])
        input_test = deval.astype('float64')/dtrain.max().max()

        if kfold_CV==True:
          k_range = range(kmin, kmax+1)
          # K-fold Cross Validation model procedure
          if problemtype=='binary_class' or problemtype=='multiclass':
            k_scores = [[],[],[],[],[],[],[],[]]
            for k in k_range:
              knn = KNeighborsClassifier(n_neighbors=k)
              if problemtype=='binary_class':
                p = 'precision'
                r = 'recall'
                f1 = 'f1'
              elif problemtype=='multiclass':
                f1 = 'f1_weighted'
                p = 'precision_weighted'
                r = 'recall_weighted'
              scores = cross_validate(knn, input_train, target_train, cv=num_folds, sc
oring=('accuracy', p, r, f1), verbose=verb)
              k_scores[0] += [scores['test_accuracy'].mean()]
              k_scores[1] += [scores['test_accuracy'].std()]
              k_scores[2] += [scores['test_'+p].mean()]
              k_scores[3] += [scores['test_'+p].std()]
              k_scores[4] += [scores['test_'+r].mean()]
              k_scores[5] += [scores['test_'+r].std()]
              k_scores[6] += [scores['test_'+f1].mean()]
              k_scores[7] += [scores['test_'+f1].std()]
            score_dict = ['accuracy','precision','recall',f1]
            return k_scores
          elif problemtype=='regression':
            k_scores = [[],[],[],[]]
            for k in k_range:
              knn = KNeighborsRegressor(n_neighbors=k)
              scores = cross_validate(knn, input_train, target_train, cv=num_folds, sc
oring=('neg_mean_squared_error', 'r2'), verbose=verb)
              k_scores[0] += [scores['test_neg_mean_squared_error'].mean()]
              k_scores[1] += [scores['test_neg_mean_squared_error'].std()]
              k_scores[2] += [scores['test_r2'].mean()]
              k_scores[3] += [scores['test_r2'].std()]
            return k_scores

        elif kfold_CV==False:
          target_test = np.array(target_test.tolist())
          if problemtype=='binary_class' or problemtype=='multiclass':
            if problemtype=='multiclass':
              c='balanced'
            else:
              c=None
            knn = KNeighborsClassifier(n_neighbors=K)
            knn.fit(input_train,target_train)
            y_pred = knn.predict(input_test)
            score = accuracy_score(target_test,y_pred)
```

```
            elif problemtype=='regression':
              knn = KNeighborsRegressor(n_neighbors=K)
              knn.fit(input_train,target_train)
              y_pred = knn.predict(input_test)
              R2 = r2_score(target_test,y_pred)
              return R2


      def rf_mod(dataframe, problemtype, kfold_CV=True, ntree=200, bootstrap=False,
                 max_features='auto', min_samples_leaf=1, min_samples_split=2,verb=1,
    n_jobs=None, class_weight=None):
          df = dataframe
          if problemtype=='regression':
              df = df[df['response'].notna()]
          df = df.sample(frac=1).reset_index(drop=True)
          #samples = len(df['p0'])

          verbosity = 1
          columns = df.columns.tolist()
          samples = len(df[columns[1]])

          if problemtype=='multiclass':
            LE = LabelEncoder()
            numerics=LE.fit(df['class'].tolist())
            transformed = LE.transform(df['class'].tolist())
            df['class'] = transformed

          # Split training and external validation set, establish number of k-
    folds for internal validation
          if samples > 100 and samples <= 500:
            dtrain, deval = train_test_split(df, test_size=0.3)
            num_folds = 10
          elif samples > 500:
            dtrain, deval = train_test_split(df, test_size=0.2)
            num_folds = 5
          elif samples <= 100:
            dtrain = df
            deval = []
            num_folds = samples

          # Determine shape of the data
          input_shape = (len(columns)-1, )

          # Define per-fold score containers
          acc_per_fold = []
          loss_per_fold = []

          # Define the K-fold Cross Validator
          kfold = KFold(n_splits=num_folds, shuffle=True)

          # Seperate inputs and target values
          target_train = dtrain.pop(columns[0])
          input_train = dtrain.astype('float64')/dtrain.max().max()
          target_test = deval.pop(columns[0])
          input_test = deval.astype('float64')/dtrain.max().max()

          if kfold_CV==True:
            # K-fold Cross Validation model procedure
            if problemtype=='binary_class' or problemtype=='multiclass':
              rf_scores = [[],[],[],[],[],[],[],[]]
              rf = RandomForestClassifier(n_estimators=ntree, min_samples_split=min_samp
    les_split,
```

```
                                            min_samples_leaf=min_samples_leaf, bootstrap=bootstrap, max_features=max_features, class_weight=class_weight)
              if problemtype=='binary_class':
                p = 'precision'
                r = 'recall'
                f1 = 'f1'
              elif problemtype=='multiclass':
                f1 = 'f1_weighted'
                p = 'precision_weighted'
                r = 'recall_weighted'
              scores = cross_validate(rf, input_train, target_train, cv=num_folds, scoring=('accuracy', p, r, f1),verbose=verb, n_jobs=n_jobs)
              rf_scores[0] += [scores['test_accuracy'].mean()]
              rf_scores[1] += [scores['test_accuracy'].std()]
              rf_scores[2] += [scores['test_'+p].mean()]
              rf_scores[3] += [scores['test_'+p].std()]
              rf_scores[4] += [scores['test_'+r].mean()]
              rf_scores[5] += [scores['test_'+r].std()]
              rf_scores[6] += [scores['test_'+f1].mean()]
              rf_scores[7] += [scores['test_'+f1].std()]
              return rf_scores
            elif problemtype=='regression':
              rf_scores = [[],[],[],[]]
              rf = RandomForestRegressor(n_estimators=ntree, min_samples_split=min_samples_split,
                                            min_samples_leaf=min_samples_leaf, bootstrap=bootstrap, max_features=max_features)
              scores = cross_validate(rf, input_train, target_train, cv=num_folds, scoring=('neg_mean_squared_error', 'r2'), verbose=verb, n_jobs=n_jobs)
              rf_scores[0] += [scores['test_neg_mean_squared_error'].mean()]
              rf_scores[1] += [scores['test_neg_mean_squared_error'].std()]
              rf_scores[2] += [scores['test_r2'].mean()]
              rf_scores[3] += [scores['test_r2'].std()]
              return rf_scores

          elif kfold_CV==False:
            target_test = np.array(target_test.tolist())
            if problemtype=='binary_class' or problemtype=='multiclass':
              if problemtype=='multiclass':
                c='balanced'
              else:
                c=None
              rf = RandomForestClassifier(n_estimators=ntree, min_samples_split=min_samples_split,
                                            min_samples_leaf=min_samples_leaf, bootstrap=bootstrap, max_features=max_features)
              rf.fit(input_train,target_train)
              y_pred = rf.predict(input_test)
              print('y_pred:')
              print(y_pred)
              print('target_test:')
              print(target_test)
              score = accuracy_score(target_test,y_pred)
              return score
            elif problemtype=='regression':
              rf = RandomForestRegressor(n_estimators=ntree, min_samples_split=min_samples_split,
                                            min_samples_leaf=min_samples_leaf, bootstrap=bootstrap, max_features=max_features)
              rf.fit(input_train,target_train)
              y_pred = rf.predict(input_test)
              R2 = r2_score(target_test,y_pred)
              return R2
```

```
521.
522.
523.
524.       def svm_mod(dataframe, problemtype, kfold_CV=True, C=1, kernel='rbf', gamma='sca
    le', verb=1, n_jobs=None):
525.         df = dataframe
526.         if problemtype=='regression':
527.             df = df[df['response'].notna()]
528.         df = df.sample(frac=1).reset_index(drop=True)
529.         #samples = len(df['p0'])
530.
531.         verbosity = 1
532.         columns = df.columns.tolist()
533.         samples = len(df[columns[1]])
534.
535.         if problemtype=='multiclass':
536.           LE = LabelEncoder()
537.           numerics=LE.fit(df['class'].tolist())
538.           transformed = LE.transform(df['class'].tolist())
539.           df['class'] = transformed
540.
541.         # Split training and external validation set, establish number of k-
    folds for internal validation
542.         if samples > 100 and samples <= 500:
543.           dtrain, deval = train_test_split(df, test_size=0.3)
544.           num_folds = 10
545.         elif samples > 500:
546.           dtrain, deval = train_test_split(df, test_size=0.2)
547.           num_folds = 5
548.         elif samples <= 100:
549.           dtrain = df
550.           deval = []
551.           num_folds = samples
552.
553.         # Determine shape of the data
554.         input_shape = (len(columns)-1, )
555.
556.         # Define per-fold score containers
557.         acc_per_fold = []
558.         loss_per_fold = []
559.
560.         # Define the K-fold Cross Validator
561.         kfold = KFold(n_splits=num_folds, shuffle=True)
562.
563.         # Seperate inputs and target values
564.         target_train = dtrain.pop(columns[0])
565.         input_train = dtrain.astype('float64')/dtrain.max().max()
566.         target_test = deval.pop(columns[0])
567.         input_test = deval.astype('float64')/dtrain.max().max()
568.
569.         if kfold_CV==True:
570.           if problemtype=='binary_class' or problemtype=='multiclass':
571.             sv_scores = [[],[],[],[],[],[],[],[]]
572.             sv = SVC(C=C, kernel=kernel, gamma=gamma, verbose=verb, class_weight='bala
    nced')
573.             if problemtype=='binary_class':
574.               p = 'precision'
575.               r = 'recall'
576.               f1 = 'f1'
577.             elif problemtype=='multiclass':
578.               f1 = 'f1_weighted'
579.               p = 'precision_weighted'
580.               r = 'recall_weighted'
```

```
            scores = cross_validate(sv, input_train, target_train, cv=num_folds, scoring=('accuracy', p, r, f1),verbose=verb, n_jobs=n_jobs)
            sv_scores[0] += [scores['test_accuracy'].mean()]
            sv_scores[1] += [scores['test_accuracy'].std()]
            sv_scores[2] += [scores['test_'+p].mean()]
            sv_scores[3] += [scores['test_'+p].std()]
            sv_scores[4] += [scores['test_'+r].mean()]
            sv_scores[5] += [scores['test_'+r].std()]
            sv_scores[6] += [scores['test_'+f1].mean()]
            sv_scores[7] += [scores['test_'+f1].std()]
            return sv_scores
          elif problemtype=='regression':
            sv_scores = [[],[],[],[]]
            sv = SVR(C=C, kernel=kernel, gamma=gamma, verbose=verb)
            scores = cross_validate(sv, input_train, target_train, cv=num_folds, scoring=('neg_mean_squared_error', 'r2'), verbose=verb, n_jobs=n_jobs)
            sv_scores[0] += [scores['test_neg_mean_squared_error'].mean()]
            sv_scores[1] += [scores['test_neg_mean_squared_error'].std()]
            sv_scores[2] += [scores['test_r2'].mean()]
            sv_scores[3] += [scores['test_r2'].std()]
            return sv_scores

        elif kfold_CV==False:
          target_test = np.array(target_test.tolist())
          if problemtype=='binary_class' or problemtype=='multiclass':
            if problemtype=='multiclass':
              c='balanced'
            else:
              c=None
            sv = SVC(C=C, kernel=kernel, gamma=gamma, verbose=verb, class_weight=c)
            sv.fit(input_train,target_train)
            y_pred = sv.predict(input_test)
            score = accuracy_score(target_test,y_pred)
            return score
          elif problemtype=='regression':
            sv = SVR(C=C, kernel=kernel, gamma=gamma, verbose=verb)
            sv.fit(input_train,target_train)
            y_pred = sv.predict(input_test)
            R2 = r2_score(target_test,y_pred)
            return R2
```