Máster en Bioinformática y Bioestadística - UOC / UB Febrero 2022 - Semestre de primavera

Trabajo de Final de Máster - Fase 1: Desarrollo de Trabajo

Alumno: Juan Vigueras Díaz **Tutor**: Jorge Valencia Delgadillo

---

A continuación se exponen las tareas que se han completado durante las semanas correspondientes a la fase 1. Cada modificación realizada respecto a lo expuesto en el plan de trabajo se indica mediante la anteposición de la palabra "**Modificación**" antes de la tarea correspondiente. Cualquier otro comentario que sea necesario resaltar se indicará mediante "**Nota Importante**". Las referencias a páginas web se marcan mediante letras y las referencias a bibliografía se marcan mediante números.

• Se ha creado en la plataforma Google Drive un directorio llamado "DEV" para el desarrollo en línea del proyecto en el que se incluirán los conjuntos de datos, el código y los resultados u *outputs* del estudio.

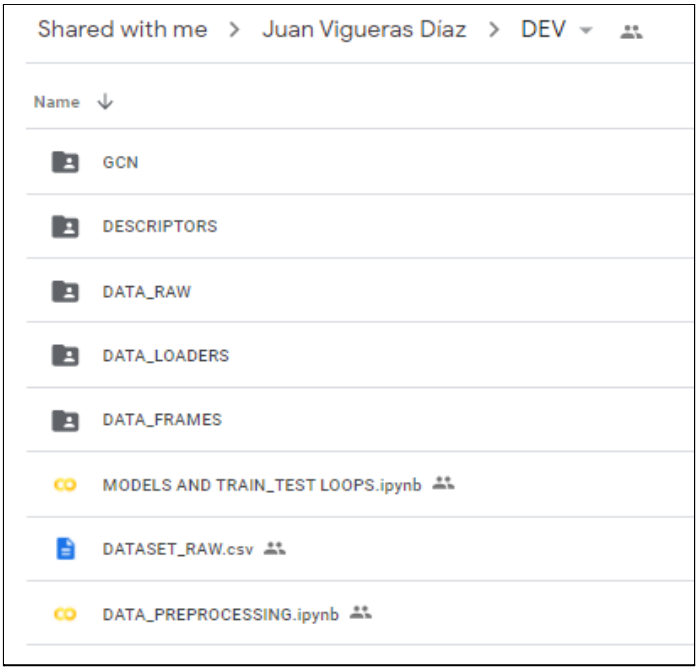

Imagen 1. Sistema de directorios de Google Drive para el desarrollo del proyecto en línea con las rutas y archivos creados hasta el momento actual

PREPROCESAMIENTO DE DATOS

• Se han descargado de la base de datos ChEMBL [a] los archivos en formato CSV de las bacterias y proteínas de estudio y se ha incluido en el directorio de desarrollo "DEV" un subdirectorio llamado "DATA_RAW" para los archivos de datos en crudo. **Modificación:** Se han descargado todas las columnas proporcionadas en la base de datos para cada *target*, ya que no se puede seleccionar la descarga de únicamente un conjunto de variables. Dentro del pre-procesamiento de datos se seleccionarán los campos de interés de cada archivo.

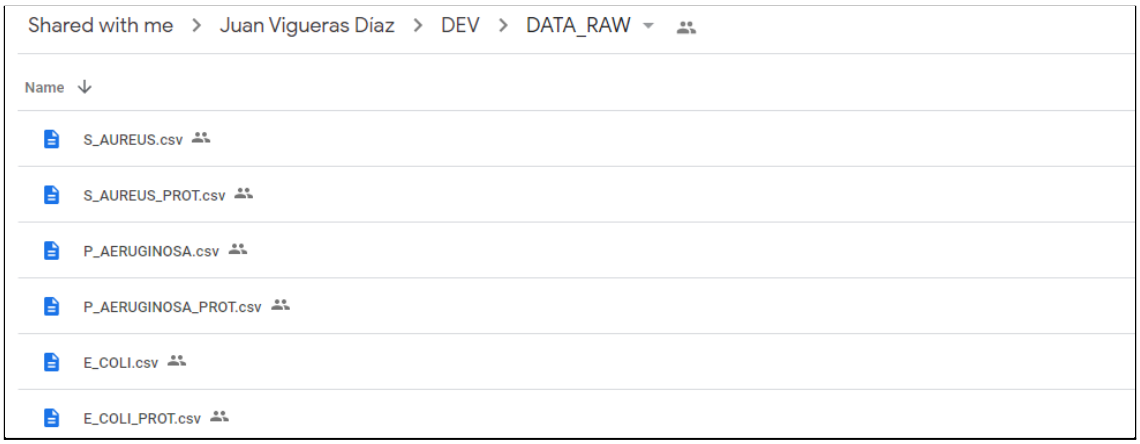

Imagen 2. Directorio 'DATA_RAW' con los archivos CSV directamente descargados de forma manual desde la web ChEMBL

En la entrega anterior ya se especificaron los identificadores de cada fuente de datos. Con el presente documento se adjunta una copia de cada archivo descargado.

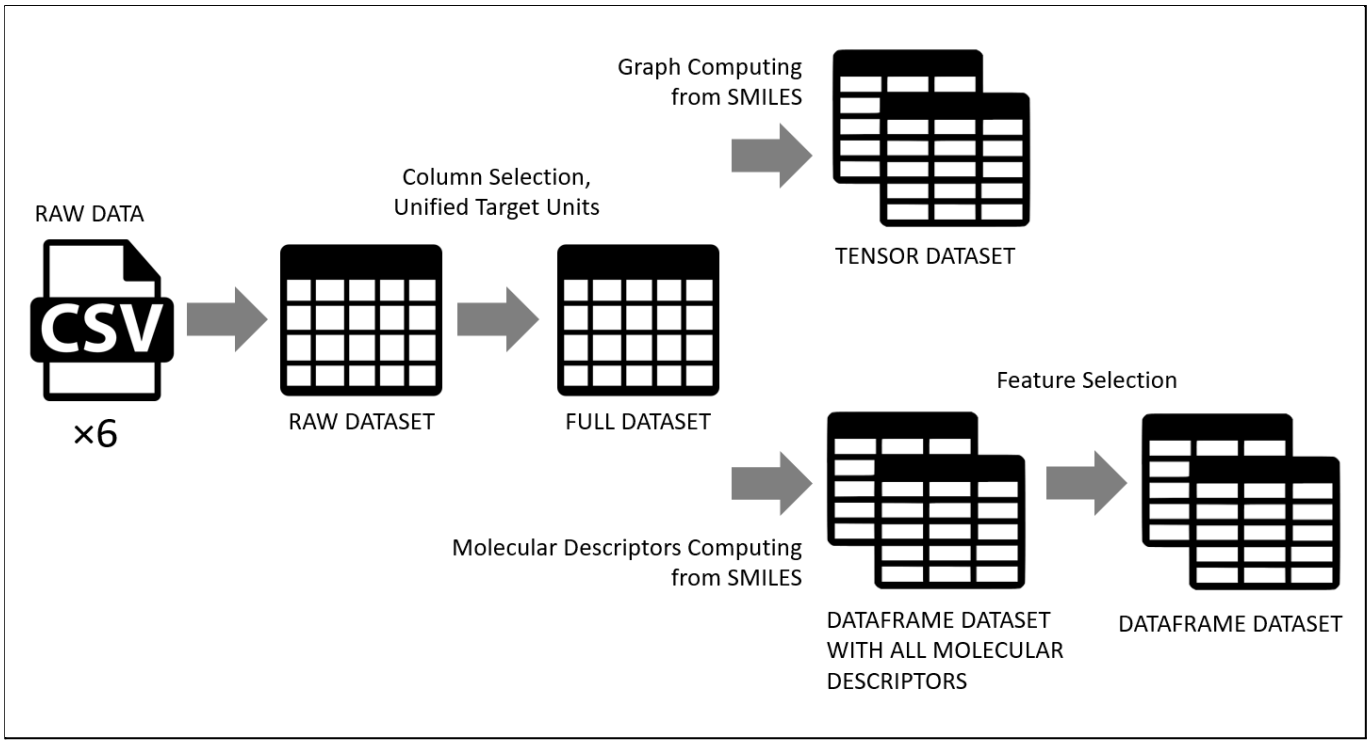

Imagen 3. Diagrama de desarrollo del preprocesamiento de datos para crear un 'dataset' conformado por grafos (tensores) y otro conformado por descriptores moleculares (dataframe)

• Se ha creado en el directorio de desarrollo "DEV" un *notebook* colaborativo con la herramienta Google Colab llamado "DATA_PREPROCESSING.ipynb" en el que se llevan a cabo las tareas indicadas en el diagrama de la Imagen 3. El desarrollo del código se ha realizado íntegramente con Python y el conjunto de librerías descargadas se puede consultar en el propio *notebook.* El enlace del *notebook* se muestra en la bibliografía [b] y se adjunta una copia descargada con el presente documento.

Los pasos detallados del diagrama son:

Paso 1. Lectura de los archivos de datos en crudo en formato CSV descargados desde la base ChEMBL y posterior unificación de todos los registros en una única tabla de origen.

Paso 2. Preprocesamiento de la tabla origen del paso anterior para obtener una tabla de datos en crudo:

2.1. Selección de las columnas de interés que se muestran en la Tabla 1. Se renombran respecto al encabezado de los archivos originales de la base de datos ChEMBL de la siguiente manera:

Columna original en ChEMBL	**Nombre Asignado**
Molecular Weight	WEIGHT
#RO5 Violations	RO5
Smiles	SMILES
Standard Type	TYPE
Standard Relation	RELATION
Standard Value	VALUE
Standard Units	UNITS

Tabla 1. Se muestran las columnas seleccionadas de los archivos de datos en crudo descargados desde ChEMBL; por un lado, el nombre en la tabla original y, por otro lado, los nuevos nombres asignados

Se eliminan aquellos registros que presentan al menos un valor nulo en los campos de interés.

A continuación se muestran en la Tabla 2 las descripciones de las columnas de esta tabla y, en algunos casos, una justificación del motivo de su selección.

Columna	**Descripción / Motivo de elección**	**Tipo de dato**
WEIGHT	Peso molecular dado en [g/mol]. Este campo se utilizará, por una parte, para unificar las unidades de los indicadores y, por otro lado, para más adelante realizar la comprobación del correcto cálculo de los descriptores moleculares para cada registro/molécula.	Coma flotante
RO5	**Modificación**: Número de violaciones según la regla de cinco de Lipinski, que permite evaluar cualitativamente la viabilidad de un compuesto químico como fármaco una vez es ingerido oralmente por humanos. No se incluye en el modelo y no se utiliza en esta fase del trabajo, pero se ha seleccionado ya que puede resultar ser una variable interesante de consultar en el futuro.	Entero
SMILES	Estructura de la molécula en formato SMILES.	Cadena de texto
TYPE	Indicador calculado sobre cada molécula/registro. En este caso y, tal y como ya se ha comentado en anteriores entregas, solamente hay dos valores posibles: MIC (concentración mínima inhibidora) y IC50 (mitad/50% de la concentración máxima inhibidora).	Cadena de texto
RELATION	Tipo de relación experimental en cuanto al valor del indicador. **Modificación**: Existen relaciones de desigualdad que no interesan en los modelos de predicción exacta del valor de bioactividad, con lo que finalmente se selecciona el tipo de relación 'igual' ("="). Esta consideración provoca que se pierdan una gran cantidad de registros, 52.233 del total de 220.283, lo cual supone casi un 24% . Se podría ampliar el estudio implementando modelos que traten de predecir el valor 'mínimo' o 'máximo' necesario del indicativo correspondiente para marcar bioactividad en la molécula correspondiente, lo cual queda fuera del alcance de este proyecto.	Cadena de texto
VALUE	Valor numérico del indicativo (MIC o IC50).	Coma flotante
UNITS	Unidades de medida del indicativo. En este caso, solo hay dos valores posibles: · nM - concentración nanomolar · microgramos/mililitro Cada registro utiliza indistintamente una magnitud u otra en función del experimento original desde donde se recupera la información.	Cadena de texto
TARGET	Columna añadida en la creación de la tabla, correspondiente al archivo origen de cada registro, es decir, el *target* de la molécula, indicando si se trata de cualquiera de las tres bacterias: ['E_COLI', 'P AERUGINOSA', 'S_AUREUS'] o de sus correspondientes proteínas - mismos nombres con la terminación '_PROT'. Como ya se ha mencionado, las proteínas seleccionadas de cada bacteria son las ya especificadas anteriormente en el plan de trabajo.	Cadena de texto

Tabla 2. Descripción y/o motivo de selección de los campos del conjunto de datos original, así como el tipo de dato correspondiente a cada columna

2.2. Unificación de las unidades de las variables de respuesta (IC50 y MIC) correspondientes a los indicadores de bioactividad. Algunos valores de indicadores vienen dados en unidades de microgramo partido mililitro, mientras que otros lo están en concentración nanomolar. Para unificar las unidades se escoge arbitrariamente la

concentración nanomolar, ya que, aunque la distribución de ambas no es similar (67.777 registros para microgramos partido mililitro contra 12.369 registros para concentración nanomolar), la bibliografía no indica motivos para la preferencia del uso de una sobre otra y, además, el mol es la unidad en el sistema internacional para la medición de cantidad de substancia. La relación entre estas unidades se puede calcular como:

$$MW = 'Molecular\ Weight',\ KD = 'Kilo\ Dalton'$$

$$\mu M = \left(\frac{\mu g}{mL}\right) / (MW[KD]) \quad (1)$$

$$(MW[KD]) = MW\left[\frac{g}{mol}\right] \cdot 10^{-3} \quad (2)$$

$$[nM] = 10^{-3}\,[\mu M]$$

Con lo que el valor de cada indicativo dado en microgramo partido mililitro se transforma en concentración nanomolar simplemente dividiendo el valor de dicho indicativo entre la masa molecular proporcionada por el *dataset*, tal y como se muestra en la Ecuación (3).

$$Valor\ Indicativo\ [nM] = \frac{Valor\ Indicativo\ [\mu g\ mL^{-1}]}{MW[g\ moll^{-1}]} \quad (3)$$

La tabla resultante con los campos de la Tabla 1 conjuntamente con esta nueva columna de indicativos con unidades unificadas en concentración nanomolar se guarda bajo el nombre de 'DATASET_RAW.csv' en el directorio 'DEV' y se adjunta una copia con el presente documento.

	index	WEIGHT	ROS	SMILES	TYPE	RELATION	VALUE	UNITS	TARGET	VALUE_NM
0	9857	740.80	2	Br.Br.CCCCC(CC)CN=c1ccn(CCCCCCCCCCCCn2ccc(=NCC...	MIC	'='	1.0	ug.mL-1	E_COLI	0.001350
1	8389	712.74	2	Br.Br.CCCCC(CC)CN=c1ccn(CCCCCCCCCCn2ccc(=NCC(C...	MIC	'='	0.4	ug.mL-1	E_COLI	0.000561
2	16103	698.72	2	Br.Br.CCCCC(CC)CN=c1ccn(CCCCCCCCCn2ccc(=NCC(CC...	MIC	'='	1.0	ug.mL-1	E_COLI	0.001431
3	3008	684.69	2	Br.Br.CCCCC(CC)CN=c1ccn(CCCCCCCCn2ccc(=NCC(CC)...	MIC	'='	1.0	ug.mL-1	E_COLI	0.001461
4	4416	670.66	2	Br.Br.CCCCC(CC)CN=c1ccn(CCCCCCCn2ccc(=NCC(CC)C...	MIC	'='	3.9	ug.mL-1	E_COLI	0.005815
...	...	...	...	...	...	...	...	...	...	...
80141	167935	234.31	0	O=c1[nH]cnc2scc(-c3cccs3)c12	IC50	'='	50000.0	nM	S_AUREUS_PROT	50000.000000
80142	167959	371.43	0	O=c1cc(-c2ccccc2)c2c(-c3cnc(-c4cccnc4)s3)n[nH]...	IC50	'='	1315.0	nM	S_AUREUS_PROT	1315.000000
80143	167981	295.33	0	O=c1ccc2c(-c3cnc(-c4cccnc4)s3)n[nH]c2[nH]1	IC50	'='	270.0	nM	S_AUREUS_PROT	270.000000
80144	167831	339.38	0	O=c1ccc2c(-c3cnc(-c4cccnc4)s3)n[nH]c2n1CCO	IC50	'='	564.0	nM	S_AUREUS_PROT	564.000000
80145	167903	454.33	0	O=c1nc(/C=C/c2ccc(/C=C/c3cc(C(F)(F)F)[nH]c(=O)...	IC50	'='	42210.0	nM	S_AUREUS_PROT	42210.000000

80146 rows × 10 columns

Imagen 4. Conjunto de datos importados conjuntamente con la columna 'TARGET' referenciando a la bacteria o proteína de estudio y la columna 'VALUE_NM' donde se unifican las unidades de la columna 'VALUE' a concentración nanomolar

Paso 3. Creación de un *dataset* de los descriptores moleculares calculados para cada molécula y de un *dataset* de los grafos correspondientes a cada molécula. Para procesar los datos desde la tabla obtenida en el paso anterior de manera que se puedan construir los conjuntos de datos finales, se definen las siguientes clases:

Clase	**Funcionalidad (atributos y métodos)**
Molecules()	Se construye mediante una lista de moléculas en formato SMILES. Devuelve los siguientes atributos: · smiles: Lista de moléculas en formato SMILES. · mols: Lista de moléculas en formato objeto de la librería RdKit, · molblocks: Lista de tablas MOL en formato sub-lista. Cada tabla MOL es una lista con cada fila como elemento en formato cadena de texto. · num_atoms: Lista con el número de átomos totales para cada molécula. **Nota Importante**: Este dato se extrae de la tabla MOL anteriormente mencionada y, en algunos casos, ésta contiene errores de cálculo que descartan la molécula para la generación del *output*. Cuando aparece un error, el valor de *num_atoms* se asocia a "0"

	y se filtra en un tratamiento posterior para no tener en cuenta el registro. · num_bounds: Lista con el número de enlaces totales para cada molécula. **Nota Importante**: Misma consideración en la gestión de errores mencionada en la nota del atributo *num_atoms*. Contiene los métodos: · unique_atoms(): Devuelve un diccionario con las claves correspondientes a los átomos únicos encontrados en toda la lista de cadenas moleculares y sus correspondientes valores de codificación etiquetada ponderada calculados. La ponderación se realiza en función de la frecuencia de cada átomo en el conjunto de datos completo, y se distribuyen los pesos en valores comprendidos entre 0 y 1. · compute_structures(): Devuelve dos listas. La primera contiene una sub-lista por cada molécula, que a su vez contiene una lista con las coordenadas bidimensionales [x,y] de cada átomo, el símbolo químico del átomo y su correspondiente valor de codificación ponderada. La segunda contiene una sub-lista para cada molécula, que a su vez contiene una lista con los emparejamientos de los enlaces [a, b] y el valor de codificación del tipo de enlace proporcionada directamente por la tabla MOL.
Descriptors()	Se construye mediante una lista de moléculas en formato SMILES. Devuelve los siguientes atributos: · smiles: Lista de moléculas en formato SMILES. · Molecules: Instancia de la clase Molecules() correspondiente. Contiene el método: · compute_descriptors(descriptors_list): Toma como parámetro la lista de descriptores moleculares que se desea calcular para cada molécula. Los descriptores han de estar indicados según la nomenclatura establecida por la librería RdKit. Devuelve una tabla en formato *dataframe* con un registro por molécula, con una columna por cada descriptor molecular calculado. **Nota Importante**: Como este método no requiere el uso de la tabla MOL para los cálculos, se obtiene un registro por cada molécula indicada sin que influya la gestión de errores mencionada en la clase Molecules().
Graphs()	Se construye mediante una lista de moléculas en formato SMILES. Devuelve los siguientes atributos: · smiles: Lista de moléculas en formato SMILES. · Molecules: Instancia de la clase Molecules() correspondiente. Contiene el método: · compute_graphs(norm): Devuelve dos listas. La primera contiene una sub-lista por cada molécula, que a su vez contiene una lista con los valores de codificación de cada átomo, una lista con las coordenadas de cada átomo y una lista con las parejas bidireccionales de cada enlace. La segunda contiene una lista con los índices numéricos de las posiciones de las moléculas en la lista proporcionada para instanciar la clase; y únicamente contiene los índices correspondientes a aquellas moléculas que han podido ser correctamente procesadas sin error. · plot_molecule(molecule, labeled, grid): Devuelve un gráfico representando la molécula indicada por el parámetro *molecule*. Este parámetro es un valor numérico correspondiente al índice numérico de la posición de la molécula en la lista

	proporcionada para instanciar la clase. Cada átomo se representa de un color distinto y se diferencia entre enlaces simples, dobles u otros. Otros parámetros opcionales son: *labeled*, si se desea un gráfico con ejes; *grid*, si se desea un gráfico con cuadrícula de fondo. Un ejemplo de *output* de esta función se muestra a continuación de esta tabla.
Descriptor_Dataset()	Se construye mediante una lista de moléculas en formato SMILES, una lista con el valor del indicador o *target* correspondiente a cada molécula y una lista con los descriptores moleculares a calcular (estos han de estar escritos según el nombre que aparezca en el registro del módulo Chem de la librería RdKit). Devuelve los siguientes atributos: · smiles: Lista de moléculas en formato SMILES. · target: Lista con los valores de *target* de cada molécula. · md_list: Lista con los descriptores moleculares a calcular. · Descriptors: Instancia de la clase Descriptors() correspondiente. Contiene el método: · create_dataset(): Devuelve tres objetos en formato *DataFrame* correspondientes a los subconjuntos de datos de *entrenamiento*, *validación* y *test* típicos en el marco de trabajo de modelos de aprendizaje automático. Cada objeto *DataLoader* contiene los datos relativos a los descriptores moleculares de cada molécula, así como al valor correspondiente de *target*.
Graph_Dataset()	Se construye mediante una lista de moléculas en formato SMILES, una lista con el valor del indicador o *target* correspondiente a cada molécula y un parámetro de *batch size* que por defecto es 32. Devuelve los siguientes atributos: · smiles: Lista de moléculas en formato SMILES. · target: Tensor con los valores de *target* de cada molécula. · Graphs: Instancia de la clase Graphs() correspondiente. · bs: Valor del parámetro *batch size*. Contiene el método: · create_dataset(): Devuelve tres objetos en formato *DataLoader* correspondientes a los subconjuntos de datos de *entrenamiento*, *validación* y *test* típicos en el marco de trabajo de modelos de redes neuronales, así como una lista con los índices numéricos de las posiciones de las moléculas en la lista proporcionada para instanciar la clase; únicamente con los índices correspondientes a aquellas moléculas que han podido ser correctamente procesadas sin error. Cada objeto *DataLoader* contiene los datos relativos a los grafos de cada molécula, así como al valor correspondiente de *target*.

Tabla 3. Nombre y descripción de las clases creadas para el procesamiento previo a la construcción de cada tipo de dataset

Paso 4. Una vez construidas las clases descritas en la tabla anterior, se han calculado los dos conjuntos de *datasets* correspondientes para cada una de las tuplas (TARGET - INDICADOR), es decir, un *dataset* de grafos en formato 'data loader' y un *dataset* de descriptores moleculares en formato 'data frame' para cada pareja.

Para la creación de los conjuntos de datos, se ha definido la función que se detalla en la Tabla 4.

Función	**Descripción**
compute_dataset()	Toma como argumento una lista con el conjunto de *targets*, el diccionario con las rutas donde ha de haber una ruta llamada 'path_dataloaders' y otra llamada 'path_dataframes'

	para los *datasets* de grafos y descriptores moleculares respectivamente, el *dataset* completo con todos los registros por cada *target* y una cadena de texto conforme si se desea calcular el conjunto de datos de grafos, en cuyo caso el parámetro ha de ser 'Data_Loaders' o el conjunto de datos de descriptores moleculares, en cuyo caso el parámetro ha de ser 'Data_Frames'. Como output, la función genera, para cada pareja *target* - *indicador*, tres archivos en el caso del *dataset* de grafos (test, validation y test, con un porcentaje de 80%, 10% y 10% de los datos respectivamente) y dos archivos en el caso del *dataset* de descriptores moleculares (80% y 20% de los datos respectivamente). La elección de estos porcentajes se escoge en base al principio de Pareto aplicado a la distribución de conjuntos para los ejercicios de machine learning / deep learning (1). Los archivos generados se guardan en las rutas indicadas en el diccionario pasado por parámetro. Los archivos tienen formato '.PTH' en el caso de los *data loaders* y formato '.CSV' en el caso de los *data frames.*

Tabla 4. Nombre y descripción de la función creada para la construcción de cada tipo de dataset

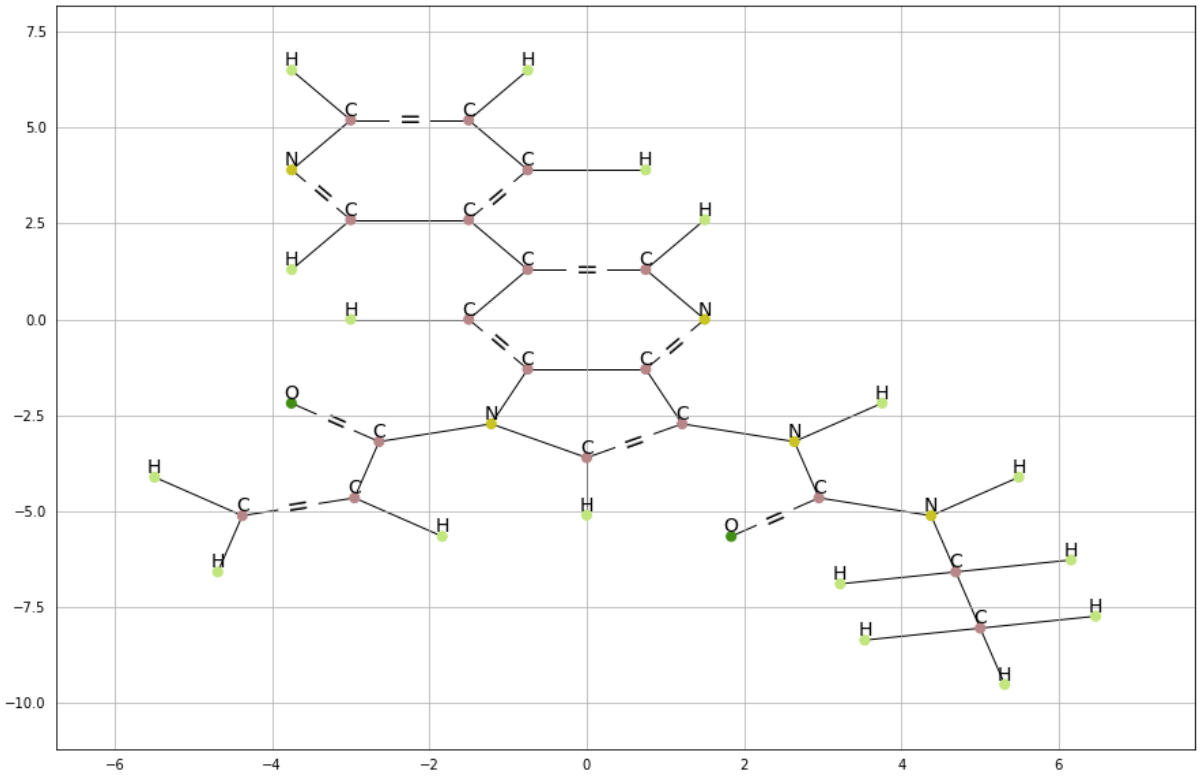

Imagen 5. Gráfico resultante de aplicar la función 'plot_molecule()' de la clase Graph() sobre una molécula de ejemplo, en este caso se trata de la molécula con cadena SMILES: 'C=CC(=O)n1cc(NC(=O)NCC)c2ncc(-c3cccnc3)cc21'

4.1. En el caso del *dataset* de grafos, el tamaño de cada una de los conjuntos se muestra en la Tabla 5. No ha sido necesario un pre-tratamiento más allá del proporcionado por las funciones de construcción del *dataset*.

	Indicador de Bioactividad					
	MIC			IC50		
	TRAIN	VALID	TEST	TRAIN	VALID	TEST
E. Coli	17.033	2.129	2.130	631	79	79
E. Coli Prot	-	-	-	272	34	34
P. Aeruginosa	10.512	1.314	1.315	337	42	43

P. Aeruginosa Prot	-	-	-	312	39	39
S. Aureus	24.948	3.119	3.119	741	93	93
S. Aureus Prot	-	-	-	177	22	23

Tabla 5. Cantidad de registros en el conjunto de datos de grafos para cada uno de los split, target e indicador

El tiempo de ejecución para la creación de este *dataset* ha sido de aproximadamente 24 minutos.

Tal y como se menciona en la Tabla 3, la información relativa a cada grafo es un objeto en formato “Data” de la librería “Pytorch Geometric” que contiene la información mostrada en la Tabla 6.

Atributo del objeto Data	**Descripción**
x	Tensor con dimensiones [número de nodos, número de atributos por nodo] en el que se almacenan los datos relativos al tipo de átomo correspondiente a cada nodo del grafo. El tensor incluye la información relativa a la codificación de ese átomo según la etiquetación ponderada de su especie química.
pos	Tensor con dimensiones [número de nodos, número de dimensiones] en el que se almacenan las parejas de coordenadas bidimensionales en formato [x, y] para cada una de las moléculas. El número de nodos varía según la molécula en particular, mientras que el número de dimensiones siempre es 2.
edge_index	Tensor con dimensiones [2, número de enlaces] en el que se almacenan las parejas de átomos indexados que tienen una conexión entre ellos. Las parejas han de estar duplicadas pero con las posiciones de los átomos intercambiadas, ya que la conexión se considera bidireccional. **Nota importante**: No se considera el tipo de enlace, ya que la capa neuronal que se detallará más adelante utiliza atributos para los enlaces sobre los que previamente se ha de saber su ponderación (relevancia) en el modelo y, en este escenario, se desconoce *a priori* qué tipo de enlace influye más en el nivel de bioactividad.
y	Tensor con dimensiones [número de *outputs*] en el que se almacena la información relativa al valor del indicador (o *target*) de cada molécula, sea IC50 o MIC según el caso correspondiente.

Tabla 6. Información que contiene cada uno de los tensores del dataset de grafos

Es importante notar que en este estudio se trabaja con un modelo estático de representación en dos dimensiones de cada molécula en cuestión. Esto implica que no se tienen en cuenta factores que afectan a la representación 2D obtenida a través de las coordenadas calculadas con la librería RdKit, tales como la información perdida al haber reducido la tercera dimensión espacial, así como la variación de las posiciones debido a los movimientos vibracionales de los átomos en la propia molécula.

Los archivos resultantes se han guardado en el directorio ‘DATA_LOADERS’ del entorno de trabajo y se adjunta una copia descargada con el presente documento.

4.2. Modificación: En el caso del *dataset* de descriptores moleculares, antes de realizar el cómputo de todos los conjuntos se realiza una selección de parámetros (***feature selection***). La librería RdKIT, que calcula los descriptores moleculares a partir de las cadenas en formato SMILES, proporciona más de 208 opciones de descriptores para calcular. Es necesario analizar y descartar los campos correlacionados entre sí para evitar problemas de multicolinealidad y los campos con baja o nula varianza estadística que no aportan información significativa a la explicación de la variable de salida.

Así pues, para cada dupla (TARGET - INDICADOR) se realiza una selección de descriptores moleculares por

separado, estudiando la variabilidad y la correlación de los descriptores en cada caso particular al tratarse ciertamente de conjuntos de datos independientes entre sí. Pese a mencionarse ciertos descriptores moleculares, las descripciones detalladas de cada uno de los descriptores obtenidos al final de este proceso se mostrarán más adelante. Los pasos que se han seguido para cada dupla son los siguientes:

1. De la lista completa de descriptores moleculares ofrecidos por RdKIT (Apéndices 1) se han descartado de forma directa los que se indican a continuación, en relación a la información obtenida en [c] y [d] (en este sentido, la documentación de la librería RdKIT es escasa y críptica respecto a la información proporcionada sobre la descripción de descriptores moleculares, con lo que se han consultado fuentes externas).
   - MaxAbsEStateIndex, MinAbsEStateIndex, MaxAbsPartitalCharge, MinAbsPartialCharge: ya existen los mismos indicadores sin el valor absoluto, con lo que sería información redundante.
   - FpDensityMorgan 1 y 2, Kappa 1 y 2: en ambos casos se utilizan, respectivamente 'FpDensityMorgan3' y 'Kappa3', ya que utilizarlos todos sería información redundante.
   - Indicadores que comienzan con 'SMR_VSA': índices calculados en base a las contribuciones atómicas a la refractividad molar, que ya se tiene en consideración mediante el indicador MolMR.
   - Indicadores que comienzan con 'SLogP_VSA': índices de la suma de las contribuciones atómicas al indicador LogP, el cual se considerará como resumen de todos los índices.
   - Indicadores que comienzan con 'chi': índices topológicos de conectividad molecular resumidos mediante el indicador 'Hall-Kier Alpha'.
   - Indicadores que comienzan con 'PEOE_VSA': indicadores de carga parcial, la cual ya se tiene en cuenta mediante los indicadores 'MinPartialCharge' y 'MaxPartialCharge'.
   - Indicadores que comienzan con 'VSA_EState' y 'EState_VSA': sus descripciones no aparecen detalladas en la documentación.
   - Indicadores que comienzan con 'BCUT_2D': sus descripciones no aparecen detalladas en la documentación.
   - A priori, se descartan todos los indicadores que comienzan por 'fr' y que indican el número de una cierta especie química dentro del compuesto. El motivo del descarte es la gran cantidad de especies distintas que hay, generando una tabla con mucha información de vacío (llena de ceros).

   Tras este primer corte, han quedado **38** descriptores restantes.
2. Se ha calculado el conjunto completo de todos los descriptores moleculares proporcionados por RdKIT a excepción de aquellos eliminados que se indican en el paso anterior. Los archivos resultantes se han guardado en el directorio '/DESCRIPTORS/RAW' del entorno de trabajo y se adjunta una copia descargada con el presente documento.
3. Se han descartado todos los descriptores con varianza estadística igual a cero.
4. Se ha calculado la correlación de Pearson de todas las parejas de variables. De presentar una correlación mayor a 0.8, se descarta el descriptor que tenga un *p valor* menor en relación a la variable de salida (sea esta MIC o IC50 según el caso correspondiente) y se mantiene el otro. La elección del valor de correlación ha sido arbitraria, ya que no se ha encontrado en la bibliografía un valor a partir del cual se considere una correlación 'alta' o 'baja' a nivel de significancia estadística. En un análisis más profundo y avanzado de selección de variables, se podría utilizar un método de correlación no-lineal.
5. Se ha calculado la varianza estadística de cada descriptor molecular resultante y se ha analizado su distribución de valores. Los resultados se muestran a continuación en la Tabla 7.

		Distribución del número de indicadores en función del porcentaje del valor máximo de varianza obtenida			
Target / Indicador	# de descriptores tras los pasos 1 y 2	Entre 0 y 25%	Entre 25% y 50%	Entre 50% y 75%	Mayor que 75%
E_COLI / MIC	20	6	7	3	4
E_COLI / IC50	19	6	2	7	4

E_COLI_PROT / IC50	20	14	0	0	6
P_AERUGINOSA / MIC	19	7	6	3	3
P_AERUGINOSA / IC50	19	10	3	1	5
P_AERUGINOSA_PROT / IC50	25	12	5	3	5
S_AUREUS / MIC	19	6	6	5	2
S_AUREUS / IC50	19	3	6	5	5
S_AUREUS_PROT / IC50	20	15	0	2	3

Tabla 7. Para cada dupla target-indicador, se muestra el número de descriptores resultantes tras aplicar los criterios de descarte para los campos con varianza nula y con correlación > 0.8. Se muestra también la distribución de estos descriptores según el valor de su varianza en relación al valor máximo obtenido

Si se representan los histogramas de la distribuciones de cada pareja Target / Indicador se obtiene la siguiente serie de figuras:

E_COLI MIC

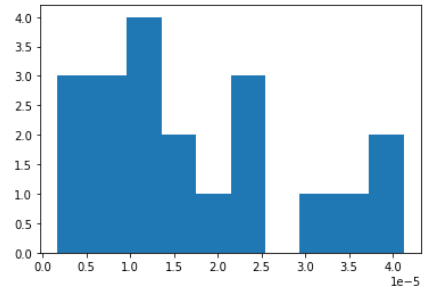

Fig. 1

E_COLI IC50

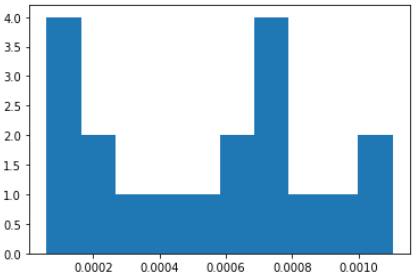

Fig. 2

E_COLI_PROT IC50

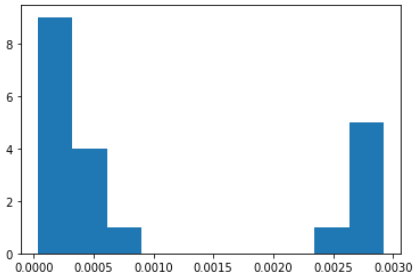

Fig. 3

P_AEURUGINOSA MIC

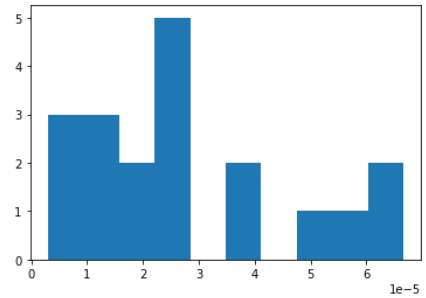

Fig. 4

P_AERUGINOSA IC50

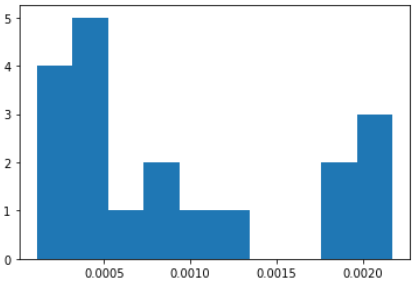

Fig. 5

P_AERUGINOSA_PROT IC50

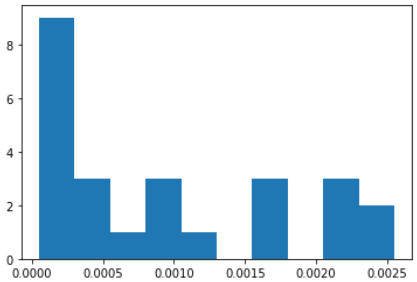

Fig. 6

S_AUREUS MIC

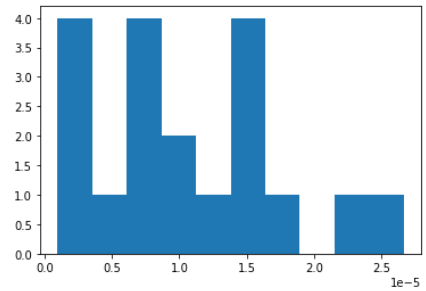

Fig. 7

S_AUREUS IC50

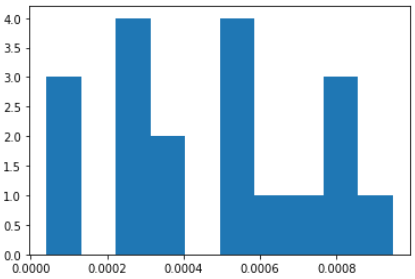

Fig. 8

S_AUREUS_PROT IC50

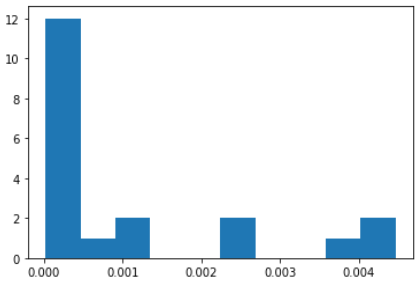

Fig. 9

La distribución resulta irregular en cada caso y, pese a que la Tabla 7 indica que un buen número de campos se acumula entre el 0 y el 25% del valor de varianza máxima, esto no proporciona una idea clara sobre cuál

debería ser el corte a partir del cual realizar la selección de variables. Tampoco se ha encontrado en la bibliografía un valor estadísticamente significativo que establezca un límite de varianza para descartar variables en este contexto de selección.

6. Así pues, se han eliminado aquellos descriptores cuyo valor de varianza estuviera entre el 0 y el 5% del valor máximo de varianza obtenido para cada conjunto de datos. La elección de este valor ha sido arbitraria y con el único objetivo de no tener en cuenta descriptores con varianza no-nula pero muy cercana a 0 en relación con el resto de variables.

Target - Indicador	**# de Descriptores Seleccionados**
E_COLI / MIC	18
E_COLI / IC50	19
E_COLI_PROT / IC50	15
P_AERUGINOSA / MIC	17
P_AERUGINOSA / IC50	18
P_AERUGINOSA_PROT / IC50	23
S_AUREUS / MIC	17
S_AUREUS / IC50	18
S_AUREUS_PROT / IC50	14

Tabla 8. Descripción de los distintos descriptores moleculares seleccionados que pueden encontrarse en al menos uno de los conjuntos de datos creados

Los descriptores seleccionados para cada conjunto de datos se guardan en sendos archivos con formato '.CSV' en el directorio 'DESCRIPTORS', dentro del entorno de trabajo y se adjunta una copia descargada con el presente documento. La descripción de cada descriptor se muestra a continuación en la Tabla 9 y la selección ha resultado en un total de **30** descriptores distintos. El tipo de datos de todos los descriptores es coma flotante.

Nombre del Descriptor (RdKit)	**Descripción (a partir de [c] y [d])**
BalabanJ	Índice de Balaban, un tipo de índice de conectividad, calculado para un grafo con *n* nodos, *m* enlaces y *c* componentes conectadas. Definido en detalle en [2].
BertzCT	Índice de Bertz, un tipo de índice de conectividad para cuantificar la complejidad de un grafo. Definido en detalle en [3].
FractionCSP3	Fracción de átomos de C que tienen hibridación SP3 (unión de un orbital s con tres orbitales p (px, py y pz).
HallKierAlpha	Indicador topológico Hall-Kier-Alpha, relaciona el número de enlaces, el de átomos y sus radios. Definido en detalle en [4].
Ipc	(Información de los Coeficientes del Polinomio): Información relativa al polinomio característico de la matriz de adyacencia del grafo de una molécula sin considerar los átomos de hidrógeno. No se ha encontrado el cálculo exacto en la bibliografía.
Kappa3	Indicador topológico Hall-Kier-Kappa, relaciona el relaciona el número de enlaces y el de átomos. Definido en detalle en [4].

MaxEStateIndex	Máximo índice del estado electro-topológico de la molécula. Definido en detalle en [5].
MaxPartialCharge	Máxima carga parcial de la molécula.
MinEStateIndex	Mínimo índice del estado electro-topológico de la molécula. Definido en detalle en [5].
MinPartialCharge	Mínima carga parcial de la molécula.
MolLogP	Coeficiente de reparto octanol-agua de la molécula (i.e., cociente entre las concentraciones de esa sustancia en una mezcla bifásica formada por dos disolventes inmiscibles en equilibrio: n-octanol y agua). Definido en detalle en [6].
NumAliphaticCarbocycles	Número de carbociclos alifáticos (que contienen al menos un enlace no aromático) de la molécula.
NumAliphaticHeterocycles	Número de heterociclos alifáticos (que contienen al menos un enlace no aromático) de la molécula.
NumAliphaticRings	Número de anillos alifáticos (que contienen al menos un enlace no aromático) de la molécula.
NumAromaticCarbocycles	Número de carbociclos aromáticos de la molécula.
NumAromaticHeterocycles	Número de heterociclos aromáticos de la molécula.
NumAromaticRings	Número de anillos aromáticos de la molécula.
NumHAcceptors	Número de aceptores de enlace de hidrógeno de la molécula.
NumHDonors	Número de donantes de enlaces de hidrógeno de la molécula.
NumHeteroatoms	Número de heteroátomos (i.e., cualquier átomo, salvo carbono o hidrógeno).
NumRadicalElectrons	El número de electrones radicales que tiene la molécula (no da información sobre el estado de espín).
NumRotatableBonds	Número de enlaces rotativos de la molécula (i.e., cualquier enlace único que no sea de anillo, unido a un átomo no terminal que no sea de hidrógeno).
NumSaturatedCarbocycles	Número de carbociclos saturados de una molécula.
NumSaturatedHeterocycles	Número de heterociclos saturados de una molécula.
NumSaturatedRings	Número de anillos saturados para una molécula.
RingCount	Número total de anillos de la molécula.
TPSA	Área de superficie polar calculada utilizando contribuciones de grupo para aproximar el área de superficie polar solo a partir de la información de la tabla de conexiones. La parametrización se puede encontrar en [7].
qed	Estimación cuantitativa de la fármaco-semblanza de la molécula. Definido en detalle en [8].

Tabla 9. Descripción de los distintos descriptores moleculares seleccionados que pueden encontrarse en al menos uno de los conjuntos de datos creados

Nota Importante: Para la selección de variables, se ha considerado utilizar la librería MLXTend con la función de selección exhaustiva de parámetros [e] en lugar de analizar la distribución de varianzas. Este método consiste en utilizar un modelo base del algoritmo de Machine Learning seleccionado y probar todas las combinaciones posibles de parámetros propuestos para obtener qué combinación obtiene el mejor rendimiento. No obstante, al tener un mínimo de 19 variables (Tabla 7) tras aplicar los dos primeros pasos de selección, esto implicaría del orden de 1e16 pruebas por cada pareja indicador-target según el cálculo combinatorio de variaciones entre permutaciones posibles, lo cual generaría un tiempo de cálculo inalcanzable con el ahora disponible para el desarrollo del proyecto.

Los archivos resultantes se han guardado en el directorio 'DATA_FRAMES' del entorno de trabajo y se adjunta una copia descargada con el presente documento.

Finalmente, el tamaño de cada una de los conjuntos se muestra en la Tabla 10.

	Indicador de Bioactividad			
	MIC		IC50	
	TRAIN	TEST	TRAIN	TEST
E. Coli	19.396	4.849	724	181
E. Coli Prot	-	-	272	69
P. Aeruginosa	12.005	3.002	367	92
P. Aeruginosa Prot	-	-	312	79
S. Aureus	30.016	7.505	843	211
S. Aureus Prot	-	-	178	45

Tabla 10. Cantidad de registros en el conjunto de datos de descriptores moleculares para cada uno de los split, target e indicador

El tiempo de ejecución para la creación de este *dataset* ha sido de aproximadamente 36 minutos.

CONSTRUCCIÓN DE MODELOS Y FUNCIONES DE AJUSTE / ENTRENAMIENTO

• Se ha creado en el directorio de desarrollo "DEV" un *notebook* colaborativo con la herramienta Google Colab llamado "MODELS AND TRAIN_TEST LOOPS.ipynb" para la instanciación de modelos de Machine Learning, la construcción del modelo de Deep Learning, así como la implementación de los bucles de entrenamiento y test. El desarrollo del código se ha realizado íntegramente con Python y el conjunto de librerías descargadas se puede consultar en el propio *notebook.* El enlace del *notebook* se muestra en la bibliografía [g] y se adjunta una copia descargada con el presente documento.

• **Modelo Deep Learning**:

Para poder realizar predicciones de bioactividad en función de los tensores con información relativa a los grafos 2D de las moléculas, se ha decidido trabajar con la arquitectura propuesta en [6] por los siguientes motivos:

- El *input* es independiente del tamaño de la muestra gracias al uso de capas de redes neuronales recurrentes [10], lo cual permite que cada uno de los tensores (i.e., moléculas) de entrada pueda tener distinto número de nodos y enlaces.
- Es una arquitectura fácilmente implementable desde la librería Pytorch Geometric del entorno Pytorch y es el tipo de capa convolucional base que ofrece.
- Ha presentado buenos resultados en distintos escenarios relacionados con la tarea de aprendizaje supervisado a partir de grafos [11].

Pese a estas ventajas, el principal inconveniente que presenta esta arquitectura es la imposibilidad de trabajar con enlaces categorizados - únicamente permite asociar una ponderación del tipo de conexiones de los grafos que ha de ser conocida previamente.

La arquitectura propuesta para este escenario se muestra a continuación en la Imagen 6.

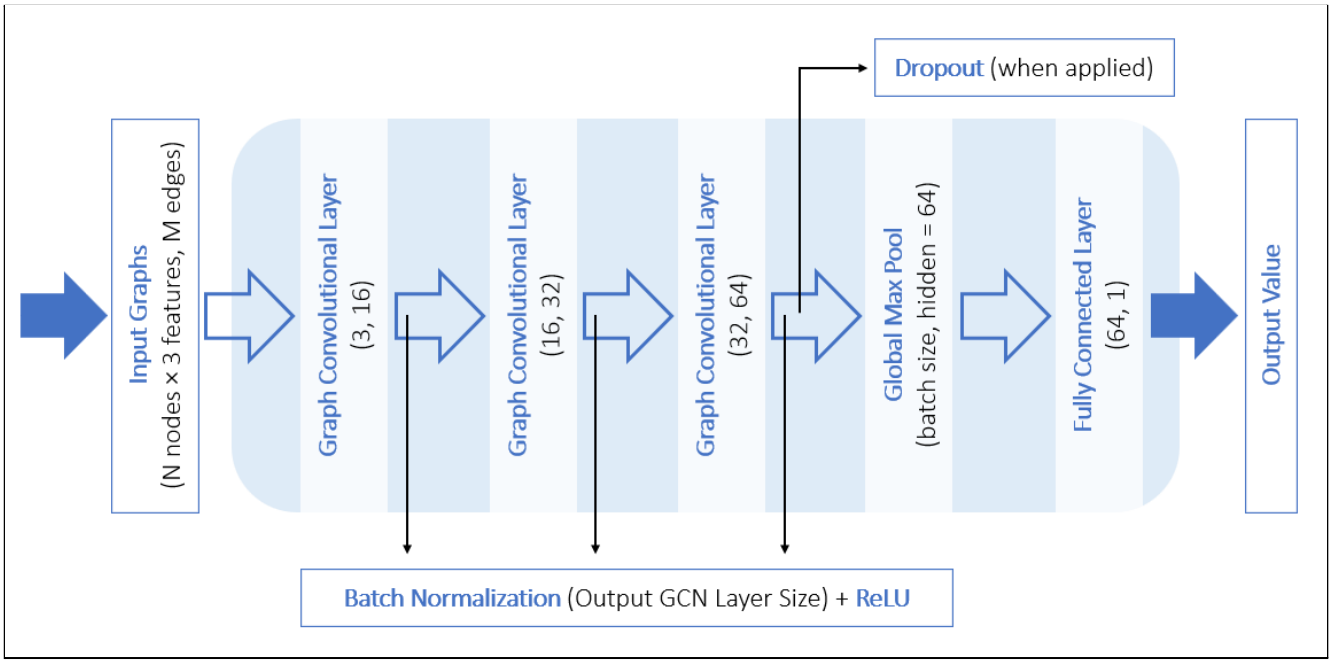

Imagen 6. Arquitectura del modelo de Deep Learning basado en capas gráficas convolucionales y capas lineales

El modelo consiste en la secuenciación de tres capas convolucionales para la abstracción sucesiva de la información de los datos contenidos en el tensor de input, seguidas de una capa de agrupación máxima para devolver la dimensionalidad del batch a su tamaño original y una capa lineal para obtener el valor de salida a partir de la abstracción generada por la convolución. En la memoria final del trabajo se expondrá con más detalle esta arquitectura. La decisión de usar tres capas ha sido arbitraria, pues el número de capas sería un hiper-parámetro a optimizar, y la dimensionalidad creciente con potencia 2 es escogida para la optimización de recursos en el procesador físico de cálculo. A continuación se muestra en la Tabla 11 la descripción de la clase creada para la implementación de esta arquitectura.

Clase	**Funcionalidad (atributos y métodos)**
GCN3()	Se construye como una clase heredada de *torch.nn.Module* y representa la arquitectura de red neuronal correspondiente a la Imagen 6. Toma como parámetro el número de características por cada elemento de entrada, que por defecto toma el valor de 3. Se añaden los siguientes atributos: · k: número de características por cada elemento de entrada. · conv1: primera capa convolucional según la clase de Pytorch Geometric GCNConv, con dimensiones (k, 16) · conv2: segunda capa convolucional según la clase de Pytorch Geometric GCNConv, con dimensiones (16, 32) · conv3: tercera capa convolucional según la clase de Pytorch Geometric GCNConv, con dimensiones (32, 64) · fc: capa lineal con dimensiones (64, 1) · bn1: normalización del batch con dimensión 16 · bn2: normalización del batch con dimensión 32

	· bn3: normalización del batch con dimensión 64 · dropout: capa de desactivación aleatoria de neuronas, inicialmente la probabilidad de desactivación se establece como 0 Contiene el método añadido: · forward(x, edge_index, batch): toma como parámetros las características de cada molécula (i.e., coordenadas y tipo de átomo), la información relativa a las conexiones entre nodos y el tamaño del batch. Computa el paso de los datos a través de toda la red neuronal según la arquitectura de la Imagen 1. Devuelve el valor numérico calculado al final de todo el flujo.

Tabla 11. Cantidad de registros en el conjunto de datos de descriptores moleculares para cada uno de los split, target e indicador

Para comprobar que la arquitectura está bien construida, se ha seleccionado un batch de 32 moléculas del conjunto de datos de entrenamiento de la tupla E_COLI - MIC y se ha pasado a través de la función 'forward()' de la clase de la red neuronal. Este paso es necesario para tener la seguridad que no existen errores en el flujo de datos a través de la arquitectura, antes de que el modelo sea entrenado. El resultado del test ha sido satisfactorio, pues se ha obtenido un tensor con 32 valores numéricos, tal y como se muestra en la Imagen 7, correspondientes a cada uno de los valores que próximamente tratarán de igualarse a los esperados según el valor asociado en la base de datos original.

```
# Forward pass:
single_output = model.forward(features, edges, value.batch.to(device))
single_output
```

```
tensor([[-0.7032],
        [-0.2976],
        [-0.0646],
        [-0.5571],
        [-0.4105],
        [-0.4194],
        [-0.9365],
        [-0.2786],
        [-0.9855],
        [-0.5543],
        [ 0.2650],
        [-0.4168],
        [-0.6780],
        [-0.6673],
        [-0.6585],
        [-0.6413],
        [-0.9089],
        [-0.7838],
        [-0.0511],
        [-0.3942],
        [-1.0368],
        [-0.2220],
        [ 0.2285],
        [ 0.1987],
        [-0.1637],
        [-0.6529],
        [-0.4658],
        [-0.6008],
        [-0.4092],
        [-0.3825],
        [-0.5298],
        [-0.1600]], grad_fn=<AddmmBackward0>)
```

```
single_output.shape
```

```
torch.Size([32, 1])
```

Imagen 7. Este es el único fragmento de código que se muestra en este informe, pero es necesario para mostrar la efectividad del método 'forward()' al testearlo con un batch de datos de entrenamiento. El código completo se encuentra en el notebook mencionado anteriormente en este mismo apartado

• **Bucles de entrenamiento, validación y test para los modelos de Deep Learning**:

En primer lugar, se ha definido la métrica de evaluación de los modelos. La métrica escogida es la raíz del error cuadrático medio, definida por la ecuación (4).

$$RMSE = \sqrt{\frac{\sum_{1}^{N}(Predicted_i - Observed_i)^2}{N}} \quad (4)$$

El valor de RMSE no se puede calcular directamente tras la obtención de valores proporcionados por la red neuronal para cada *batch* y promediarlo una vez terminada la época (en la que ya se ha observado todo el conjunto de datos), puesto que la suma de RMSE (i.e., la suma de raíces) no es equivalente a la suma de sus radicandos. Para solucionar esto, es necesario crear una clase que almacene el valor de los radicandos en cada *batch* y que calcule el valor de época del RMSE al acabar todas las observaciones. La clase implementada se detalla a continuación en la Tabla 12.

Clase	**Funcionalidad (atributos y métodos)**
RMSE()	Se construye como una clase heredada de *torchmetrics.Metric* y permite actualizar el valor de RMSE tras la evaluación de cada batch de entrenamiento para el posterior cálculo de la métrica al final de cada época. No añade atributos nuevos. Inicializa los estados de las variables internas 'sum_squared_errors' y 'n_observations' a cero, correspondientes al numerador y denominador del radicando del RMSE. Contiene los métodos añadidos: · update(predictions, target): toma como parámetros las predicciones y los valores esperados del indicador de bioactividad y actualiza las variables internas 'sum_squared_errors' y 'n_observations' correspondientes al numerador y denominador del RMSE según la ecuación (4). · compute(): toma los valores de las variables internas 'sum_squared_errors' y 'n_observations' y devuelve el valor de RMSE calculado según la ecuación (4).

Tabla 12. Nombre y descripción de las funciones creadas para el entrenamiento de modelos de Deep Learning

La función de pérdida para el entrenamiento de los modelos es el error cuadrático medio, implementado a partir de [h], aplicando la reducción 'promedio' por defecto, que aplica de forma directa la actualización de valores expuesto anteriormente mediante el cálculo de la métrica.

Para los bucles de entrenamiento y validación se ha definido una función genérica de ajuste que contiene las funciones de 'entrenamiento' del modelo, en la que se ajustan los parámetros (pesos y sesgos) a través del método de propagación inversa, y de 'validación' del modelo, donde se testean las métricas de evaluación con un conjunto de datos no-visto para el entrenamiento. El detalle de las funciones implementadas en el *notebook* anteriormente mencionado en este apartado se muestra en la Tabla 13.

Función	**Descripción**
fit()	Toma como parámetros el modelo instanciado desde la clase correspondiente, los conjuntos de datos de entrenamiento y validación en formato 'Data Loader', la ruta donde almacenar los resultados, el número de épocas, el ratio de aprendizaje, el decaimiento de pesos, el optimizador y el *scheduler*. Genera dos archivos en 'TensorBoard' con la librería 'TensorFlow' de Python [12] para el registro del progreso del ajuste en cuanto a los valores de pérdida y métrica de evaluación. Se establece una pérdida máxima de valor infinito. Para cada época, aplica las funciones train_epoch() y valid_epoch(), actualiza el *scheduler* de haberlo proporcionado, registra en los *logs* de 'TensorBoard' los valores de pérdida y métrica de evaluación para los conjuntos de entrenamiento y validación y, por último, guarda el modelo si su valor de métrica de evaluación sobre el conjunto de datos de validación es más bajo que el último valor registrado (el valor de métrica se actualiza si es menor que el de la época anterior y, en caso de obtener

	un modelo mejor, se sobre-escribe). Devuelve el mejor valor de métrica de evaluación calculada y los pesos del modelo que ha obtenido dicho valor de métrica.
train_epoch()	Toma como parámetros el modelo instanciado desde la clase correspondiente, el conjunto de datos de entrenamiento en formato 'Data Loader', el optimizador, la función de pérdida del modelo, la métrica de evaluación del modelo, el aparato de cálculo (GPU/CPU) y el *scheduler*. Calcula para cada *batch* las predicciones del modelo para los datos proporcionados y los valores de pérdida y métrica indicadas (MSE y RMSE, respectivamente). Según el valor de la pérdida, se actualizan los pesos del modelo mediante el método de propagación inversa a partir del descenso de gradiente. Finalmente, promedia los valores de pérdida y métrica para todos los *batch* (i.e., una época completa). En cada *batch* se actualiza el *scheduler* en caso de haberlo proporcionado por parámetro. Devuelve los valores calculados de pérdida y métrica.
valid_epoch()	Toma como parámetros el modelo instanciado desde la clase correspondiente, el conjunto de datos de validación en formato 'Data Loader', la función de pérdida del modelo, la métrica de evaluación del modelo, el aparato de cálculo (GPU/CPU) y el *scheduler*. Calcula para cada *batch* las predicciones del modelo para los datos proporcionados y los valores de pérdida y métrica indicadas (MSE y RMSE, respectivamente). Finalmente, promedia estos valores para todos los *batch* (i.e., una época completa). En cada *batch* se actualiza el *scheduler* en caso de haberlo proporcionado por parámetro. Devuelve los valores calculados de pérdida y métrica.

Tabla 13. Nombre y descripción de las funciones creadas para el entrenamiento de modelos de Deep Learning

Para el test de los modelos, se ha creado una función similar a la de validación, simplificada ya que esta no necesita los elementos necesarios para el entrenamiento. El detalle de las funciones implementadas en el *notebook* anteriormente mencionado en este apartado se muestra en la Tabla 14.

Función	**Descripción**
test()	Toma como parámetros el modelo instanciado desde la clase correspondiente, el conjunto de datos de validación en formato 'Data Loader', y el archivo con los pesos correspondientes al modelo en cuestión que se desea evaluar. Calcula en un único pase las predicciones del modelo para los datos proporcionados y el valor de métrica indicada (RMSE). Devuelve el valor RMSE calculado.

Tabla 14. Nombre y descripción de la función creada para el testeo de modelos de Deep Learning

Como optimizadores se considerarán 'Adam' y 'SGD' y, en caso de necesitar *schedulers*, se considerarán 'StepLR', 'OneCycleLR' y 'ReduceOnPlateau'. Sobre optimizadores y *schedulers* se desarrollará su parte correspondiente en la siguiente fase del trabajo, haciendo referencia a la bibliografía correspondiente.

El directorio 'GCN' del entorno de trabajo, que actualmente se encuentra vacío, servirá para almacenar los resultados de los experimentos sobre esta red neuronal.

• **Modelos Machine Learning**:

Los modelos de Machine Learning se instancian directamente desde las librerías correspondientes. Para cada uno de los algoritmos seleccionados, se han escogido las librerías [i], [j] e [k] para su implementación.

Dado que no ha habido tiempo suficiente para la construcción de la función general de entrenamiento, no se han podido instanciar ni testear los modelos sobre un conjunto de datos de prueba.

EVALUACIÓN DE LAS TAREAS REALIZADAS

En general, se han resuelto satisfactoriamente la mayoría de tareas planificadas para esta primera fase del proyecto en relación al planteamiento realizado en el Plan de Trabajo. Partes de este informe se utilizarán en la redacción de la memoria final del proyecto.

PREPROCESAMIENTO DE DATOS

Todos los objetivos establecidos en relación al preprocesamiento de datos se han resuelto satisfactoriamente. Prueba de ello son los distintos ficheros que se adjuntan con este informe.

En general, los primeros pasos han resultado más complejos de elaborar de lo que se esperaba. La creación de clases en Python para la escalabilidad y buen funcionamiento del código ha retrasado el desarrollo.

Se ha añadido un bloque más no contemplado, la selección de variables. Siguiendo las guías del tutor, se ha puesto énfasis en establecer algunos criterios de selección para no escoger descriptores moleculares al azar, que más adelante pudieran presentar problemas de multicolinealidad o ruido en el modelo. Esto también ha retrasado el desarrollo de esta fase.

CONSTRUCCIÓN DE MODELOS Y EXPERIMENTOS

Dado que esta primera fase del proyecto ha requerido profundizar en la selección de variables para el conjunto de datos de descriptores moleculares, finalmente no ha habido tiempo suficiente para la implementación de los primeros entrenamientos de prueba antes de la generalización de múltiples entrenamientos para tratar de optimizar los distintos modelos de Deep Learning / Machine Learning, lo cual se deja para la siguiente fase.

La implementación de la arquitectura de Deep Learning se ha llevado a cabo satisfactoriamente, así como su testeo con una prueba de mini-batch. No obstante, dado que los algoritmos de Machine Learning se planea implementarlos directamente en una función general de entrenamiento todavía no construida, aún no han sido importados ni aplicados.

El preprocesamiento de datos es una fase delicada en la que pueden aparecer imprevistos ocasionados por la exploración inicial (en la que uno no sabe qué va a encontrarse de manera subyacente), no se considera lo transcurrido como un gran retraso y se espera poder proceder con los objetivos del proyecto establecidos en primer lugar. Así pues y teniendo esto en cuenta, el cronograma ha quedado modificado de la siguiente manera:

<u>Fase 1:</u>

Como se puede observar en la Imagen 8, las pruebas de entrenamiento de modelos se sustituyen por la selección de variables y otros aspectos de preprocesamiento de datos.

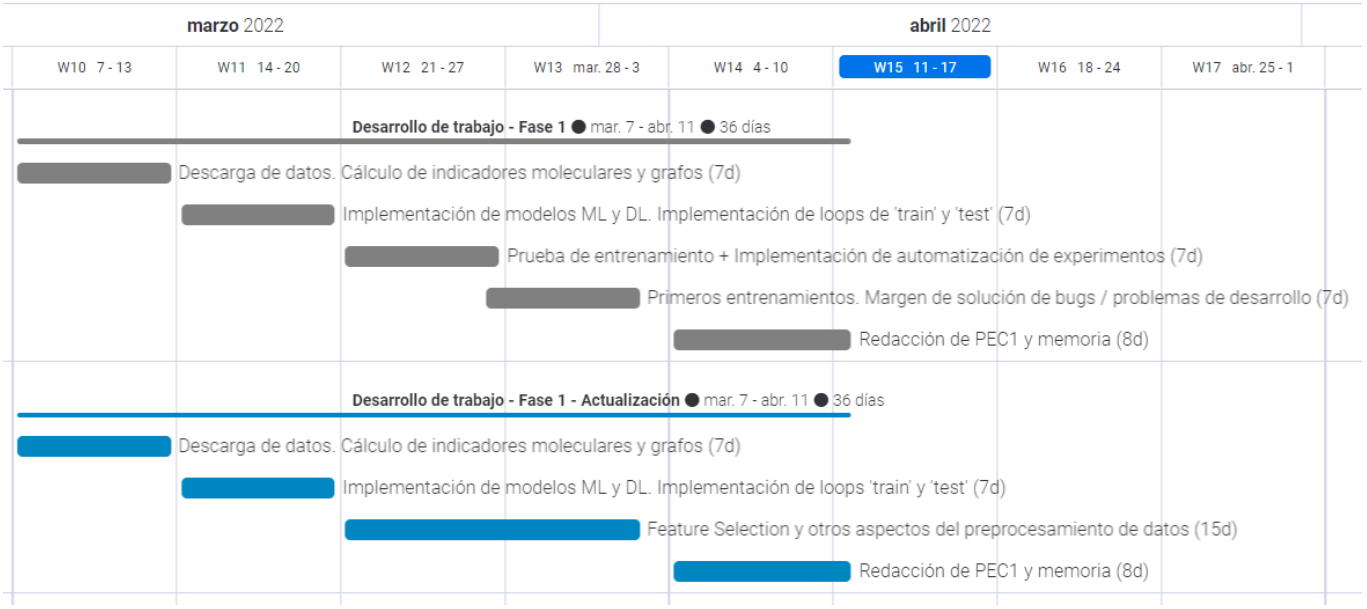

Imagen 8. Actualización de plan de trabajo para la Fase 1 del proyecto

Fase 2:

Como se puede observar en la Imagen 9, los entrenamientos se limitan únicamente a dos semanas. Con esto, se espera implementar una función de automatización para minimizar el desarrollo excesivo de código y que la mayor parte del tiempo pueda ser dedicada al análisis de múltiples experimentos para poder validar o rechazar las hipótesis iniciales del proyecto.

Imagen 9. Actualización de plan de trabajo para la Fase 2 del proyecto

Últimas Fases (Quedan inalteradas):

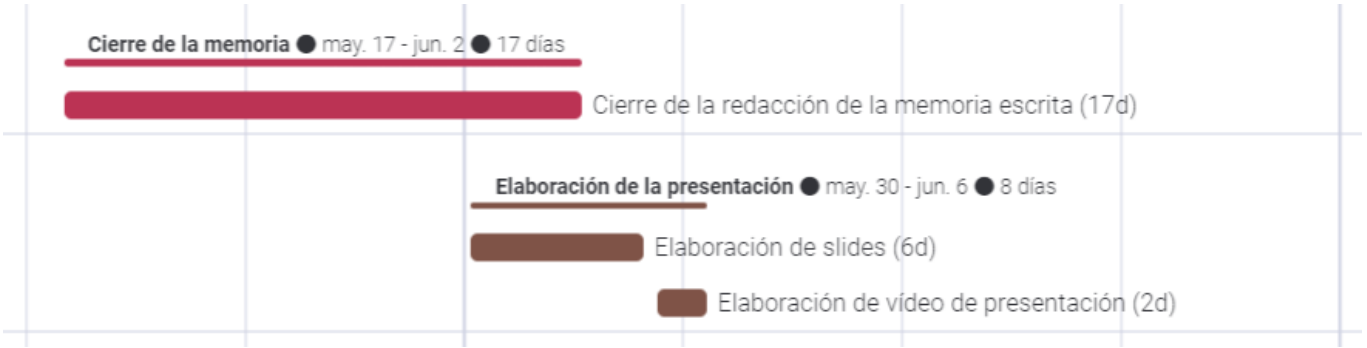

Imagen 10. Actualización de plan de trabajo para la Fase final del proyecto

REFERENCIAS Y BIBLIOGRAFÍA

[a] ChEMBL Database from European Bioinformatics Institute. (Last visit: April 2022): https://www.ebi.ac.uk/chembl/g/#search_results/all

[b] Notebook de desarrollo 'DATA_PREPROCESSING.ipynb' (Last visit: April 2022): https://colab.research.google.com/drive/1u9vtlcuErSKeg-5sR0UEWagHVfFAAcPC?usp=sharing

[c] Datagrok. CHEM Molecular descriptors. (Last visit: April 2022) https://datagrok.ai/help/domains/chem/descriptors

[d] Chemical Computing Group. MOE 2008.10. QuaSAR-Descriptor. 2D Molecular Descriptors. (Last visit: April 2022): http://www.cadaster.eu/sites/cadaster.eu/files/challenge/descr.htm

[e] MLXTend. Exhaustive Feature Selection method. (Last visit: April 2022): https://rasbt.github.io/mlxtend/user_guide/feature_selection/ExhaustiveFeatureSelector/

[f] Graph Convolutional Layer from Convolutional Layers available in Pytorch Geometric Package from Pytorch (Last Visit: April 2022): https://pytorch-geometric.readthedocs.io/en/latest/_modules/torch_geometric/nn/conv/gcn_conv.html

[g] Notebook de desarrollo 'MODELS AND TRAIN_TEST LOOPS.ipynb' (Last Visit: April 2022): https://colab.research.google.com/drive/192kig8DU5jAK_9mfCUfgV4p_W3TvVPN3?usp=sharing

[h] Criterio para la función de pérdida como error cuadrático medio en el entorno Pytorch. (Last Visit: April 2022): https://pytorch.org/docs/stable/generated/torch.nn.MSELoss.html

[i] XGBoost Regression algorithm for Python. (Last Visit: April 2022):

https://xgboost.readthedocs.io/en/stable/python/python_api.html

[j] Support Vector Machine Regression algorithm for Python. (Last Visit: April 2022):

https://scikit-learn.org/stable/modules/svm.html

[k] Random Forest Regression algorithm for Python. (Last Visit: April 2022):

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

[1] Dunford. R., Su, Q., and Tamang, E. (2014) 'The Pareto Principle', The Plymouth Student Scientist, 7(1), p. 140-148. http://hdl.handle.net/10026.1/14054

[2] Klein, D.J., Topological Indices and Related Descriptors in QSAR and QSPR Edited by James Devillers & Alexandru T. Balaban. Gordon and Breach Science Publishers: Singapore. 1999. 811 pp. 90-5699-239-2. . J. Chem. Inf. Comput. Sci.1507 VL 42 IS 6 SN 0095-2338, American Chemical Society. Published 2002/11/01. https://doi.org/10.1021/ci010441h

[3] Steven H Bertz, Branching in graphs and molecules, Discrete Applied Mathematics, Volume 19, Issues 1–3, 1988, Pages 65-83, ISSN 0166-218X, https://doi.org/10.1016/0166-218X(88)90006-6

[4] Hall, L.H., Kier, L.B.; The Molecular Connectivity Chi Indices and Kappa Shape Indices in Structure-Property Modeling; Reviews of Computational Chemistry 2 (1991).

[5] Hall, L.H., Mohney, B. and Kier, L.B. (1991), The Electrotopological State: An Atom Index for QSAR. Quant. Struct.-Act. Relat., 10: 43-51. https://doi.org/10.1002/qsar.19910100108

[6] Wildman, S.A., Crippen, G.M.; Prediction of Physiochemical Parameters by Atomic Contributions; J. Chem. Inf. Comput. Sci. 39 No. 5 (1999) 868–873.

[7] Ertl, P., Rohde, B., Selzer, P.; Fast Calculation of Molecular Polar Surface Area as a Sum of Fragment-Based Contributions and Its Application to the Prediction of Drug Transport Properties; J. Med. Chem. 43 (2000) 3714–3717.

[8] Bickerton GR, Paolini GV, Besnard J, Muresan S, Hopkins AL. Quantifying the chemical beauty of drugs. Nat Chem. 2012;4(2):90-98. Published 2012 Jan 24. https://doi.org/10.1038/nchem.1243

[9] Kipf, T. N., & Welling, M. (2016). Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* . https://doi.org/10.48550/arXiv.1609.02907

[10] Sepp Hochreiter, Jürgen Schmidhuber; Long Short-Term Memory. Neural Comput 1997; 9 (8): 1735–1780. doi: https://doi.org/10.1162/neco.1997.9.8.1735

[11] Zhang, S., Tong, H., Xu, J. et al. Graph convolutional networks: a comprehensive review. Comput Soc Netw 6, 11 (2019). https://doi.org/10.1186/s40649-019-0069-y

[12] K. Wongsuphasawat et al., "Visualizing Dataflow Graphs of Deep Learning Models in TensorFlow," in IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 1, pp. 1-12, Jan. 2018, doi: 10.1109/TVCG.2017.2744878.

APÉNDICES

(1) Lista completa de descriptores moleculares ofrecidos por RdKIT versión (2022.3.1):

'MaxEStateIndex', 'MinEStateIndex', 'MaxAbsEStateIndex', 'MinAbsEStateIndex', 'qed', 'MolWt', 'HeavyAtomMolWt', 'ExactMolWt', 'NumValenceElectrons', 'NumRadicalElectrons', 'MaxPartialCharge', 'MinPartialCharge', 'MaxAbsPartialCharge', 'MinAbsPartialCharge', 'FpDensityMorgan1', 'FpDensityMorgan2', 'FpDensityMorgan3', 'BCUT2D_MWHI', 'BCUT2D_MWLOW', 'BCUT2D_CHGHI', 'BCUT2D_CHGLO', 'BCUT2D_LOGPHI', 'BCUT2D_LOGPLOW', 'BCUT2D_MRHI', 'BCUT2D_MRLOW', 'BalabanJ', 'BertzCT', 'Chi0', 'Chi0n', 'Chi0v', 'Chi1', 'Chi1n', 'Chi1v', 'Chi2n', 'Chi2v', 'Chi3n', 'Chi3v', 'Chi4n', 'Chi4v', 'HallKierAlpha', 'Ipc', 'Kappa1', 'Kappa2', 'Kappa3', 'LabuteASA', 'PEOE_VSA1', 'PEOE_VSA10', 'PEOE_VSA11', 'PEOE_VSA12', 'PEOE_VSA13', 'PEOE_VSA14', 'PEOE_VSA2', 'PEOE_VSA3', 'PEOE_VSA4', 'PEOE_VSA5', 'PEOE_VSA6', 'PEOE_VSA7', 'PEOE_VSA8', 'PEOE_VSA9', 'SMR_VSA1', 'SMR_VSA10', 'SMR_VSA2', 'SMR_VSA3', 'SMR_VSA4', 'SMR_VSA5', 'SMR_VSA6', 'SMR_VSA7', 'SMR_VSA8', 'SMR_VSA9', 'SlogP_VSA1', 'SlogP_VSA10', 'SlogP_VSA11', 'SlogP_VSA12', 'SlogP_VSA2',

'SlogP_VSA3', 'SlogP_VSA4', 'SlogP_VSA5', 'SlogP_VSA6', 'SlogP_VSA7', 'SlogP_VSA8', 'SlogP_VSA9', 'TPSA', 'EState_VSA1', 'EState_VSA10', 'EState_VSA11', 'EState_VSA2', 'EState_VSA3', 'EState_VSA4', 'EState_VSA5', 'EState_VSA6', 'EState_VSA7', 'EState_VSA8', 'EState_VSA9', 'VSA_EState1', 'VSA_EState10', 'VSA_EState2', 'VSA_EState3', 'VSA_EState4', 'VSA_EState5', 'VSA_EState6', 'VSA_EState7', 'VSA_EState8', 'VSA_EState9', 'FractionCSP3', 'HeavyAtomCount', 'NHOHCount', 'NOCount', 'NumAliphaticCarbocycles', 'NumAliphaticHeterocycles', 'NumAliphaticRings', 'NumAromaticCarbocycles', 'NumAromaticHeterocycles', 'NumAromaticRings', 'NumHAcceptors', 'NumHDonors', 'NumHeteroatoms', 'NumRotatableBonds', 'NumSaturatedCarbocycles', 'NumSaturatedHeterocycles', 'NumSaturatedRings', 'RingCount', 'MolLogP', 'MolMR', 'fr_Al_COO', 'fr_Al_OH', 'fr_Al_OH_noTert', 'fr_ArN', 'fr_Ar_COO', 'fr_Ar_N', 'fr_Ar_NH', 'fr_Ar_OH', 'fr_COO', 'fr_COO2', 'fr_C_O', 'fr_C_O_noCOO', 'fr_C_S', 'fr_HOCCN', 'fr_Imine', 'fr_NH0', 'fr_NH1', 'fr_NH2', 'fr_N_O', 'fr_Ndealkylation1', 'fr_Ndealkylation2', 'fr_Nhpyrrole', 'fr_SH', 'fr_aldehyde', 'fr_alkyl_carbamate', 'fr_alkyl_halide', 'fr_allylic_oxid', 'fr_amide', 'fr_amidine', 'fr_aniline', 'fr_aryl_methyl', 'fr_azide', 'fr_azo', 'fr_barbitur', 'fr_benzene', 'fr_benzodiazepine', 'fr_bicyclic', 'fr_diazo', 'fr_dihydropyridine', 'fr_epoxide', 'fr_ester', 'fr_ether', 'fr_furan', 'fr_guanido', 'fr_halogen', 'fr_hdrzine', 'fr_hdrzone', 'fr_imidazole', 'fr_imide', 'fr_isocyan', 'fr_isothiocyan', 'fr_ketone', 'fr_ketone_Topliss', 'fr_lactam', 'fr_lactone', 'fr_methoxy', 'fr_morpholine', 'fr_nitrile', 'fr_nitro', 'fr_nitro_arom', 'fr_nitro_arom_nonortho', 'fr_nitroso', 'fr_oxazole', 'fr_oxime', 'fr_para_hydroxylation', 'fr_phenol', 'fr_phenol_noOrthoHbond', 'fr_phos_acid', 'fr_phos_ester', 'fr_piperdine', 'fr_piperzine', 'fr_priamide', 'fr_prisulfonamd', 'fr_pyridine', 'fr_quatN', 'fr_sulfide', 'fr_sulfonamd', 'fr_sulfone', 'fr_term_acetylene', 'fr_tetrazole', 'fr_thiazole', 'fr_thiocyan', 'fr_thiophene', 'fr_unbrch_alkane', 'fr_urea'