Instituto Tecnológico de Buenos Aires

Escuela de Ingeniería y Tecnología

Implementación y validación de un modelo de Deep Learning para la clasificación de toxicidad de compuestos de interés farmacéutico

Alumna: Valeria Scardino (Legajo 57277)

Tutor: Claudio Cavasotto

Trabajo final presentado para la obtención del título de Bioingeniero

Buenos Aires

1º Cuatrimestre 2021

Índice

Resumen

Para que un nuevo medicamento llegue finalmente al mercado se requieren más de diez años y una enorme inversión. Muchos de estos recursos se invierten en cientos de compuestos que no superan las primeras etapas de evaluación debido a su toxicidad. Por lo tanto, conocer, o al menos inferir los efectos tóxicos para la salud de estos compuestos es igual de importante que conocer su efectividad.

A raíz de la competencia Tox 21 Data Challenge de 2014, los métodos de aprendizaje automático o Machine Learning cobraron interés para la toxicología computacional. El grupo ganador de la competencia implementó un modelo de Deep Learning basado en redes neuronales artificiales que obtuvo los mejores resultados entre los métodos propuestos. Sin embargo, la métrica de evaluación utilizada en la competencia es la conocida ROC-AUC, cuya principal limitación es no tener en cuenta el desbalance entre las clases. El gran desafío de las bases de datos de toxicología es justamente que se encuentran notoriamente desbalanceadas teniendo, en algunos casos, un 1 % de compuestos tóxicos. A la vez, se debe considerar la importancia de la clasificación incorrecta de la clase tóxica, ya que ésta trae consigo peores consecuencias que la clasificación incorrecta de la clase no tóxica recurrente. En el presente trabajo se propone un modelo de Deep Learning que utiliza la combinación de datos de diversos efectos tóxicos relevantes a la salud para la clasificación conjunta de toxicidad de compuestos de interés farmacéutico. El mismo es evaluado según el F_2 score que se construye a partir de la sensibilidad y la precisión y es una métrica más adecuada al problema que se busca resolver. Los resultados, en general, muestran una sensibilidad alta y una precisión baja. Los resultados altos de sensibilidad demuestran la capacidad potencial de Deep Learning para detectar compuestos tóxicos. Sin embargo, aún es necesario un mayor estudio para mejorar los resultados de la precisión. Utilizar métricas adecuadas demuestra el desafío de introducir modelos de Machine Learning en el pipeline del descubrimiento de fármacos, en particular, para la clasificación de toxicidad.

Capítulo 1

1. Introducción

El rápido avance de la tecnología está permitiendo un conocimiento más profundo de las enfermedades a nivel molecular. Esto genera grandes desafíos para la industria de desarrollo de fármacos que se hace cada vez más compleja y con procesos más extensos. Para que un nuevo medicamento llegue finalmente al mercado se requieren más de diez años y una inversión de dinero que promedia los 2.6 billones de dólares [1]. Encontrar nuevas drogas es una tarea complicada principalmente debido al gran tamaño del espacio químico, que se estima aproximadamente en 10^{60} moléculas [2]. Las nuevas entidades moleculares (NME, New Molecular Entities) son compuestos biológicos novedosos que presentan una actividad prometedora para interactuar contra un blanco terapéutico específico que se considera importante en una enfermedad. De todas maneras, si bien su actividad despierta interés, aún no se conoce con certeza su efectividad ni su seguridad o comportamiento tóxico. Las NME surgen en las etapas iniciales del desarrollo de un nuevo medicamento que duran aproximadamente entre tres a seis años. Gran parte de este tiempo se destina a evaluar cientos e incluso miles de compuestos que no terminan superando estas etapas debido a su toxicidad [3]. Ningún agente terapéutico carece de riesgo y por tanto la identificación del mismo y su relación con el beneficio de administrarlo son aspectos importantes a analizar.

El aumento de datos disponibles en los últimos años tuvo como consecuencia que los métodos computacionales se presenten como una gran herramienta potencial para acelerar y reducir el costo del proceso de descubrimiento de fármacos. La disponibilidad de estructuras tridimensionales (3D) de proteínas de alta calidad permitió utilizar métodos computacionales para analizar las interacciones de complejos proteína-ligando mediante procesos de screening virtual con grandes bibliotecas de moléculas. Al día de hoy, el diseño

de fármacos asistido por computadora ya es una herramienta consolidada en el pipeline de desarrollo de un nuevo medicamento [4]. Estos métodos muestran muchas ventajas frente a los métodos experimentales ya que son efectivos, menos costosos, rápidos y más acordes al cuidado del medio ambiente minimizando las pruebas en animales. Los mismos permiten estudiar el comportamiento de compuestos de interés antes de ser sintetizados químicamente y así seleccionar únicamente a los mejores candidatos para los estudios experimentales.

El proceso de desarrollo de fármacos puede dividirse aproximadamente en cuatro etapas principales: 1) Una etapa de descubrimiento en la que se seleccionan las proteínas blanco de una enfermedad (targets) y se seleccionan moléculas candidatas a interactuar con ese blanco; 2) una etapa en la que se realizan ensayos *in vitro* y estudios pre-clínicos *in vivo*; 3) una etapa de ensayos clínicos en humanos; 4) una última etapa en la que se busca la aprobación de un ente regulatorio como la FDA de Estados Unidos o REACH de la Unión Europea para poder ser comercializado. Cada uno de estos pasos funciona como un filtro para ir seleccionando los compuestos que cumplan con las características buscadas. Durante la primera etapa de descubrimiento se realiza una búsqueda entre aproximadamente 10,000 moléculas que cumplan ciertas propiedades para ser consideradas como fármacos. En total, el desarrollo de un nuevo medicamento puede durar de siete a 15 años. Este proceso se encuentra ilustrado en la figura 1. Existe en la industria una gran necesidad de métodos más rápidos y fiables que complementen a los métodos tradicionales para acelerar estos procesos y reducir los costos [5].

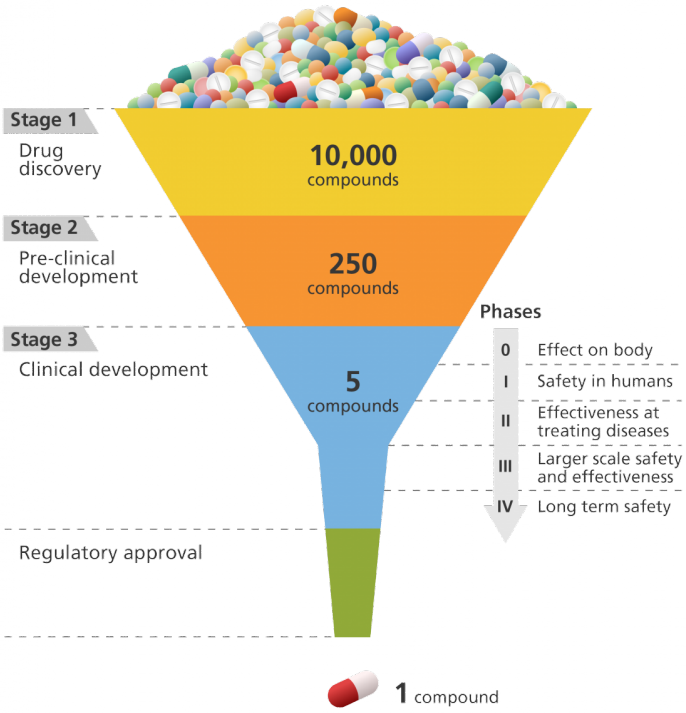


Figura 1: Esquema de las etapas principales en el descubrimiento de fármacos [6].

1.1. Toxicología

La toxicología tiene un rol importante durante todo el proceso de desarrollo de medicamentos especialmente en la evaluación de las NME en etapas tempranas. La toxicidad es la medida que describe el grado en el cual una sustancia o compuesto químico puede generar alguna lesión a la salud [7]. En las últimas décadas las agencias de protección han experimentado una creciente frustración debido al gran número de fracasos en pruebas de toxicidad para proporcionar información relevante para la regulación de compuestos químicos. En el período del 2006 - 2015 sólo un 9.7 % de los compuestos que pasaron de una etapa de descubrimiento llegaron a ser comercializados [8]. Incluso después de ser aprobados, muchos medicamentos son retirados del mercado por presentar riesgos a la

salud. Esto genera una pérdida de confianza en la industria tanto para los pacientes, los profesionales sanitarios, los inversores y las entidades reguladoras [9].

En 2010, la iniciativa de Krewski, D. et al. [10] *"Toxicity Testing in the 21st Century"* enumeró algunos de los grandes desafíos a los que se enfrentan las pruebas de toxicidad de medicamentos y que se buscan resolver actualmente:

- Necesidad de evaluar una gran cantidad de compuestos químicos existentes, muchos de los cuales carecen de datos de toxicidad.
- Reducir el costo y el tiempo necesarios para la evaluación de la seguridad química.
- Evaluar a una gran cantidad de compuestos nuevos y materiales novedosos, como los nanomateriales que se introducen en el mercado cada año.
- Necesidad de evaluar los posibles efectos adversos con respecto a puntos finales críticos en la investigación de un medicamento.
- Minimizar el uso de animales.
- Aumentar la eficacia de las pruebas de toxicidad para colaborar con la toma de decisiones regulatorias.

Dentro de la etapa de estudios preclínicos, la toxicidad de una NME puede estimarse experimentalmente de dos formas principales: mediante ensayos celulares *in vitro* y con ensayos *in vivo* en modelos de roedores. Estos procesos generalmente no son suficientes para garantizar un comportamiento biológico en humanos ya que no tienen en cuenta la complejidad del conjunto e interacciones del organismo [11]. Si se llegan a obtener resultados positivos de los ensayos de eficacia y toxicología *in vitro*, se inician los estudios clínicos de seguridad, farmacodinámica y farmacocinética. A medida que avanzan estos ensayos clínicos en humanos, las NME van progresando a través de distintas evaluaciones toxicológicas que duran de dos semanas a un año [12].

Existen criterios de valoración clínica relacionados con el comportamiento tóxico de una molécula que establecen puntos de corte en el proceso de desarrollo de un nuevo fármaco. A estos criterios se los conoce como endpoints. Actualmente se busca definir estos puntos para intentar identificar tempranamente toxicidades específicas para humanos, especialmente toxicidades para las que los modelos preclínicos comunes no tienen buenos resultados predictivos [11]. Las propiedades ADMET: Absorción, Distribución, Metabolismo, Excreción y Toxicidad, buscan describir el comportamiento químico de los compuestos en el organismo. Las mismas pueden ser determinadas actualmente mediante diversos métodos *in silico*. A su vez, algunas de las propiedades ADMET se relacionan con indicadores fisicoquímicos simples como la solubilidad (que influye en la absorción de las drogas), la lipofilicidad (responsable del pasaje a través de membranas celulares y barreras lipídicas) y la permeabilidad, entre otros. Conocerlos permite predecir los endpoints que se relacionan con distintos procesos tóxicos. Por ejemplo, procesos de toxicidad cardíaca, efectos hepáticos, pasaje a través de la barrera hematoencefálica, carcinogenicidad o mutagenicidad. De todas maneras, determinar estos endpoints es una tarea compleja ya que suelen estar relacionados con las distintas especies o con la vía de administración de los fármacos entre otras cosas [5].

1.2. Machine Learning en toxicología

Los algoritmos de aprendizaje automático o Machine Learning (ML) son unos de los métodos más comunes dentro del área de la Inteligencia Artificial (IA). Los mismos tienen la capacidad de identificar características ocultas dentro de grandes conjuntos de datos [13]. Dentro de los métodos de ML se distinguen dos grandes grupos: métodos de aprendizaje supervisado y no supervisado. Los primeros permiten mapear automáticamente un conjunto de entradas a un conjunto de salidas a partir de datos anotados. Los segundos sirven para aprender directamente relaciones subyacentes en un conjunto dado de datos.

Algunos ejemplos de estos modelos son los algoritmos de k-Nearest Neighbours (kNN), Support vector machine (SVM), Random forest (RF) y algoritmos basados en redes neuronales artificiales (NN, Neural Networks). A su vez, Deep Learning (DL) es un campo dentro de ML (figura 2) que incluye métodos de muchos parámetros computables basados en varias capas de redes neuronales artificiales.

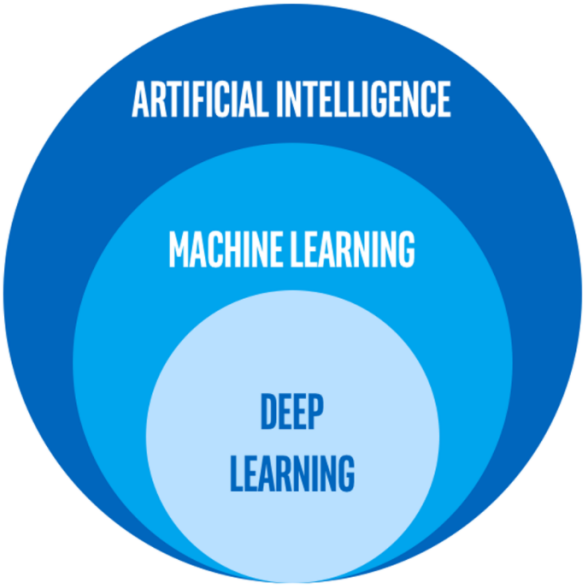


Figura 2: Relación entre métodos de Inteligencia Artificial

Recientemente ha habido un gran interés en el uso de métodos de ML en el proceso de descubrimiento de fármacos [14]. Una de las grandes áreas en las que se implementan es en la técnica de Virtual Screening (VS), en donde se busca mejorar el reconocimiento de moléculas que interaccionan contra una proteína blanco específica. En la investigación de Lenselink et al. de 2017 [15], por ejemplo, se evaluó el desempeño de modelos de ML basados en redes neuronales artificiales para clasificar compuestos como activos o inactivos para interactuar frente a un conjunto de varias proteínas. A la vez, debido a la disponibilidad de una gran cantidad de datos relacionados a distintas toxicidades, los métodos de ML están convirtiéndose también en una herramienta importante en la

toxicología computacional [16]. Entidades de aprobación de medicamentos como la FDA en Estados Unidos y REACH en la Unión Europea han promovido su desarrollo en las etapas tempranas de la investigación de nuevos medicamentos [5, 17]. También ha habido últimamente una mayor colaboración entre la industria de descubrimiento de fármacos y empresas o grupos de investigación dedicados a la inteligencia artificial [18].

1.2.1. Tox 21 Data Challenge

En 2014, el Centro Nacional de Ciencias Traslacionales Avanzadas (NCATS) de los Institutos Nacionales de Salud (NIH) lanzó la competencia Tox21 Data Challenge. El objetivo fue unificar el análisis de datos mediante modelos computacionales por parte de investigadores independientes para revelar qué tan bien pueden predecir una serie de efectos tóxicos utilizando solo datos de su estructura química [19]. Los investigadores partían de una base de datos de más de 10,000 moléculas anotadas para 12 efectos tóxicos diferentes. Estos incluyen efectos del receptor nuclear y efectos de respuesta al estrés, que son respuestas tóxicas muy relevantes en la salud, ya que los primeros pueden afectar al sistema endocrino y los segundos pueden producir daños en el hígado o incluso cáncer [20]. Según lo establecido por la entidad organizadora NCATS, los modelos de ML presentaron alta calidad lo que genera "confianza para aplicarlos en la identificación de los productos químicos con mayor potencial de toxicidad y deben seguir siendo estudiados en futuras investigaciones" [21].

El grupo de Mayr et al. fue el ganador de esta competencia implementando un modelo de DL al que llamaron DeepTox [22]. El mismo mostró el rendimiento más alto en promedio entre los 12 efectos tóxicos medidos, sobresaliendo sobre otros métodos como SVM y RF. A partir de este resultado, la utilización de modelos de DL ha ido creciendo en la toxicología computacional. Un ejemplo es la investigación de Wenzel et al. [23] en la que se utiliza DL para la predicción de propiedades ADMET de compuestos químicos. También se ha

evaluado, por ejemplo, el rendimiento de distintas arquitecturas de DL para la predicción de toxicidad hepática [24]. A estos trabajos se les suman otras investigaciones similares en las que se busca evaluar la efectividad de estos modelos para estimar distintas toxicidades [25, 26, 27, 28]. En general, los resultados sugieren una gran capacidad de estos modelos para seguir influenciando el campo de la toxicología en el futuro.

1.3. Deep Learning

Deep Learning comprende uno de los métodos de ML más comunes que se basa en varias capas de redes neuronales. Al día de hoy ya ha impactado en diversos campos de procesamiento de información como en el procesamiento de imágenes, reconocimiento de voz, procesamiento de lenguaje natural y textos e incluso en aplicaciones médicas [29]. La cantidad de arquitecturas y algoritmos distintos que se utilizan en DL es amplia y variada. Los más comunes incluyen a las redes de perceptrones de múltiples capas (MLP, Multi Layer Perceptron) que se muestran en la figura 3, redes neuronales recurrentes (RNN, Recurrent neural Networks) y redes convolucionales (CNN, Convolutional Neural Networks). Dentro de cada uno de estos grupos existen a su vez distintas arquitecturas. Cuándo usar una u otra dependerá del problema que se quiera resolver. Los MLP que se encuentran representados en la figura 3 comprenden una arquitectura básica que ha probado tener buenos resultados para diversos problemas de clasificación y regresión [30]. Las CNN fueron diseñadas especialmente para el procesamiento de imágenes y son muy utilizadas en ese campo. Las RNN fueron diseñadas para la predicción de secuencias donde se parte de una observación de entrada que es mapeada a una secuencia de múltiples pasos a la salida.

El modelo DeepTox del grupo ganador del Tox21 Data Challenge tiene como base un modelo MLP. En otras investigaciones, como la de Fernandez et al. [25], se utilizaron modelos CNN implementados directamente sobre una representación 2D de la estructura

molecular pero no lograron, hasta el momento, superar el desempeño de modelos MLP. En este trabajo se implementará un modelo MLP cuyo funcionamiento se introduce a continuación.

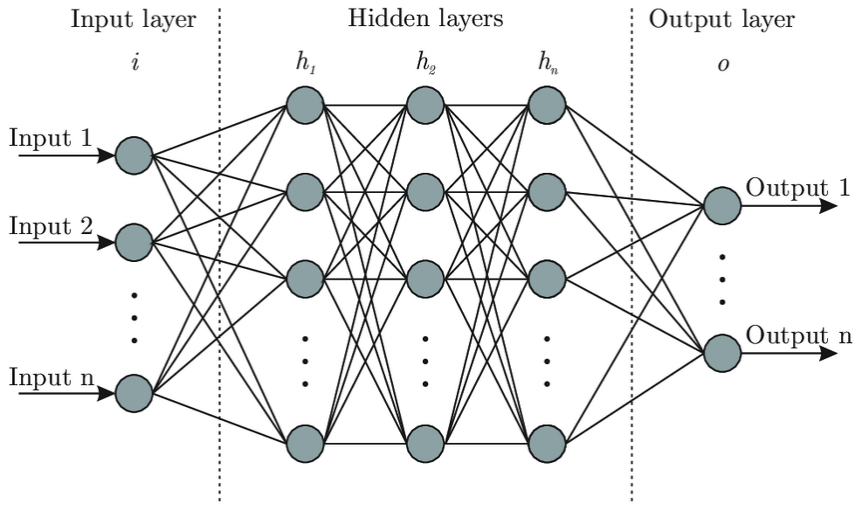


Figura 3: Representación esquemática de un Multi Layer Perceptron (MLP).

Cuando se utilizan NN para el aprendizaje supervisado, durante la etapa de entrenamiento se le proporciona al modelo un conjunto de datos mediante los cuales el modelo produce una salida en forma de vector de probabilidades de pertenecer a cada categoría dentro del conjunto de datos. Esta salida depende de un conjunto de parámetros internos y de una función de activación. Para esto se parte de una matriz de datos de entrada X donde cada fila representa una muestra y cada columna representa las características elegidas para describir esa muestra. Luego, la NN calcula la salida de una capa oculta H mediante la ecuación:

$$H = f(X.W + b)$$

donde f es la función de activación, W es el vector de pesos de la primera capa y b es el sesgo utilizado para propagar la entrada X y calcular H. Este resultado pasa entonces por

una capa de salida para dar la matriz final Y:

$$Y = f'(H.W' + b')$$

La salida Y contiene en cada fila los valores de predicción obtenidos para cada muestra de la entrada.

Para el aprendizaje se calcula iterativamente una función de costo $L = L$(W,b) que mide el error entre los datos de salida del modelo y los valores reales. Mediante un vector gradiente de esta función de costo el método conoce en qué magnitud el error disminuiría al incrementar los parámetros internos en una pequeña cantidad. De esta manera el optimizador busca minimizar la función de costo para que el modelo se comporte de la forma más parecida posible a la deseada. Así en cada iteración el modelo va modificando estos parámetros o pesos de manera que ese error se reduzca hasta alcanzar, en el mejor de los casos, el mínimo de la función de costo. A la salida del modelo se espera que la probabilidad del vector de salida para la clase correcta sea la mayor, lo cual indica un aprendizaje del modelo a partir de los datos de entrenamiento. De todas formas existen mínimos locales y llanuras o plateaus en la función de costo por lo que se emplean distintas estrategias según el optimizador que se este utilizando para evitar converger en estos puntos.

Un MLP, como el de la figura 3 es básicamente una NN con más de una capa oculta. En este caso la primera capa H_1 se calcula como:

$$H_1 = f_0(X.W_0 + b_0)$$

siendo f_0, W_0 y b_0 la función de activación, la matriz de pesos y el sesgo de la primera capa respectivamente. Las siguientes capas ocultas se calculan de la misma manera, con diferentes conjuntos de parámetros y funciones de activación:

$$H_i = f_{i-1}(H_{i-1}.W_{i-1} + b_{i-1})$$

Se obtienen así múltiples niveles de representación al combinar módulos simples pero no lineales, de los cuales cada uno transforma la representación de los datos en un nivel dado en una representación en un nivel más abstracto. Finalmente la salida al final de la última capa de un modelo MLP puede representarse como:

$$Y = f_N(H_N.W_N + b_N)$$

siendo N el número total de capas ocultas.

Los hiperparámetros son las variables que no forman parte de los parámetros internos o pesos de las NN y cuyo valor no puede ser estimado directamente a partir de los datos. Son especificados desde un principio por el programador mediante una búsqueda para ajustar los algoritmos de aprendizaje. Estos pueden dividirse en hiperparámetros de control de entrenamiento y de arquitectura de la red. Algunos ejemplos de los primeros son las distintas funciones de activación o tasas de aprendizaje empleadas. Los segundos incluyen, por ejemplo, la cantidad de capas o la cantidad de neuronas por capa. En general, a partir de los datos anotados disponibles se separa un grupo de entrenamiento, uno de validación y uno de prueba. El grupo designado para la validación se utiliza para monitorear el desempeño de modelos con distintos hiperparámetros en datos que no se emplearon directamente para optimizar los parámetros internos. Lo que se espera es un modelo que permita una adecuada generalización, es decir, que se comporte de manera similar tanto para el entrenamiento como para la validación [29]. Finalmente, el conjunto de datos de prueba se utiliza para evaluar el desempeño del modelo seleccionado en datos que nunca han sido vistos por el algoritmo.

Un problema inherente de MLP es el requerimiento de una gran cantidad de datos

disponibles que contrarresten a la alta cantidad de parámetros que requieren los modelos. Para mejorar el entrenamiento, se requiere, en la medida de lo posible, de un gran número de datos disponibles, integrados de múltiples fuentes y recopilados en múltiples marcos de tiempo [30]. Utilizar DL en grandes conjuntos de datos requiere un alto poder de cómputo debido a la gran cantidad de cálculos que se realizan durante el entrenamiento. Por este motivo las unidades de procesador gráfico (GPU) se han convertido en herramientas esenciales en este área.

1.3.1. Evaluación de modelos

A la hora de evaluar el comportamiento de un modelo de DL es necesario seleccionar una métrica de evaluación adecuada para el problema que se aborda. En una tarea de clasificación, por ejemplo, en algunos casos será más importante priorizar la clasificación correcta de la clase verdadera y en otros la clasificación de la clase falsa. Usar una u otra métrica depende del problema, pero comprender las ventajas y desventajas entre las mismas ayuda a tomar una decisión correcta. Algunas de las métricas más utilizadas para resolver problemas de clasificación se abordan a continuación:

Matriz de confusión

En problemas de clasificación, a la salida de un modelo de DL se tienen predicciones: una predicción verdadera es cuando se obtiene el valor real y una predicción falsa es cuando el mismo no se obtiene. La matriz de confusión es una matriz que permite visualizar directamente y de forma clara los resultados de la clasificación con respecto a los valores verdaderos. Esto permite una mejor comprensión de cómo está funcionando el modelo. A partir de ella se derivan otras métricas importantes para la evaluación. Se define de la siguiente manera:

	N'	**P'**
N	VN	FP
P	FN	VP

Tabla 1: Matriz de confusión. N (Negativo) y P(Positivo) corresponden a la clase real de las muestras, mientras que N' y P' corresponden a la clasificación otorgada por el método.

En este caso Verdadero Negativo (VN) corresponde a las muestras que son clasificadas como negativas y son realmente negativas, Falso Negativo (FN) a las clasificadas como negativas pero que en realidad son positivas, Falso Positivo (FP) a las muestras clasificadas como positivas que en realidad son negativas, y Verdadero Positivo (VP) a las muestras clasificadas como positivas que realmente son positivas.

Accuracy

Esta métrica mide cuántas observaciones, tanto positivas como negativas, se clasificaron correctamente.

$$ACC = \frac{VP + VN}{VP + FP + VN + FN} \tag{1}$$

Es una de las métricas más utilizadas ya que permite obtener una idea general de cómo está funcionando el modelo. Sin embargo, no debe utilizarse en problemas con desbalance de clases ya que en estos casos es fácil obtener una puntuación alta de

Accuracy simplemente clasificando todas las observaciones como la clase mayoritaria.

Precisión

La precisión, también llamada valor predictivo positivo (PPV, Positive Predictive Value), es la fracción de instancias reales o relevantes entre las instancias positivas recuperadas. Se calcula como la relación:

$$Precisión = \frac{VP}{VP + FP} \tag{2}$$

La precisión es intuitivamente la capacidad del clasificador de no etiquetar como positiva una muestra que es negativa. Será mayor cuanto menor sea la tasa de FP.

Sensibilidad

La sensibilidad, también llamada Recall o tasa de verdaderos positivos (TPR, True Positive Rate), es la fracción del total de instancias relevantes que son recuperadas. Puede obtenerse como:

$$Sensibilidad = \frac{VP}{VP + FN} \tag{3}$$

Es la capacidad del clasificador de no etiquetar como negativa una muestra que es positiva. Será mayor cuanto menor sea la tasa de FN. En general suele haber un compromiso entre ésta tasa y la PPV.

ROC-AUC

ROC-AUC significa área bajo la curva ROC (Reciever Operating Characteristic). Esta curva es un gráfico que visualiza la relación entre la sensibilidad y la tasa de falsos positivos (FPR, False Positive Rate). Para distintos umbrales de separación de clases, se calculan estas tasas y se trazan en un gráfico. La métrica asociada es el área debajo de la curva la cual da información de la relación entre TPR y FPR de manera cuantitativa y por ende muestra qué tan bueno es el modelo para clasificar las predicciones. Éste área toma un valor de 0.5 para clasificadores aleatorios o random y un valor de 1 para clasificadores perfectos. Es una métrica robusta y muy utilizada para comparar distintos modelos. Tampoco es adecuada para conjuntos de datos muy desequilibrados ya que la tasa de falsos positivos en estos casos se reduce debido a una gran cantidad de verdaderos negativos clasificados.

PR-AUC

PR es una curva que combina precisión y sensibilidad en una sola visualización. Ésta métrica calcula el área bajo la curva PR. Al igual que en la curva ROC, para cada umbral se calculan las tasas PPV y TPR y se traza un gráfico. Cuanto más alta sea la curva en el eje y, mejor será el rendimiento del modelo. Esta métrica es más adecuada para conjuntos de datos desbalanceados ya que se centra principalmente en la clasificación de la clase positiva (PPV y TPR) y por ende se preocupa menos por la clase negativa más frecuente.

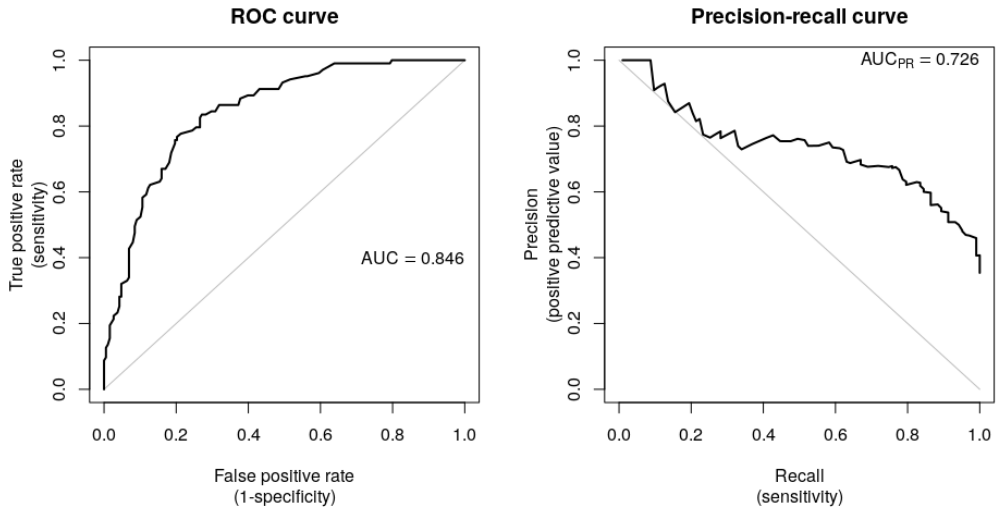


Figura 4: Área bajo las curvas ROC y PR. Un valor del 50 % indica que no hubo aprendizaje de los datos.

F_β score

El F_β score es una métrica que combina precisión y sensibilidad al calcular la media armónica entre ambas. El β es un parámetro que pondera más una u otra métrica según la importancia que se les quiera dar. En caso de elegir $\beta = 1$ la precisión y el recall se ponderan de igual manera. En este caso un F_1 cercano a uno indica que se están minimizando tanto los falsos positivos como los falsos negativos y un valor cercano a 0 indica que el modelo no es confiable para aprender los datos y generalizar. Un valor de β entre cero y uno pondera más la precisión y un valor mayor que uno pondera más el recall. En este último caso se busca minimizar prioritariamente los falsos negativos. Se puede ajustar el umbral seleccionado para la separación de clases con el fin de optimizar la métrica. Es muy útil cuando la clasificación de la clase positiva es más importante para el problema.

Se define como:

$$F_\beta score = (1 + \beta^2)\frac{precision * recall}{\beta^2 * precision + recall} \tag{4}$$

1.3.2. Sobre-entrenamiento

Un modelo de ML será exitoso en la medida en que logre generalizar su aprendizaje, es decir, si logra desempeñarse adecuadamente en datos nunca antes vistos. Éste es uno de los desafíos centrales en los problemas de ML y en particular DL [30]. El sobre-ajuste, conocido como overfitting, ocurre cuando hay una gran diferencia entre el error del entrenamiento y el error de prueba final. En este caso el modelo aprende muy rigurosamente las características del conjunto de datos de entrenamiento y por ende no puede generalizar ese aprendizaje hacia los datos de prueba. Está relacionado con la cantidad de parámetros internos del modelo (o pesos) y la cantidad de datos disponibles para el entrenamiento. Se han desarrollado diversas técnicas de regularización para minimizar este problema.

La mejor solución para prevenirlo es obtener más datos [29]. Muchas veces eso no es posible y en ese caso se busca regular la cantidad de características que el modelo puede aprender. La idea es que la red se enfoque únicamente en el aprendizaje de patrones más importantes y de esa manera pueda generalizar bien. En este sentido, una manera simple de prevenir el sobre-ajuste es modificando la arquitectura de la red de forma tal de reducir la cantidad de pesos entrenables en el modelo. Muchos pesos le otorgan más capacidad de memorizar y por lo tanto se genera un mapeo ideal entre los datos de entrenamiento y sus clases y menos capacidad de generalización. Ésta relación entre el tamaño de la red neuronal y la capacidad de aprendizaje se ilustra en la figura 5.

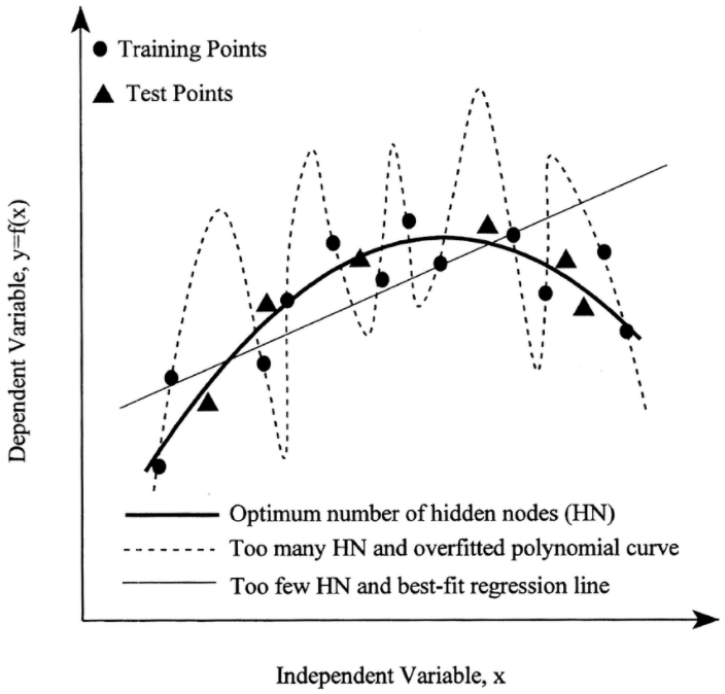


Figura 5: Efectos del tamaño de la red neuronal en la capacidad de generalización. Imágen obtenida de [29].

El Dropout es una de las formas más habituales de regularizar redes neuronales [31]. Ésta técnica elimina algunas unidades de las redes agregando ruido y de esta manera previniendo el sobre-ajuste. La tasa de Dropout es la fracción de salidas de una capa que son puestas en cero para agregar ruido y suele tener valores entre 0.2 y 0.5. Otra técnica de regularización muy usada es agregar una regularización a los pesos. La misma busca evitar el sobre-ajuste restringiendo la complejidad del modelo forzando a los pesos a tomar valores pequeños. De esta manera se obtiene una distribución de pesos más regular y un modelo más simple enfocado en aprender los patrones más importantes. Para esto se agrega a la función de costo una penalización si se tienen pesos muy grandes. Esta penalización puede darse de tres maneras: agregando un costo proporcional a la suma del valor absoluto de los pesos (norma $l1$ de los pesos), agregando un costo proporcional a la

suma del cuadrado de los pesos (norma *l2* de los pesos) o con una combinación de ambos (normas *l1* y *l2*).

Al entrenar un modelo de DL, el optimizador actualiza los pesos en cada iteración para que se ajusten mejor a los datos de entrenamiento. Hasta cierto punto, esto mejora el rendimiento sobre los datos de validación que están fuera del conjunto de entrenamiento. Sin embargo, mejorar el ajuste más allá de ese punto produce overfitting. Para evitar este problema se usa una técnica de detención temprana del entrenamiento llamada Early Stopping.

1.4. Problemática a tratar y objetivos

Los modelos de DL han mostrado recientemente un potencial para ser utilizados en la clasificación de compuestos tóxicos en las etapas tempranas de la investigación de un nuevo medicamento. Sin embargo, aún es necesaria una mayor comprensión de las ventajas y aplicabilidad de DL ya que también ha habido resultados de otras investigaciones en las que no han sobresalido frente a otros métodos de ML como RF [32, 26, 27]. Además, debido a que es un área de aplicación reciente, aún se requieren mejoras significativas para la predicción cualitativa de toxicidad de compuestos químicos mediante DL.

Los conjuntos de datos de toxicología presentan un desafío para los métodos de aprendizaje automático al ser altamente desbalanceados. El impacto de este desbalance así como la importancia de clasificar de manera correcta a la clase positiva o tóxica no ha sido aún estudiado de manera rigurosa. Al momento se ha comparado el rendimiento de distintos modelos de clasificación utilizando métricas generales como la AUC-ROC o el valor RMSD [22, 25, 26, 27, 28] que si bien dan una primera idea de la performance, no son las más adecuadas para este problema.

En el presente trabajo se implementará un modelo MLP con el objetivo principal

de evaluar DL en toxicología usando una métrica que tenga en cuenta el desbalance de los datos y la importancia de clasificar mal una de las clases. Como primer paso en la evaluación de NME alcanza con conocer si el compuesto es tóxico o no tóxico y no es necesario obtener un valor exacto de su toxicidad, por lo tanto se utilizará un método de clasificación binaria. A su vez, como objetivos más específicos se espera comparar el rendimiento del entrenamiento singular frente al entrenamiento multi-tarea, que se describirán más adelante, teniendo en cuenta la correlación entre los datos de las distintas tareas. También se espera conocer qué características de los compuestos son más importantes para modelos de DL al clasificar su toxicidad. Para la implementación de DL, será necesario analizar los distintos factores que entran en juego en los problemas de clasificación como la arquitectura, los hiperparámetros y los descriptores químicos empleados entre otras cosas.

1.5. Estructura del trabajo

En el **capítulo 2** se desarrollan los métodos utilizados para la clasificación de toxicidad de compuestos químicos. Se presenta una descripción del dataset y de las herramientas de visualización y análisis molecular, se introducen los descriptores moleculares empleados, las métricas y función de costo que se utilizarán para evaluar los modelos de DL y los aspectos teóricos y herramientas empleados para la implementación de estos modelos.

En el **capítulo 3** se describen los criterios de diseño, construcción y optimización del modelo de DL. También se realiza un análisis del nivel de importancia de los descriptores moleculares y se presentan y analizan los resultados obtenidos.

En el **capítulo 4** se exponen las conclusiones obtenidas a partir de los resultados y el nivel de cumplimiento de los objetivos del trabajo.

En el **capítulo 5** se proponen mejoras y trabajo a realizar en el futuro para la

optimización de DL en la clasificación de toxicidad y su introducción al pipeline de descubrimiento de fármacos.

Capítulo 2

2. Métodos

2.1. Tox 21 dataset

Actualmente existen diversas bibliotecas de compuestos anotados según su toxicidad tanto públicas como privadas. Como se estableció previamente, contar con datos robustos y de una calidad adecuada es un requisito para el buen funcionamiento de los modelos de DL. En el presente trabajo se utilizaron los mismos datos de compuestos y anotaciones de toxicidad otorgados por la competencia Tox 21 Data Challenge para el entrenamiento, validación y evaluación final de los modelos de ML.

La base de datos construida por el programa Tox21 [19] es una de fuentes de datos más utilizada en toxicología. La misma fue construida a partir de la técnica de quantitative high-throughput screening (qHTS) donde una librería de alrededor de 12,000 compuestos químicos y drogas ambientales se midieron experimentalmente para 12 efectos tóxicos diferentes. Esta fue otorgada por la competencia para el entrenamiento de los modelos. Para la validación, se utilizó un subconjunto de datos de la biblioteca LOPAC_{1280} (Biblioteca de compuestos farmacológicamente activos) [33] que se llamó Leaderboard Set. Para la evaluación y puntuación final, se evaluó para los 12 ensayos un nuevo conjunto de compuestos proporcionados por la Agencia de Protección Ambiental Estadounidense (EPA) para los cuales no había datos experimentales disponibles en el momento. Este nuevo conjunto de datos junto con el resto de los datos de LOPAC se utilizó para evaluar las presentaciones finales de la competencia [19].

Los 12 ensayos de toxicidad se seleccionaron según la calidad de los datos, la tasa de actividad y la relevancia toxicológica [19]. Pueden dividirse en Nuclear Receptor Effects (NR) y Stress Response Effects (SR). Ambos grupos incluyen respuestas tóxicas

muy relevantes para la salud humana pudiendo generar afecciones en el sistema endócrino y diversas enfermedades. Los efectos del receptor nuclear (NR) incluyen el dominio de unión a ligando del receptor de estrógenos alpha (NR-ER-LBD), el receptor de estrógeno completo (NR-ER), enzima aromatasa (NR-Aromatasa), receptor de hidrocarburo de arilo (NR-AhR), receptor de andrógenos completo (NR-AR), receptor de andrógenos LBD (NR-AR-LBD) y receptor γ activado por proliferador de peroxisoma (NR-PPAR-γ). Los efectos de la respuesta al estrés (SR) incluyen el factor nuclear de respuesta antioxidante (SR-ARE), el SR-ARE que expresa células renales embrionarias humanas ATAD5 (SR-ATAD5), el elemento de respuesta al factor de choque térmico (SR-HSE), el potencial de membrana mitocondrial (SR-MMP) y la vía de señalización de p53 (SR-p53). El receptor de hidrocarburo de arilo por ejemplo, está involucrado en la regulación del sistema inmune y en la diferenciación celular. Los efectos de respuesta al estrés se encuentran involucrados en enfermedades como el cáncer, diabetes, daños en el hígado, enfermedades neurodegenerativas y cardiovasculares [34].

La cantidad de compuestos activos e inactivos para las 12 distintas tareas se muestran en las tablas 2, 3 y 4.

	NR-AhR	NR-AR	NR-AR-LBD	NR-Aromatase	NR-ER	NR-ER-LBD	NR-PPAR-γ	SR-ARE	SR-ATAD5	SR-HSE	SR-MMP	SR-p53
Activos	946	380	303	357	933	446	220	1092	333	427	1140	529
Inactivos	7165	8916	8237	6817	6714	8246	7906	6032	8693	7670	6130	8044
Porcentaje de Activos	11	4	3	4	12	5	2	15	3	5	15	6
Total	8111	9296	8540	7174	7647	8692	8126	7124	9026	8097	7270	8573

Tabla 2: Distribución de clases activa (tóxica) e inactiva (no tóxica) en el set de entrenamiento dado por la competencia.

	NR-AhR	NR-AR	NR-AR-LBD	NR-Aromatase	NR-ER	NR-ER-LBD	NR-PPAR-γ	SR-ARE	SR-ATAD5	SR-HSE	SR-MMP	SR-p53
Activos	30	3	4	18	27	10	15	46	25	10	37	28
Inactivos	239	286	248	196	237	274	249	186	246	254	198	239
Porcentaje de Activos	11	1	1	8	10	3	5	19	9	3	15	10
Total	269	289	252	214	264	284	264	232	271	264	235	267

Tabla 3: Distribución de clases activa (tóxica) e inactiva (no tóxica) en el set de validación Leaderboard dado por la competencia.

	NR-AhR	NR-AR	NR-AR-LBD	NR-Aromatase	NR-ER	NR-ER-LBD	NR-PPAR-γ	SR-ARE	SR-ATAD5	SR-HSE	SR-MMP	SR-p53
Activos	73	12	8	39	51	20	31	93	38	22	60	40
Inactivos	534	571	571	487	462	577	571	459	581	585	481	573
Porcentaje de Activos	12	2	1	7	9	3	5	16	6	3	11	6
Total	607	583	579	526	513	597	602	552	619	607	541	613

Tabla 4: Distribución de clases activa (tóxica) e inactiva (no tóxica) en el set de prueba final dado por la competencia.

Todos los conjuntos de datos con sus anotaciones son de acceso público actualmente y se encuentran disponibles en la página web de la competencia [21]. Los mismos se dieron en formato SDF (Standard Database Format) y en formato SMILES (especificación de introducción lineal molecular simplificada). El set de entrenamiento consiste de alrededor de 11 mil compuestos, el set Leaderboard para la validación de 293 compuestos y por último el set de evaluación que se utilizó para obtener los resultados finales consiste de aproximadamente 700 compuestos. No todos fueron testeados para los 12 efectos tóxicos, teniendo en promedio 10,000 compuestos por cada tarea.

2.2. Preparación de los datos

Para este trabajo se construyó un archivo SDF a partir de las estructuras 2D de las moléculas descargadas de la página web de la competencia y otro archivo csv con sus

correspondientes actividades para cada respuesta tóxica. Es importante destacar que no todos los compuestos fueron medidos para todos los efectos tóxicos o tasks y por ende existen datos perdidos o "missing labels" en el archivo de las anotaciones. Estos datos perdidos se guardaron con una etiqueta NaN (not a number).

Se utilizó el software ICM de MolSoft [35] para la visualización y análisis de los compuestos químicos del dataset (figura 6) con el objetivo de aumentar la calidad de los datos. Se vio que muchos de los compuestos presentan agregados o mezclas de otras moléculas que no están unidas con enlaces covalentes. Estos agregados deben ser separados y retirados del compuesto antes de calcular los descriptores moleculares. Para esto se utilizó la librería Open Babel [36]. La misma posee funciones que permiten convertir estructuras moleculares entre distintos formatos introduciéndoles ciertas modificaciones, por ejemplo, quedándose con el fragmento unido más largo y eliminando así las sales presentes.

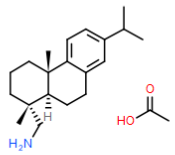
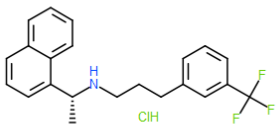
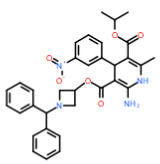
mol	Molecule Name	Formula	FW	DSSTox CID	Active
	NCGC00255644-01	C22H35NO2	345.5188 (60.0520+285.4668)	27102	0
	NCGC00181002-01	C22H23ClF3N	393.8729 (357.4120+36.4609)	26792	0
	NCGC00167436-01	C33H34N4O6	582.6463	120	0

Figura 6: Visualización de estructuras 2D de compuestos del conjunto NR-AhR de la base de datos Tox 21 utilizando ICM.

Otro aspecto importante a tener en cuenta es el estado de protonación. Considerar a las moléculas en su estado de protonación a pH fisiológico es de gran importancia previo al cálculo de los descriptores moleculares ya que estos átomos suelen estar involucrados en interacciones no covalentes como puentes de hidrógeno con otros compuestos, lo que influye en su actividad tóxica. Las bibliotecas de moléculas suelen no incluir los hidrógenos con el fin de optimizar el espacio disponible. El agregado de átomos de hidrógeno a pH fisiológico (7.4) se realizó también con la librería Open Babel. A modo de ejemplo se muestra en la figura 7 una de las moléculas de la librería en su forma cruda y después de prepararla con Open Babel.

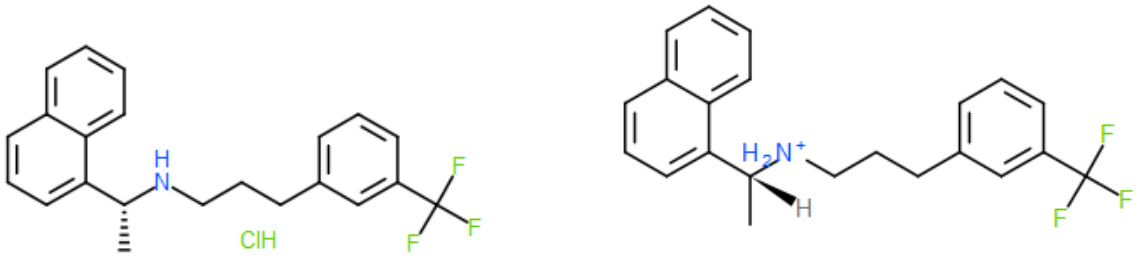


Figura 7: Preparación de compuestos químicos. **izq.:** Molécula cruda obtenida de la base datos. **der.:** Molécula preparada con Open Babel con las sales eliminadas y los hidrógenos agregados a pH fisiológico.

2.3. Multi-Task Learning

Si bien no todos los compuestos fueron medidos para los 12 efectos tóxicos, la mayoría fue medido para más de 10 de ellos. Por este motivo, se puede aplicar un método de aprendizaje de los datos multi-tarea (MTL, Multi Task Learning). En este caso un único compuesto presenta etiquetas para varios tasks. Se ha demostrado en previas investigaciones que los modelos de ML pueden mejorar la performance al entrenarlos con más de

un task a la vez [23]. Especialmente, se vio que el desempeño mejora aún más cuando las anotaciones de las distintas tareas se encuentran correlacionadas. Es un área activamente utilizada y estudiada ya que reduce la cantidad de datos necesarios para aprender nuevos conceptos [37]. A la vez, se cree que MTL refleja el proceso de aprendizaje de los seres humanos con mayor precisión que el aprendizaje de una sola tarea ya que la integración del conocimiento de distintos dominios es una capacidad central de la inteligencia humana [38].

Los métodos existentes de MTL generalmente se han dividido en dos grupos: de entrenamiento conjunto o de entrenamiento alterno. Ambos se diferencian en si se comparten o no los pesos asociados entre las distintas tareas. El entrenamiento conjunto consiste en el uso compartido de parámetros. Se comparten en este caso los pesos de modelos entre múltiples tareas, de modo que cada peso esté entrenado para minimizar conjuntamente múltiples funciones de costo. En el entrenamiento alterno, el gradiente de la función de costo se calcula, en un primer paso, para una única unidad de salida. Superado un cierto número de iteraciones, el optimizador cambia hacia la siguiente unidad de salida, partiendo de los pesos optimizados en el paso anterior. En este trabajo se implementó un modelo de MTL conjunto donde durante el entrenamiento todas las unidades de salida son entrenadas al mismo tiempo y el gradiente se basa en una unidad de salida conjunta global, que es generada por el promedio de las funciones de costo de todas las unidades de salida individuales [38].

Para un mejor análisis y comprensión de los datos disponibles, se calculó el coeficiente absoluto de correlación entre las 12 tareas según la ecuación:

$$r = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2 \sum_{i=1}^{N}(y_i - \bar{y})^2}}$$

donde N es el número de moléculas medidas para los dos tasks evaluados.

Se vio que los receptores NR-ER y NR-ER-LBD presentan un coeficiente absoluto de correlación de 0.60 al igual que los receptores NR-AR y NR-AR-LBD. Los siguientes tasks más correlacionados son el SR-MMP y el SR-ARE que muestran un coeficiente de 0.48. El resto de los tasks entre sí presentan coeficientes menores a 0.40. A partir de estos datos, se probará la posibilidad de entrenar en conjunto los 12 tasks o distintas agrupaciones manteniendo juntos a los más correlacionados. La figura 8 se obtuvo con la librería de Python Seaborn y muestra gráficamente las correlaciones entre las distintas tareas. Se calculó utilizando las anotaciones de los conjuntos de entrenamiento y Leaderboard otorgados por la competencia.

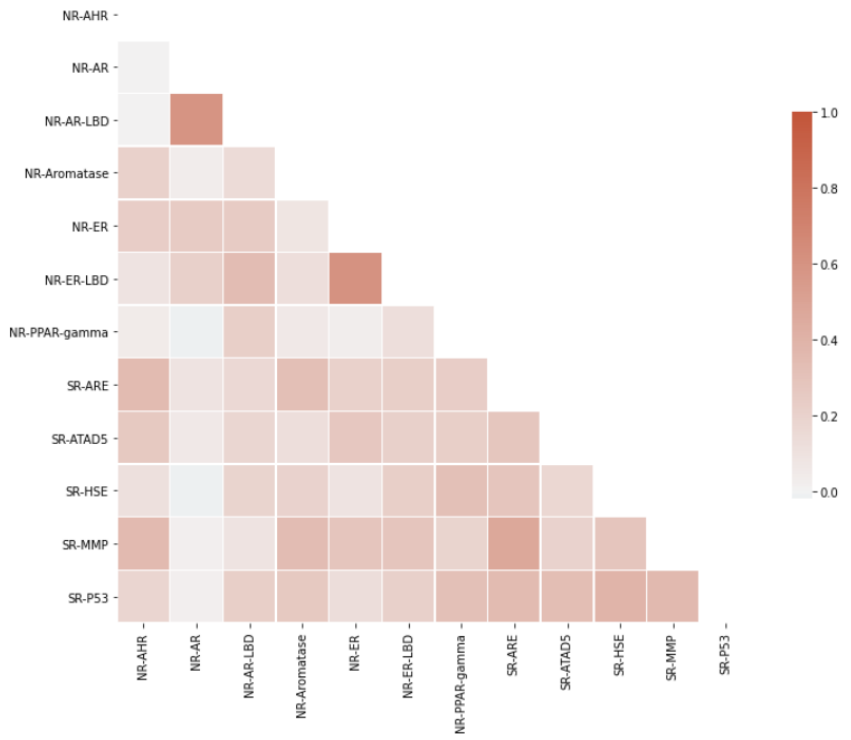


Figura 8: Coeficiente de correlación entre las distintas tareas del Tox21 Data Challenge.

2.4. Descriptores moleculares

La selección y el cálculo de los descriptores moleculares es uno de los primeros pasos para la construcción de un modelo de DL para la predicción de toxicidad. Estos descriptores son características numéricas que permiten representar cuantitativamente la estructura de las moléculas y que servirán de input para los modelos [16]. Ejemplos de ellos son el número de átomos, presencia de alguna subestructura tóxica conocida, peso molecular, carga eléctrica, propiedades atómicas locales como cargas parciales o área de la superficie entre muchos otros. Se calculan mediante algoritmos que reconocen características simples en las estructuras o fórmula molecular, y permiten distinguir a las moléculas entre sí. Al día de hoy se utilizan como input para diversos modelos computacionales, por ejemplo, para correlacionar la relación estructura-actividad (QSAR) o para predecir las propiedades ADMET de las moléculas.

Existen dos tipos principales de descriptores. Una forma es utilizar características numéricas que se calculan a partir de las propiedades fisicoquímicas o topológicas de los compuestos. Estos descriptores pueden ser unidimensionales (1D), bidimensionales (2D) o tridimensionales (3D) según el nivel de representación de la estructura molecular utilizado para calcular el descriptor. Los descriptores 1D representan información que se calcula directamente de la fórmula molecular del compuesto por ejemplo la cantidad y tipo de átomos presentes o el peso molecular. Los 2D son un poco más complejos y representan información molecular sobre el tamaño, la forma y la distribución electrónica en la molécula. Los descriptores 3D son los más complejos y describen propiedades relacionadas con la estructura 3D de la molécula por ejemplo los puentes de hidrógeno intramoleculares o el área de superficie polar y no polar [39]. Los descriptores electrónicos 2D que representan propiedades atómicas locales como cargas parciales, electronegatividad residual o polarizabilidad efectiva y descriptores relacionados con la conectividad y forma como el volumen molecular son muy utilizados para la construcción de modelos de clasificación de

toxicidad [16]. Los descriptores de propiedades fisicoquímicas como la lipofilicidad o logP indican propiedades importantes de la farmacocinética y metabolismo de los compuestos químicos y también se encuentran entre los más utilizados.

La otra forma de representar moléculas es mediante fingerprints moleculares. Estos consisten en una serie de dígitos binarios (bits) que representan la presencia o ausencia de ciertas subestructuras particulares en una molécula. Son muy interpretables ya que cada bit corresponde a una subestructura o patrón específico. Algunos ejemplos son MACCS Keys o Molecular Access System Keys [40], PubChem Substructure Fingerprints (PCFP) y Extended Connectivity Fingerprints (ECFP) [41] entre otras. Los ECFP son fingerprints moleculares circulares diseñados para la caracterización molecular, la búsqueda de similitudes entre compuestos y el modelado de estructura-actividad. Es uno de los más utilizados como herramienta de búsqueda de similitudes en el descubrimiento de fármacos porque se puede calcular muy rápidamente y ha mostrado buenos resultados en una amplia variedad de aplicaciones [41]. Los PCFP también son muy utilizados y son una representación booleana de la presencia o ausencia de una subestructura determinada. Otras representaciones de la estructura son las MACCS Keys que representan la presencia de 166 estructuras codificadas en formato SMARTS. Es importante destacar que los fingerprints recién mencionados contienen únicamente información de las estructuras 2D de las moléculas y no contienen información sobre la conformación 3D. La publicación de Clark et al. [42], muestra que en una comparación entre fingerprints 2D y sus análogas 3D, generalmente las 3D no ofrecen un rendimiento superior para la búsqueda de similitudes entre moléculas.

En el caso de este trabajo se incorporarán ambos tipos de representaciones (descriptores de propiedades fisicoquímicas y fingerprints) y se realizará un análisis para ver cuáles de estos descriptores resultan más ponderados o importantes para clasificar la toxicidad de los compuestos por el modelo de ML implementado.

2.4.1. Herramientas

Actualmente existen herramientas de quimioinformática como el software PaDEL-Descriptor [43], Open Babel [36], Mordered-Descriptors [44] o RDKit [45] que permiten obtener cientos de descriptores de propiedades fisicoquímicas a partir de un archivo de moléculas. El tiempo de cálculo de estos descriptores dependerá principalmente del tamaño de la base de datos y del algoritmo que cada uno implementa. La tabla 5 muestra una descripción de los distintos softwares open-source para el cálculo de descriptores y fingerprints moleculares.

Software	**Descripción**
Padel Descriptors	PaDEL permite calcular un total de 1875 descriptores (1444 descriptores 1D y 2D y 431 3D) más 12 tipos de fingerprints moleculares. Los mismos se calculan utilizando el Chemistry Development Kit (CDK) y es un sofware de acceso público.
RDKit Descriptors	RDKit Descriptors es un módulo de la librería de Python RDKit diseñada para el manejo de compuestos químicos. Calcula 39 descriptores 1D y 2D y 13 3D. También permite el cálculo de algunos fingerprints moleculares.
Open Babel Descriptors	Open Babel Descriptors es un módulo de la librería Open Babel que permite calcular 27 descriptores 1D y 2D. .
Mordered Descriptors	Mordered es una librería de Python que permite calcular de forma rápida 1825 descriptores (1613 1D y 2D y 212 3D).

Tabla 5: Comparación de software para el cálculo de descriptores moleculares.

En el presente trabajo la librería Mordered-Descriptors se utilizó para obtener 1613 descriptores 1D y 2D. Estos incluyen el logP, número de dadores y aceptores de hidrógeno, violaciones a la regla de Lipinski [46], tipos de átomos y tipos de enlaces entre otros. PaDEL-Descriptors se utilizó para calcular 3 tipos de fingerprints moleculares: 307 fin-

gerprints de subestructuras reconocidas de PubChem (SubFP) y 307 fingerprints con la cuenta de la aparición de estas subestructuras, 1024 extended fingerprint (ExtFP) que extienden los fingerprints con bits adicionales que describen las características de los anillos y 166 MACCS fingerprints (MACCSFP). En total se obtuvo un vector de 3418 valores representativos de distintas características moleculares. Se incluyeron en el mismo archivo únicamente los parámetros que son computables para todas las moléculas. Se eliminaron luego los descriptores cuya varianza fue superior a 0,001 para evitar columnas iguales entre sí que no agreguen información importante para el aprendizaje. De esta manera, se obtuvo finalmente una matriz de 3024 descriptores que servirá de entrada al modelo de DL.

2.4.2. Normalización

La normalización o escalado de los datos es esencial para tener un vector de entrada en un rango de tratamiento similar (por ejemplo, 0-1). Esto permite evitar que números más grandes prevalezcan sobre los más pequeños impidiendo el proceso de aprendizaje. Una vez obtenida la matriz cruda con los descriptores moleculares, los datos se normalizaron utilizando la función Standard Scaler de Python. La misma permite estandarizar las características moleculares restando la media del conjunto de entrenamiento y escalando con la desviación estándar:

$$z = \frac{(x - u)}{s}$$

donde u es la media y s es el desvío estándar del conjunto de descriptores del entrenamiento. Ésta misma media y desvío calculados sobre los datos de entrenamiento se utilizaron para escalar a los valores de los conjuntos de validación y de evaluación final.

2.5. Métrica de evaluación

Uno de los puntos clave en el desarrollo de este trabajo fue la elección de una métrica adecuada para el problema. En la competencia Tox21 Data Challenge se utilizó el área bajo la curva ROC (ROC-AUC) para evaluar los modelos de ML presentados por los grupos investigadores. Si bien es una métrica muy utilizada y sirve especialmente para comparar el rendimiento entre distintos modelos, no es la más adecuada para el problema que se busca resolver. Esto se debe a dos motivos principales. En primer lugar, en el caso de toxicidad de moléculas potenciales a ser utilizadas como fármacos es necesario darle más importancia a clasificar de manera correcta la clase positiva (tóxica) ya que es más grave clasificar a un compuesto tóxico como si no lo fuera que el caso contrario. En segundo lugar, los datos están altamente desbalanceados. Se cuenta únicamente con un 5 % de datos anotados con clase positiva en promedio entre los tasks. Por esta razón la tasa de falsos positivos (FPR) se reduce debido a una gran cantidad de verdaderos negativos y por ende el valor de ROC-AUC será alto a priori y habrá menos margen para poder discriminar entre dos modelos con distintos hiperparámetros.

Una métrica más adecuada para el problema es el F_β score. Como se estableció en la introducción, la misma refleja principalmente la clasificación de la clase positiva. Si se selecciona un valor de β mayor a uno se dará más importancia a la sensibilidad minimizando la tasa de falsos negativos. Como lo que se busca justamente es minimizar los falsos negativos, se utilizará un valor de β mayor que uno para este trabajo. De todas formas, como esta herramienta busca dar soporte al proceso de diseño de fármacos, se utilizó un valor de β de 2 para no tener una tasa de falsos negativos tan alta y perder posibles candidatos que tengan actividad frente a un blanco de interés farmacéutico.

Esta métrica se calcula a partir de la precisión y el recall que, a su vez, se calculan sobre la clase predicha y no sobre las probabilidades. Por este motivo es importante una adecuada selección del umbral de separación de clases. La figura 9 muestra, a modo de

ejemplo, el F_2 score de un task calculado para distintos umbrales de separación con su umbral óptimo seleccionado. En el caso de este problema, al usar un valor de β igual a 2 se espera que el umbral óptimo para la separación de clases sea menor al estándar de 0.5 ya que de esta forma la clasificación tenderá a ser positiva. En este trabajo la métrica se calcula por separado para cada uno de los tasks con sus respectivos umbrales óptimos y luego se promedia para obtener un F_2 Macro score.

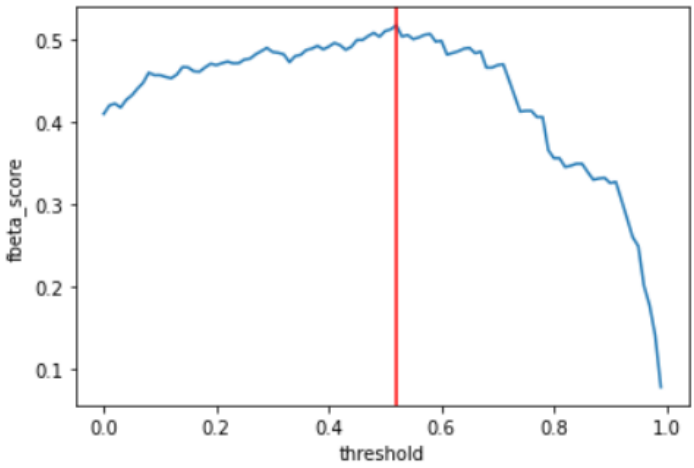


Figura 9: Ejemplo de F_β score de un único task calculado para distintos umbrales y selección del umbral óptimo.

El F_β score se implementó con el fin de ser utilizada por la librería Keras durante el entrenamiento de los modelos. Keras es una librería de Python que proporciona, de una manera sencilla, la creación de una gran gama de modelos de DL usando como backend otras librerías como TensorFlow [47]. La métrica se implementó de forma tal de obtener el score para el umbral óptimo. Se calculó para un set de 10 umbrales entre 0 y 1 de a pasos de 0.1 y se selecciona la mejor junto con el mejor umbral. De esta manera se tiene un valor de métrica y umbral óptimo para cada task que luego se promedian para tener un valor de F_β Macro. Para manejar los datos faltantes o "missing labels" presentes debido a que no todos los compuestos fueron medidos para todos los efectos tóxicos, se utilizó

una máscara con valores 0 para los datos que faltan y 1 en el resto de los casos. Siendo y_v e y_p vectores con los datos verdaderos y predichos respectivamente y M el número de muestras, se calcularon los valores de los VP, FP, VN y FN para cada tarea de la siguiente manera:

$$VP = \sum_{i=1}^{M} y_{v_i} * mask * y_{p_i}$$

$$VN = \sum_{i=1}^{M} (1 - y_{v_i}) * mask * (1 - y_{p_i})$$

$$FP = \sum_{i=1}^{M} (1 - y_{v_i}) * mask * y_{p_i}$$

$$FN = \sum_{i=1}^{M} y_{v_i} * mask * (1 - y_{p_i})$$

A partir de los mismos se obtienen la precisión y el recall para cada tarea con las ecuaciones 2 y 3. Luego se puede calcular el F_β score a partir de la ecuación 4. En caso de hacer un promedio entre los scores de las distintas tareas, se tendrá el F_β-Macro score.

Otra métrica adecuada al problema es el área bajo curva PR (AUC-PR). Al igual que con el F_β score, ésta se centra principalmente en la clase positiva y por ende se preocupa menos de la clase negativa más frecuente. Si bien el F_2 score se utiliza para la elección de los hiperparámetros, se calculará también el AUC-PR para evaluar el rendimiento y para la elección del modelo final. La misma también se implementó utilizando una máscara que permita evitar los datos faltantes no anotados. Se calculó obteniendo los valores para cada task y luego promediando entre los mismos.

2.6. Función de costo

La elección de la función de costo debe ser cuidadosamente analizada para cada problema en particular ya que será la que indique la dirección en la que se modifiquen los pesos del modelo. Una de las razones por las que muchas veces los algoritmos de ML son difíciles de optimizar es que las funciones de costo utilizadas por defecto no siempre son buenas aproximaciones a las métricas de evaluación. Las funciones de costo están diseñadas para ser diferenciables, y preferiblemente tener una convergencia suave, mientras que muchas métricas de evaluación no lo son. Por lo tanto, optimizar la métrica directamente como función de costo pareciera ser la forma más adecuada de abordar el problema. Sin embargo, en la práctica, muchas veces optimizar la métrica directamente (convirtiéndola en función de costo) no trae grandes beneficios frente a optimizar una función de costo estándar.

Una de las funciones más utilizada en modelos de DL para problemas de clasificación binaria es la Binary Cross Entropy o Entropía Cruzada. Esta función da una medida de la disimilaridad entre los valores reales de la clase y el valor estimado. La misma se calcula a partir de las probabilidades obtenidas para cada clase y los valores anotados de los datos según la siguiente ecuación:

$$\text{BCE} = -g_1 \log(p_1) - (1 - g_1)\log(1 - p_1) \tag{5}$$

siendo g_1 y p_1 los valores verdadero y obtenido respectivamente.

Como puede notarse, esta función no hace diferencia entre las clases lo cual es esencial en problemas donde una clase es más importante que la otra. En general se utiliza asociada a métricas como el Accuracy o el ROC-AUC. Por esta razón, entrenar un modelo de ML minimizando la entropía cruzada, no necesariamente maximizará el F_β score. Para esta

clase de problemas, suele utilizarse la Binary Cross Entropy Weighted (BCE Weighted) que es una modificación de la BCE otorgando pesos a la clasificación de las distintas clases.

En la investigación de Hashemi et al. [48] se propone una función de costo optimizable basada en el F_β score. La misma se calcula directamente de las probabilidades de pertenecer a la clase positiva obtenidas para cada tarea. Siendo M la cantidad de muestras, la ecuación 4 puede escribirse según los valores predichos p_i y reales g_i como:

$$F_\beta = \frac{(1+\beta^2)\sum_{i=1}^{M} p_i g_i}{(1+\beta^2)\sum_{i=1}^{M} p_i g_i + \beta^2 \sum_{i=1}^{M}(1-p_i)g_i + \sum_{i=1}^{M} p_i(1-g_i)} \tag{6}$$

Como se muestra en la ecuación 5 de la publicación recientemente mencionada [48], esta ecuación puede derivarse respecto de p_i para obtener su gradiente. Maximizar el F_β score es lo mismo que minimizar $1-$ F_β score. De esta manera puede optimizarse la métrica directamente sin la necesidad de utilizar pesos como en la BCE Weighted. A esta función de costo se la llamará en el desarrollo de este trabajo como F_β Loss. Al igual que en el F_β score, ajustando el hiperparámetro β podemos controlar el equilibrio entre precisión y recall (FP y FN).

Esta función de costo, al igual que la métrica, se implementó utilizando una máscara para lidiar con los "missing labels" o datos no anotados de ciertos compuestos.

2.7. Implementación de DL

El código se implementó en Python utilizando principalmente la librería Keras anteriormente mencionada para la construcción de los modelos de DL. Como entorno de programación se utilizó Google Colab [49], un servicio en la nube que provee de una

Jupyter Notebook y tiene la ventaja de contar con una GPU la cual acelera el proceso de entrenamiento de los modelos hasta 10 veces. En el Apéndice B se incluyen ciertas partes del código, consideradas importantes, para la implementación del modelo de DL.

2.7.1. Búsqueda de hiperparámetros

Se utilizaron distintas librerías de Python como Talos [50] y SkMultiLearn para realizar una búsqueda de los hiperparámetros que mejor se ajusten al problema que se busca resolver. Para la separación de los datos en conjuntos de entrenamiento y validación, se utilizan las librerías SKMultiLearn y Iterative Stratification que permiten separar datos Multi-Label de forma tal que se mantenga lo más posible la distribución de clases de todos los tasks del conjunto original [51]. Se utilizó el método de Cross-Validation mediante el cual los datos disponibles se dividen en subconjuntos de entrenamiento y validación y se calcula un promedio del desempeño en los mismos. Más adelante, en la sección 3.1, se explica en detalle cómo se decidió implementar la búsqueda por Cross-Validation. La figura 10 muestra un diagrama con el proceso de búsqueda y selección de hiperparámetros dentro de la implementación de DL.

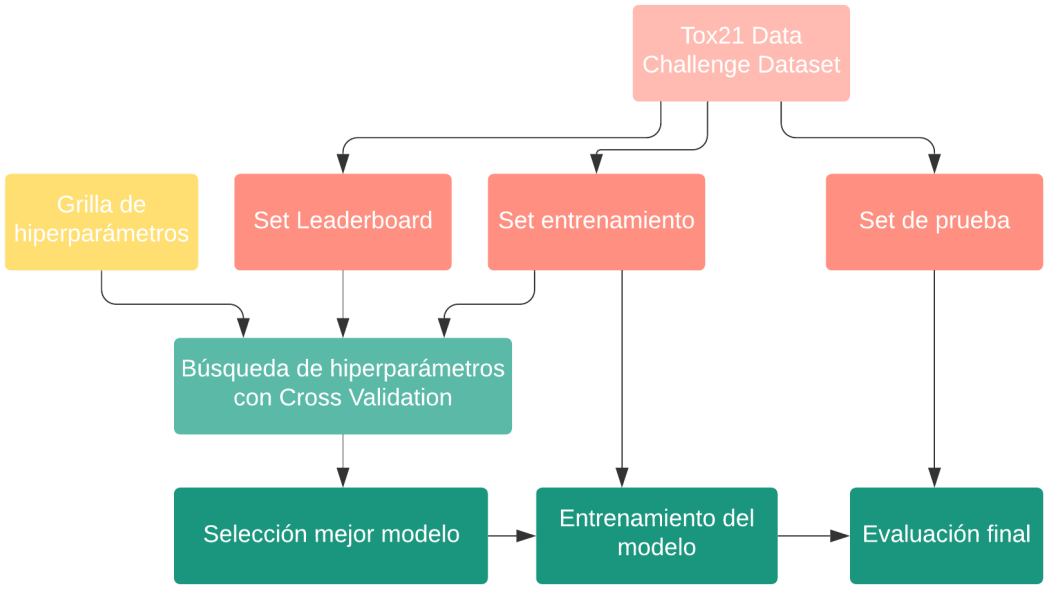


Figura 10: Esquema representativo de la implementación de DL y la selección de hiperparámetros. Con los sets de entrenamiento y Leaderboard de la competencia y una grilla de hiperparámetros determinada se realiza una búsqueda del mejor modelo utilizando Cross-Validation (que se detalla en la sección 3.1). Luego se entrena este modelo con la totalidad de los datos del set de entrenamiento y se presentan los resultados sobre el conjunto de prueba o testeo final de la competencia.

Como hiperparámetros de control del entrenamiento se prueban distintos algoritmos de optimización, tasas de aprendizaje o learning rates, número de epochs, funciones de activación, inicializadores, tasas de Dropout y técnicas de regularización. Los valores de los distintos hiperparámetros buscados se muestran en la tabla 6. Se evaluó también el uso de capas de Batch Normalization entre cada capa densa. Para las funciones de activación, se consideran tanto funciones ReLU, PreLu y tanh. Las primeras dos se muestran en la figura 11. La función de activación ReLU o unidad rectificada lineal es una función lineal por partes que mantiene el valor de entrada si la misma es positiva y, de lo contrario, asigna

cero. Se ha convertido en una de las funciones de activación predeterminada para muchos tipos de redes neuronales porque los modelos que las usan son fáciles de entrenar y suelen lograr buenos rendimientos especialmente por ser una gran solución para el problema del gradiente de fuga o "vanishing gradient" al que se enfrentan las redes de muchas capas [22]. Este problema se refiere a los casos en los que el gradiente de la función de costo se va desvaneciendo a lo largo de las capas a valores muy pequeños, impidiendo el aprendizaje de la red en las capas más profundas. La función PreLu es una función de activación Relu tradicional pero con una pendiente para valores negativos como se muestra en la figura 11. La función de activación Tanh es una función no lineal diferenciable con forma sigmoidea que devuelve los resultados entre -1 y 1. Para el entrenamiento se utilizó en todos los casos el optimizador Adam implementado en la función de Keras con los parámetros que trae por defecto.

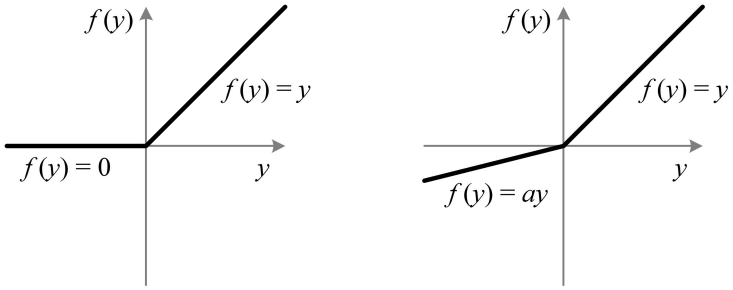


Figura 11: Funciones de activación Relu (izquierda) y PreLu (derecha) de redes neuronales.

Para la capa de salida se implementó la función sigmoidea Softmax. Esta última devuelve, para cada tarea, la distribución de probabilidad de pertenecer a las clases de salida. Su gráfica se encuentra en la figura 12. La misma calcula la relación de la exponencial de un valor de entrada con respecto a la suma de los valores exponenciales de todas las entradas según la siguiente ecuación:

$$\sigma(z) = \frac{\exp z_j}{\sum_{i=0}^{K} \exp z_i}, j = 0, .., K \tag{7}$$

donde K es el número de valores de la salida.

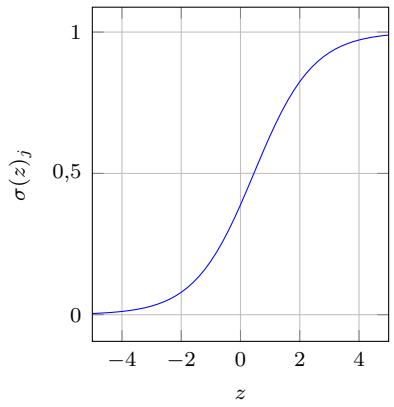


Figura 12: Función Softmax utilizada como función de activación en la capa de salida del modelo.

Con respecto a los hiperparámetros relacionados a la arquitectura de la red, se consideran distintas cantidades de capas ocultas (3 o 4) y distinto número de neuronas por cada capa (512, 1024, 3500 y 4096).

Hiperparámetros	**Valores considerados**
Número de neuronas por capa	{512, 1024, 3500, 4096}
Cantidad de capas ocultas	{3,4}
Dropout	{0, 0.5 capa de entrada y 0.25 en capas ocultas, 0.3 capa de entrada y 0.15 en capas ocultas, 0.4 en entrada y 0.2 en capas ocultas }
Regularización L2	{l2 = 0.01, l2 = 0.001, l2= 0.0001}
Batch Size	{256, 512}
Learning rate	{0.01, 0.05, 0.1}
Funciones de activación	{ReLu, PreLu, tanh}
Inicializadores	{Glorot Normal, Glorot Uniform, Random Normal}

Tabla 6: Grilla de hiperparámetros con los valores considerados.

Capítulo 3

3. Clasificación de compuestos y resultados.

El procedimiento general para construir un modelo de DL consiste en aproximadamente cuatro pasos: recolección de datos de las bases de datos disponibles, descripción numérica de los datos y pre-procesamiento, diseño e implementación del modelo y por último su evaluación [13]. El pipeline general que se implementa en este trabajo se muestra en la figura 13. En esta sección se desarrollan las consideraciones que se tienen en cuenta en cada uno de estos pasos y los resultados obtenidos en los mismos.

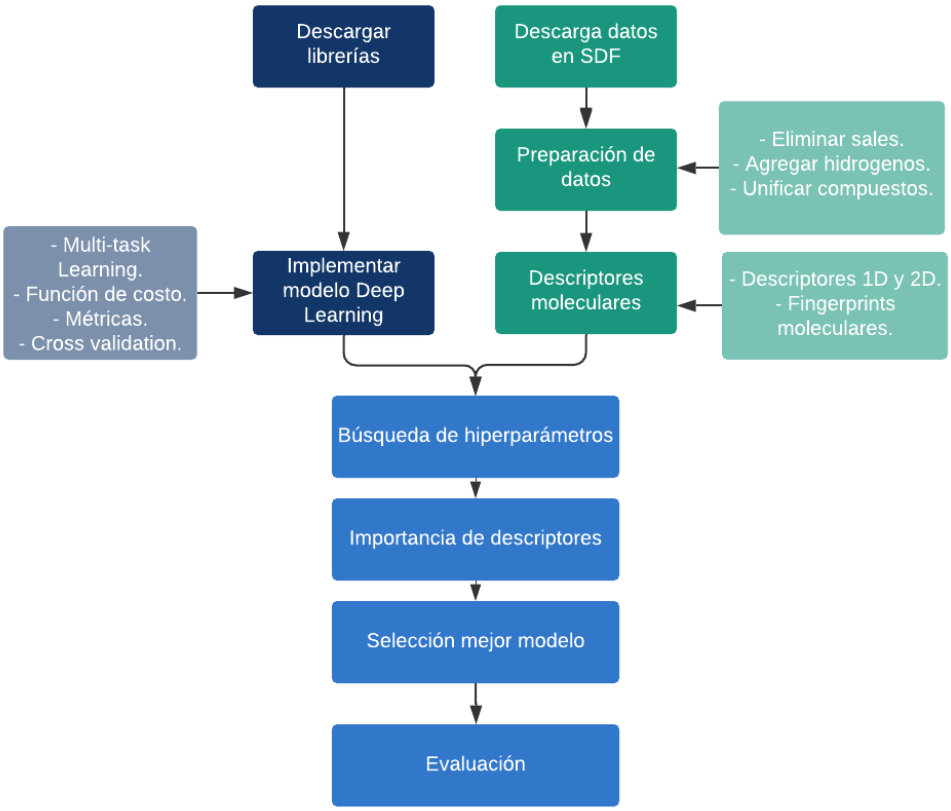


Figura 13: Pipeline para la clasificación tóxica de compuestos químicos.

3.1. Diseño y optimización de modelos de DL

Una vez preparados los datos y calculados los descriptores moleculares se implementó un modelo de DL, en particular un MLP, y se evaluó el mismo en base a una métrica adecuada. Para esto, se realizó una búsqueda exhaustiva de hiperparámetros mediante el método de Cross-Validation que maximicen el F_2-Macro score de todas las tareas. Es decir, se buscó el modelo general (con entrenamiento MTL) que permita obtener el mejor F_2 score en promedio de las 12 tareas. En este punto también se consideraron los valores mínimo y máximo del score obtenido, priorizando modelos donde el valor mínimo y máximo de F_2 en el set Leaderboard se encuentren más cerca de la media.

Con la librería Iterative Stratification se obtuvieron dos 'folds' o subconjuntos de índices de los datos disponibles (el conjunto de entrenamiento junto con el Leaderboard) separados en un 80/20 % para entrenamiento y para validación respectivamente. El tercer fold se dividió aparte para contener al set de validación Leaderboard otorgado por la competencia. Para decidir la elección del mejor modelo, se le dio más importancia al F_2 score obtenido por éste último subconjunto Leaderboard. Este proceso se encuentra ilustrado en el diagrama de la figura 14 a continuación.

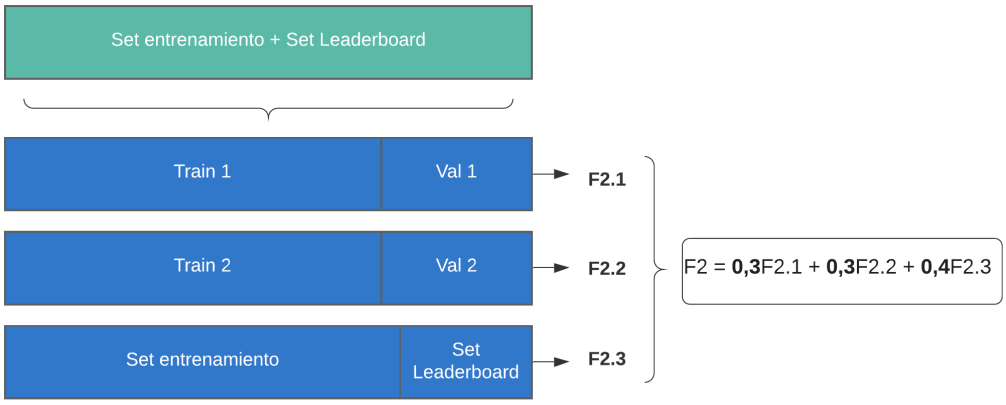


Figura 14: Esquema de la división de los datos de entrenamiento y validación de la competencia. Se calcula el F_2-Macro score medio como la media ponderada del F_2-Macro score obtenido para los 3 subconjuntos de validación. En base a este resultado se selecciona el mejor modelo.

Como primer paso, se buscaron los hiperparámetros para el control de entrenamiento. Para esto se partió de un modelo elegido por prueba/error con una arquitectura de cuatro capas ocultas de 1024 neuronas en cada capa y función de activación ReLu. Una vez seleccionados el número de epochs, la tasa de aprendizaje y el batch size, se probaron distintas funciones de activación e inicializadores. En todos los casos para la capa de salida se implementó la función Softmax.

Los mejores resultados se obtuvieron para la función ReLu con inicializador Glorot-Normal que inicializa los pesos con una distribución normal. Partiendo de estos valores se probaron los hiperparámetros correspondientes a la arquitectura de la red: cantidad de capas y cantidad de neuronas por capa.

3.2. Modelo final elegido

Se seleccionó finalmente un modelo de cuatro capas densas de 1024, 512, 512, 512 neuronas por capa con una función de activación ReLU tanto para la capa de entrada como para las ocultas. La función Softmax se implementó a la salida con 12 neuronas que se corresponden a los 12 tasks que fueron evaluados en simultáneo. Con respecto al control del entrenamiento, se seleccionó una tasa de aprendizaje de 0.01, un inicializador Glorot-Normal en cada capa y se implementó una regularización l2 con un valor de l2=0.001. Las mejores tasas de dropout fueron de 0.3 en la capa de entrada y 0.15 en capas ocultas. Además se incluyó una capa de Batch Normalization entre cada capa densa. Se utilizó el optimizador Adam implementado en la librería de Keras con sus valores defecto, Early Stopping para frenar el sobre-ajuste a los datos de entrenamiento y la función de Keras Reduce Learning Rate para evitar caer en plateaus de la función de costo.

3.3. Selección de características

El uso de una gran cantidad de descriptores moleculares puede resultar en overfitting cuando el conjunto de datos de entrenamiento es pequeño y por lo tanto la selección de un subconjunto de características que genere una más alta performance debe realizarse antes de la construcción del modelo final. La experiencia de previas investigaciones muestra que un buen algoritmo de selección de features puede eliminar de forma eficaz elementos irrelevantes y características redundantes [52]. Esto conduce a una mejor comprensión de los datos y una mejora en el rendimiento de los modelos de ML al mejorar la generalización e interpretabilidad del modelo de aprendizaje. Por otro lado, las características moleculares que tienen más influencia en la toxicidad de los compuestos (por lo menos para el aprendizaje de la red) resulta información interesante de conocer.

Para esto se decidió implementar una función que permite obtener un ranking de los descriptores a partir de una red neuronal y del peso que le otorga a cada descriptor durante

el entrenamiento. La idea es que para comprender verdaderamente cómo una característica afecta las predicciones, se mantienen todos los valores de entrada constantes y solo se varía la característica que se quiere analizar. Al medir el resultado de la nueva predicción, se puede establecer una relación entre la entrada y la predicción. Para esto se usan funciones de Python diseñadas para este fin: feature_selection de la librería SKLearn y otras dos funciones de la librería Eli5.

Por lo tanto, se analizó la importancia de cada uno de estos descriptores para el entrenamiento y optimización de la F_2 Loss. Se implementó la función PermutationImportance de la librería Eli5 que permite generar un ranking según importancia de todos los descriptores moleculares. El Apéndice B muestra el código con la implementación de esta función y su resultado. Para esto, el modelo final elegido se entrena utilizando el conjunto de datos de entrenamiento y la función iterativamente altera los valores de un único descriptor insertando ruido en el mismo. En cada una de estas iteraciones se evalúa el F_2-Macro score sobre el conjunto Leaderboard y se compara este resultado respecto al obtenido en el modelo original entrenado con todos los descriptores. En base a estas diferencias obtenidas para cada descriptor, se asigna más peso a los que influencian más el resultado de la métrica y de esta manera se establece un ranking según el peso asignado a cada descriptor. La tabla 7 muestra los 10 descriptores a los que se les asignó el mayor peso para la clasificación de toxicidad.

Ranking	**Nombre del descriptor**	**Descripción**
1	SubFP166	Presencia de tiocianato
2	SubFPC166	Cuenta de grupos tiocianatos
3	SubFP53	Presencia de ion immonium
4	SubFP47	Presencia de compuestos organometálicos
5	SubFPC47	Cuenta de compuestos organometálicos
6	ETA_epsilon_5	Descriptores fisicoquímicos de posiciones atómicas
7	ATSC1m	Descriptores de autocorrelación
8	MACCSFP12	Presencia de Cu, Zn, Ag, Cd, Au y Hg
9	SubFPC76	Presencia de Enaminas
10	AATS0m	Descriptores de autocorrelación

Tabla 7: Top 10 ranking de descriptores moleculares a los que se les asignó más importancia en la clasificación de toxicidad del modelo entrenado con MTL.

A partir de este ranking se seleccionó un subconjunto de 1028 descriptores que mejoraron el F_2-Macro score evaluado en el conjunto Leaderboard. El descriptor al que se le asignó más importancia es el SubFP166 y el SubFPC166. Estos dos son fingerprints moleculares que indican la presencia y la cuenta de grupos tiocianatos ($[SCN]^-$) en la estructura de las moléculas. Los tiocianatos son un grupo de compuestos formados por una combinación de azufre, carbono y nitrógeno. Es el producto principal que se forma en el cuerpo humano en casos de intoxicación con cianuro para bajar las concentraciones del mismo en sangre [53]. Si bien los tiocianatos son menos perjudiciales que el cianuro en seres humanos, se sabe que afectan la glándula tiroides, reduciendo la habilidad de la glándula para producir hormonas que son necesarias para el funcionamiento normal del

cuerpo y produciendo intoxicación. Se han descrito hipoxia, náuseas, acúfenos, espasmo muscular, desorientación y psicosis cuando los niveles del tiocianato son superiores a los 10 mg/100 ml [53].

También siguen en el ranking algunos de los Extended Topochemical Atom (ETA) descriptors que son parte de los descriptores de propiedades fisicoquímicas calculados con PaDEl y que determinan contribuciones de vértices o posiciones específicas de átomos dentro de subestructuras comunes hacia la funcionalidad total de la molécula. Se ha estudiado anteriormente la influencia de estos descriptores en toxicología y se ha llegado a la conclusión de que "los descriptores de ETA son lo suficientemente ricos en información química para codificar las características estructurales que contribuyen a las toxicidades y estos índices pueden usarse en combinación con otros descriptores topológicos y fisicoquímicos para el desarrollo de modelos predictivos de relación estructura-actividad" [54].

Otro descriptor alto en el ranking es el MACCS Keys número 12. Este indica la presencia u ausencia de los elementos metálicos: Cu, Zn, Ag, Cd, Au y Hg. Todos los metales pueden tener efecto tóxico pero los que tienen más interés en toxicología clínica son el mercurio, plomo, hierro, cadmio, cromo, cobalto, cobre, niquel y zinc [55]. Si bien su toxicidad dependerá en gran medida de la dosis ingerida, son un indicador posible de toxicidad ya que generan efectos tóxicos en los riñones, en los sistemas óseo y respiratorio y algunos están clasificados como carcinógenos para los seres humanos. A estos les siguen, por ejemplo, la cuenta del SubFP5 que indica la cantidad de dobles enlaces presentes en la estructura, la presencia de di-fenoles y el número de átomos en grupos aromáticos.

3.4. Resultados

3.4.1. Validación

El modelo final elegido se entrenó en conjunto para las 12 tareas con el subset de descriptores seleccionado y se evaluó el rendimiento del F_2-Macro score en los tres conjuntos de validación usados previamente para la búsqueda de hiperparámetros por Cross-Validation: dos sets separados de los datos de entrenamiento y el set Leaderboard. Los resultados obtenidos en cada uno de estos conjuntos se muestran en la tabla 8 a continuación.

	Val 1	**Val 2**	**Leaderboard**
NR-AhR	0.710	0.728	0.605
NR-AR	0.579	0.672	0.110
NR-AR-LBD	0.748	0.792	0.469
NR-Aromatase	0.556	0.568	0.648
NR-ER	0.543	0.568	0.488
NR-ER-LBD	0.645	0.608	0.357
NR-PPAR-γ	0.640	0.511	0.417
SR-ARE	0.654	0.674	0.733
SR-ATAD5	0.616	0.636	0.526
SR-HSE	0.631	0.564	0.476
SR-MMP	0.799	0.803	0.706
SR-p53	0.637	0.614	0.492
Promedio	0.647	0.645	0.502

Tabla 8: F_2 score en los 3 folds de validación.

Como se puede ver, el conjunto Leaderboard obtuvo peores desempeños de F_2 score

en la mayoría de las tareas. En este conjunto de datos la mayoría de las tareas se encuentran más desbalanceadas hacia la clase negativa que en los datos de entrenamiento. Por lo tanto, al tener menos valores activos para la validación, un error en la clasificación tiene alta influencia en los resultados de la métrica. Esto puede notarse al observar la matriz de confusión del conjunto Leaderboard de la tabla 9. En el caso de NR-AR, por ejemplo, solo se cuenta con 3 compuestos activos y la clasificación incorrecta de uno solo de ellos genera un recall de 0.67 que si bien es aceptable, está por debajo del promedio obtenido para todas las tareas. A su vez, es interesante notar que los compuestos del set Leaderboard corresponden a otra librería (LOPAC) y por ende fueron medidos bajo distintas condiciones experimentales que los compuestos de la librería Tox21 que se utiliza para el entrenamiento y están incluidos en los otros dos conjuntos de validación.

Como se estableció recientemente, una mejor visualización del desempeño puede tenerse mirando las matrices de confusión. La tarea que obtuvo el mejor desempeño en el set Leaderboard (0.733) es el receptor SR-ARE que obtuvo la siguiente matriz de confusión:

$$\begin{bmatrix} 137 & 49 \\ 6 & 40 \end{bmatrix}$$

En ese caso seis de 46 compuestos tóxicos no fueron detectados por la red. Se nota una más alta sensibilidad que precisión como era de esperarse al elegir el factor β mayor a uno.

El peor desempeño se obtuvo en el set de NR-AR (0.110) que es el más desbalanceado dentro de los conjuntos del Leaderboard (1 % de compuestos activos) y para el mismo la matriz de confusión fue la siguiente:

$$\begin{bmatrix} 209 & 77 \\ 1 & 2 \end{bmatrix}$$

Puede verse que el F_2 score es bajo ya que hay solo tres compuestos activos y uno no fue detectado por la red. Además, hay un porcentaje alto de falsos positivos que disminuye la precisión. Es importante mencionar que el conjunto NR-AR también obtuvo un desempeño bajo de ROC-AUC score (0.34) en el set Leaderboard por el grupo ganador de la competencia [22].

A continuación se muestran los resultados de las distintas métricas de evaluación sobre los 12 conjuntos de datos entrenados en simultáneo:

	F_2 **score**	**Recall**	**Precision**	**VN**	**FP**	**FN**	**VP**
NR-AhR	0.605	0.867	0.274	170	69	4	26
NR-AR	0.120	0.667	0.025	209	77	1	2
NR-AR-LBD	0.469	0.750	0.188	235	13	1	3
NR-Aromatase	0.648	1.000	0.269	147	49	0	18
NR-ER	0.488	0.741	0.206	160	77	7	20
NR-ER-LBD	0.357	0.700	0.121	223	51	3	7
NR-PPAR-γ	0.417	0.400	0.500	243	6	9	6
SR-ARE	0.733	0.870	0.449	137	49	6	40
SR-ATAD5	0.526	0.640	0.308	210	36	9	16
SR-HSE	0.476	0.600	0.261	237	17	4	6
SR-MMP	0.706	0.973	0.336	127	71	1	36
SR-p53	0.492	0.643	0.254	186	53	10	18
Promedio	0.502	0.738	0.266				

Tabla 9: Métricas de evaluación para las 12 tareas sobre el conjunto de validación Leaderboard.

La mayoría de las tareas obtuvieron un Recall mayor a 0.60 con un máximo de 1 en NR-Aromatase donde los 18 compuestos tóxicos estuvieron bien clasificados por la red. El NR-PPAR-γ es el único que muestra un valor de precisión mayor al recall. El NR-AR

que tiene un F_2 score muy bajo, tiene un valor de recall aceptable. La precisión en general no fue tan buena ya que hay una tasa alta de falsos positivos al elegir un umbral de separación que favorece más la sensibilidad.

Como se estableció previamente, es de esperarse obtener un umbral óptimo de separación de clases menor a 0.50 para dar más importancia a la sensibilidad que a la presición. El siguiente gráfico muestra los resultados de F_2 score en un conjunto de validación y en el conjunto Leaderboard junto con el umbral óptimo seleccionado en color rojo. Para la predicción de toxicidad del conjunto de prueba final, se utilizaron los umbrales óptimos identificados para el Leaderboard.

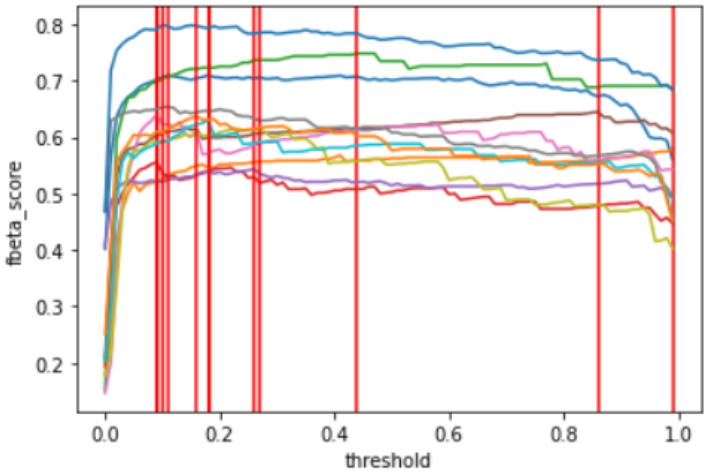


Figura 15: F_2 score para distintos umbrales y umbral óptimo para cada tarea en un set de validación separado del entrenamiento.

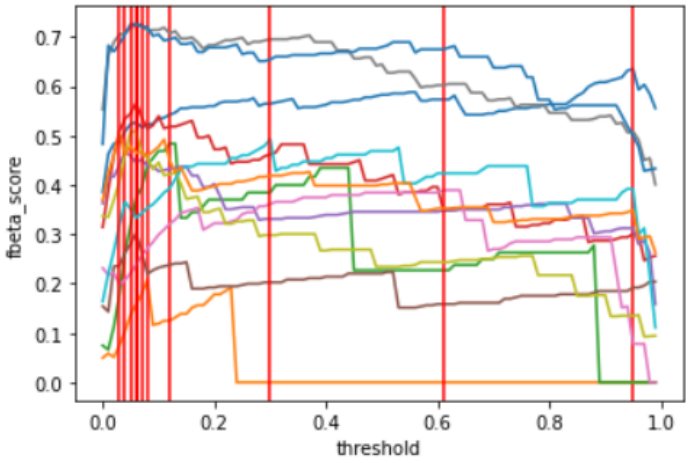


Figura 16: F_2 score y umbral óptimo para cada tarea en el set Leaderboard.

Es interesante notar que para dos de las tareas el umbral óptimo de separación está por arriba de 0.50. En el caso de NR-PPAR-γ, el umbral que maximiza el F_2 es de 0.96, lo cual explica que esta tarea tenga un valor de precisión mayor al recall.

3.4.2. Funciones de costo

El modelo para MTL se entrenó utilizando tres funciones de costo distintas para comparar la incidencia de las mismas en el F_2 score sobre el conjunto Leaderboard. Además de la F_2 Loss, se utilizó la BCE y la BCE Weighted dándole más peso a la clasificación de la clase positiva (0.6 a la clase tóxica y 0.4 a la no tóxica mayoritaria). Los resultados del score obtenidos en el conjunto de validación de la competencia se muestran en la tabla 10 a continuación.

	Entrenamiento con F_2 Loss	**Entrenamiento con BCE weighted**	**Entrenamiento con BCE**
NR-Ahr	0.635	0.608	0.657
NR-AR	0.208	0.072	0.179
NR-AR-LBD	0.484	0.429	0.313
NR-Aromatase	0.567	0.563	0.559
NR-ER	0.467	0.388	0.421
NR-ER-LBD	0.303	0.377	0.339
NR-PPAR-γ	0.390	0.380	0.353
SR-ARE	0.726	0.775	0.724
SR-ATAD5	0.509	0.495	0.431
SR-HSE	0.493	0.424	0.380
SR-MMP	0.726	0.708	0.738
SR-p53	0.506	0.552	0.445
Promedio	0.501	0.481	0.461

Tabla 10: F_2 score sobre el set Leaderboard con entrenamiento MTL utilizando distintas funciones de costo

Los resultados muestran que en este problema utilizar directamente el F_2 score como función de costo (F_2 Loss) no mejora significativamente el rendimiento de la métrica evaluada en promedio frente a utilizar la BCE Weighted. Sin embargo, el valor mínimo obtenido con la F_2 Loss fue 3 veces más grande que entrenar con la BCE Weighted. Agregarle pesos a las clases sobre la función de costo estándar BCE, genera que los resultados de F_2 score obtenidos en este caso se parezcan más a los resultados al optimizar directamente esta métrica.

3.4.3. Testeo final

Para el testeo final del modelo se utiliza el conjunto de evaluación de la competencia cuyos datos nunca fueron vistos por los modelos durante el entrenamiento. Para la preparación de estos datos se implementaron los mismos pasos de las secciones anteriores: eliminación de sales, protonación, cálculo de descriptores y luego se normalizó el conjunto con la media y el desvío obtenidos para el set de entrenamiento. Para la separación de clases se utilizaron los umbrales óptimos de separación que maximizan el score del conjunto Leaderboard. En este caso los resultados de F_2 obtenidos con el modelo MTL fueron los siguientes (Tabla 11):

	F_2 **score**	**Recall**	**Precision**	**VN**	**FP**	**FN**	**VP**
NR-AhR	0.691	0.808	0.437	458	76	14	59
NR-AR	0.172	0.167	0.200	563	8	10	2
NR-AR-LBD	0.091	0.125	0.044	549	22	7	1
NR-Aromatase	0.259	0.256	0.270	460	27	29	10
NR-ER	0.492	0.804	0.193	290	172	10	41
NR-ER-LBD	0.200	0.200	0.200	561	16	16	4
NR-PPAR-γ	0.349	0.484	0.165	495	76	16	15
SR-ARE	0.527	0.624	0.326	339	120	35	58
SR-ATAD5	0.395	0.632	0.158	453	128	14	24
SR-HSE	0.401	0.500	0.225	547	38	11	11
SR-MMP	0.672	0.917	0.325	367	114	5	55
SR-p53	0.462	0.825	0.168	409	164	7	33
Promedio	0.400	0.530	0.230				

Tabla 11: Métricas de evaluación para las 12 tareas sobre el conjunto de prueba final.

En este caso el valor de F_2 score más alto se obtuvo para NR-AhR con un resultado de 0.69. La sensibilidad para esta tarea fue de 0.80 donde 14 de los 73 compuestos tóxicos

no fueron detectados por la red. La tarea NR-AR-LBD obtuvo el valor de F_2 más bajo. El conjunto de datos presentaba ocho compuestos activos de los cuales sólo uno fue detectado por la red.

3.4.4. Single-Task vs Multi-Task Learning

Uno de los objetivos específicos de este trabajo es evaluar el desempeño de los modelos MTL frente al entrenamiento singular de cada tarea. Para ello, los compuestos y sus anotaciones para cada ensayo de toxicidad se entrenan por separado. Todos los hiperparámetros de la red se mantuvieron igual con respecto al modelo general elegido para poder comparar los resultados con los del entrenamiento MTL.

La tabla 12 muestra estos resultados con las 12 tareas para el set Leaderboard y para el set de evaluación final.

	Leaderboard set		*Final Test set*	
	Single Task F_2	**Multi Task F_2**	**Single Task F_2**	**Multi Task F_2**
NR-AhR	0.595	**0.605**	0.632	**0.691**
NR-AR	**0.203**	0.110	**0.238**	0.172
NR-AR-LBD	**0.526**	0.469	**0.100**	0.091
NR-Aromatase	0.496	**0.648**	**0.400**	0.259
NR-ER	0.385	**0.488**	0.445	**0.492**
NR-ER-LBD	0.306	**0.357**	**0.230**	0.200
NR-PPAR-γ	0.3846	**0.417**	0.289	**0.349**
SR-ARE	0.699	**0.733**	**0.547**	0.527
SR-ATAD5	0.506	**0.526**	0.337	**0.395**
SR-HSE	**0.522**	0.476	0.308	**0.401**
SR-MMP	0.703	**0.706**	0.553	**0.672**
SR-p53	**0.509**	0.492	0.444	**0.462**
Promedio	0.486	**0.502**	0.376	**0.400**

Tabla 12: Comparación de F_2 score entre entrenamiento Single-Task y Multi-Task en los conjuntos Leaderboard y de evaluación final.

En el conjunto de prueba final se ve que en la mayoría de los casos el entrenamiento conjunto mejora el desempeño del F_2 score pero no en todos. Por ejemplo en las tareas NR-AR, NR-Aromatase y SR-ARE la métrica empeora levemente. En los otros nueve conjuntos, el entrenamiento MTL permitió mejorar los resultados frende al entrenamiento simple. En el caso de SR-MMP, el score mejora notablemente pasando de un valor de 0.55 a

un score de 0.67. Estos resultados favorecen lo determinado por previas investigaciones de que el entrenamiento MTL de distintos conjuntos de datos puede mejorar el rendimiento ya que la red transfiere características aprendidas en una tarea a otras en las que esa característica es faltante.

3.4.5. Entrenamiento de tareas correlacionadas.

Según lo establecido en la publicación de Wenzel et al., entrenar únicamente tareas que se encuentran correlacionadas puede aumentar, en algunos casos, el desempeño de los modelos de ML [23]. En el caso del trabajo mencionado, al entrenar un modelo de regresión de Deep Learning de tres conjuntos de compuestos con anotaciones de toxicidad hepática correlacionados, los tres mejoraron levemente su desempeño frente a ser entrenados por separado. Sin embargo, al entrenarlos junto con una tarea de permeabilidad no correlacionada, ésta última mostró un peor desempeño en el entrenamiento MTL frente a ser entrenada por separado. En esta sección se analiza si existe un subconjunto de tareas que al entrenarse en conjunto mejore la performance del entrenamiento MTL de los 12 tasks.

En el caso de Tox 21, ningún par de conjuntos de datos muestra un coeficiente de correlación absoluto mayor a 0.60. Las respuestas del receptor nuclear NR-ER y NR-ER-LBD son las más correlacionadas junto con NR-AR y NR-AR-LBD. Estos dos pares de conjuntos entre sí también presentan un coeficiente de correlación significativo (entre 0.30 y 0.50, siendo NR-AR la menos correlacionada con NR-ER y NR-ER-LBD). Se evaluó entonces el desempeño del modelo MTL entrenando en conjunto a estas cuatro tareas.

Se observa en los resultados de la tabla 13 que el único task que se ve favorecido por este entrenamiento es el NR-ER. El NR-AR-LBD apenas mejora su resultado y las otras dos tareas muestra un igual o más bajo desempeño del F_2 score frente al entrenamiento múltiple con los 12 tasks. Como se observó en la investigación de Wenzel et al. [23], si bien

los 12 conjuntos de datos no están correlacionados, la red logra distinguir correlaciones ocultas entre algunas características y de esta forma transfiere el conocimiento entre las distintas tareas aún sin estar muy correlacionadas.

	12 tasks training	**4 Nuclear-response training**
NR-AR	0.172	0.117
NR-AR-LBD	0.091	0.138
NR-ER	0.385	0.420
NR-ER-LBD	0.306	0.303
Promedio	0.239	0.245

Tabla 13: F_2 score del set de evaluación final entrenando a cuatro tareas del panel de Nuclear Response Effects.

En la figura 8 se ve que NR-AR muestra una correlación baja con la mayoría de las tareas, menos con NR-AR-LBD. La misma obtuvo un F_2 score mayor al entrenarse por separado (Single-Task) que en el entrenamiento conjunto. Como se estableció previamente, lo mismo ocurrió en la publicación de Wenzel et al. [23], donde la tarea de permeabilidad que se encontraba poco correlacionada empeoró su desempeño al entrenarse en conjunto. Se entrenó entonces a esta tarea junto con NR-AR-LBD y el F_2 score obtenido en el conjunto de prueba final fue de 0.24, lo cual es igual que el score obtenido en su entrenamiento Single-Task. Por lo tanto, esta tarea no se ve afectada al ser entrenada con NR-AR-LBD las cuales estan correlacionadas pero sí disminuye su desempeño al entrenarse en conjunto con tareas con las cuales no mantiene correlación.

Con respecto al panel de respuesta al estrés (SR), las cinco tareas presentan coeficientes de correlación entre 0.40 y 0.50 entre sí. Se evaluó entonces el F_2 score al entrenar

estas tareas por separado. Los resultados que se obtuvieron en el set de evaluación final se muestran en la tabla 14.

	12 tasks training	**Stress-response training**
SR-ARE	0.527	0.583
SR-ATAD5	0.395	0.412
SR-HSE	0.401	0.387
SR-MMP	0.672	0.684
SR-p53	0.462	0.428
Promedio	0.492	0.499

Tabla 14: F_2 score sobre el set de evaluación final entrenando a las cinco tareas del panel de Stress Response Effects.

En este caso la tarea SR-ARE mejoró significativamente su desempeño al entrenarla junto con el panel SR. La misma tiene un coeficiente de correlación mayor a 0.50 con todas las tareas de este panel. SR-ATAD5 y SR-MMP también mostraron una mejora del F_2 score.

3.5. Búsqueda de hiperparámetros por separado

Al observar los resultados se puede ver que en general se obtiene un valor de sensibilidad alto y una precisión baja al utilizar el F_2 score como función de costo directamente. Sin embargo, algunas tareas muestran mejores resultados que otras habiendo una diferencia grande entre las que obtienen los valores máximos y las que obtienen los mínimos. Es decir, la capacidad de aprendizaje del modelo final elegido varía mucho entre tareas.

Una alternativa para resolver el problema es realizar una búsqueda de hiperparámetros separada para cada task. En este caso, aún entrenando las 12 tareas en simultáneo se buscan los hiperparámetros que maximizan el F_2 score de una tarea en particular. De esta forma se tendrían 12 modelos distintos de entrenamiento múltiple, uno para entrenar cada task. Esto a su vez permite aplicar la técnica de Early Stopping a la métrica del task particular que se está evaluando y por ende reduce el overfitting sobre el mismo. Si bien esta técnica podría mejorar los resultados también lleva más tiempo de entrenamiento y más costo computacional. Por este motivo, con el fin de probar si es posible mejorar los resultados de F_2 score, se realizó una búsqueda de hiperparámetros separada para algunas tareas. Para esto se siguen los pasos de secciones anteriores y se utilizan los mismos hiperparámetros de la tabla 6. La búsqueda se lleva a cabo con el fin de maximizar el F_2 score de las tareas NR-AR-LBD, NR-PPAR-γ y SR-HSE que no habían tenido un buen desempeño en el conjunto de prueba con el modelo previo. Los modelos elegidos para cada task se detallan en el Apéndice A. La tabla 15 muestra los nuevos resultados obtenidos al realizar una búsqueda separada para cada tarea. La tabla 16 muestra los resultados sobre el set de prueba final junto con los del modelo final obtenidos previamente.

	Leaderboard			*Testeo Final*		
	F_2 score	**Recall**	**Precision**	**F_2 score**	**Recall**	**Precision**
NR-AR-LBD	0.600	0.750	0.330	0.230	0.250	0.170
NR-PPAR-γ	0.500	0.800	0.150	0.370	0.550	0.160
SR-HSE	0.530	0.500	0.630	0.460	0.450	0.440

Tabla 15: Resultado de las métricas al realizar la búsqueda de hiperparámetros por separado para cada tarea.

	Modelo general			*Modelo optimizado*		
	F_2 score	**Recall**	**Precision**	**F_2 score**	**Recall**	**Precision**
NR-AR-LBD	0.091	0.125	0.044	**0.230**	0.250	0.170
NR-PPAR-γ	0.349	0.484	0.160	**0.370**	0.550	0.160
SR-HSE	0.401	0.500	0.225	**0.460**	0.450	0.440

Tabla 16: Comparación de métricas evaluadas en el set de **testeo final** para el modelo optimizado frente al F_2-Macro score y para el modelo optimizado para cada tarea por separado.

El conjunto de NR-AR-LBD obtuvo un F_2 score de 0.60 en el set Leaderboard con valores de sensibilidad de 0.75 y de precisión de 0.33. Con el modelo general anterior, el F_2 score obtenido en Leaderboard fue de 0.47 con el mismo valor de sensibilidad pero una precisión de 0.19. Se nota una mejora de la precisión y por ende del F_2 score, presentando un valor mayor que el promedio general anterior. A la vez, el F_2 score de 0.60 supera al obtenido en el entrenamiento Single-task de esta tarea (0.52). Con este modelo se evaluó al conjunto de prueba final y se obtuvo un F_2 score de 0.23 con respecto al 0.10 obtenido en el modelo general. Esto indica una mejora en el resultado de la clasificación del conjunto final de prueba, aunque los valores siguen siendo bajos. Esto último puede deberse a que esta tarea es la más desbalanceada del set de prueba teniendo solo un 1 % de compuestos activos. Para este caso, se podría pensar otra alternativa de entrenamiento considerando otras técnicas como por ejemplo el sobre-sampleo de la clase minoritaria que ha mostrado resultados satisfactorios en otros problemas con alto desbalance de clases [56].

Con respecto al set de NR-PPAR-γ, se obtuvo un F_2 score de 0.50 en el conjunto Leaderboard frente a 0.42 obtenidos con el modelo general. Para el set de prueba final se vieron minimos cambios en el score (0.35 con el modelo general y 0.37 con el modelo

optimizado). Se nota una leve mejora en el valor del recall.

En el conjunto SR-HSE se eligió un modelo que obtuvo un F_2 score de 0.53 en Leaderboard con respecto a 0.47 obtenido con el modelo general. Se ve una gran mejora en la precisión pasando de 0.26 en el modelo general a 0.63 en el modelo optimizado. Con respecto al set de prueba final también se ve una mejora pasando de un F_2 score de 0.40 a 0.46 (superando ahora el resultado del entrenamiento Single-Task) y de una precisión de 0.22 a 0.44.

Estos resultados demuestran que seleccionar los hiperparámetros por separado para cada tarea mejoran los resultados del F_2 score de las mismas y podría ser una manera más adecuada de resolver un problema de entrenamiento multi-tarea aunque esto lleve más costo computacional. En particular en estos tres casos se vio una mejora significativa en la precisión.

Capítulo 4

4. Discusión y conclusiones

La predicción computacional de la toxicidad de las NME sigue siendo una tarea desafiante debido a la gran cantidad de factores y variables que se involucran en la misma, especialmente debido a la complejidad del entorno biológico. A raíz de los resultados obtenidos por el grupo ganador del Tox21 Data Challenge, los modelos de Deep Learning han cobrado mucho interés para ser utilizados en la clasificación *in silico* de compuestos tóxicos. De esta manera se busca reducir los tiempos y ahorrar dinero en el proceso de descubrimiento de fármacos. En este trabajo se evaluó el desempeño de estos modelos según el F_2 score, métrica que da más importancia a la clasificación de la clase positiva (tóxica). Tanto la métrica como la función de costo se implementaron de forma tal de lidiar con los datos faltantes debido a que no todos los compuestos disponibles en la base de datos Tox 21 se midieron para todas las tareas.

Al diseñar un modelo clasificador de DL MTL se obtuvo un valor de F_2 score promedio de 0.40 en la evaluación del conjunto de prueba final con un score máximo de 0.70. Esto se obtuvo, en general, a partir de una sensibilidad alta y una precisión baja. Es de esperar que la sensibilidad sea mayor que la precisión debido a un factor $\beta > 1$. Al observar la matriz de confusión de las tareas también se corrobora un valor bajo de falsos negativos que son los que se buscan minimizar prioritariamente.

El modelo implementado no logró aprender adecuadamente la tarea NR-AR-LBD del conjunto de prueba final que tiene únicamente ocho compuestos positivos y hay solo un 3 % de activos en el conjunto de entrenamiento. Por lo tanto, el espacio químico del conjunto de entrenamiento no llegó en este caso a ser representativo de los compuestos tóxicos del conjunto de prueba final. Lo mismo ocurrió para el caso de NR-AR, tarea

que también se encuentra muy desbalanceada. Es interesante notar que esta última tarea mencionada tampoco logró un desempeño de ROC-AUC adecuado (ROC-AUC = 0.3) por el grupo DeepTox en el set Leaderboard. Se puede notar a partir de los resultados que el entrenamiento de las redes se encuentra sesgado hacia las tareas que tienen mas compuestos de clase tóxica, siendo éstas las que obtienen mejores resultados. Al entrenar todas las tareas en conjunto, los batches de entrenamiento reciben más datos tóxicos de estas tareas y por ende su aprendizaje de las mismas es mayor. Una alternativa para resolver este problema podría ser entrenar la red con batches que contengan forzadamente más compuestos tóxicos de las tareas minoritarias.

Respondiendo a uno de los objetivos específicos de este trabajo y corroborando lo dispuesto por previas investigaciones se nota que, en general, el entrenamiento múltiple mejora el desempeño de los modelos de DL ya que la red logra transferir características aprendidas en un task a otro donde esos datos faltan. Para el conjunto Leaderboard, en nueve de las 12 tareas se obtuvieron mejores resultados con el entrenamiento MTL. El entrenamiento múltiple mostró aún mejores resultados al realizar una búsqueda de hiperparámetros separada para cada tarea. En el caso de este trabajo, entrenar juntos a los tasks más correlacionados entre sí no necesariamente aumentó la performance del modelo.

El conjunto Leaderboard se entrenó utilizando distintas funciones de costo y se nota que, en promedio, los resultados son levemente mejores al minimizar directamente la métrica de interés. De todas formas, utilizar una función de costo estándar como la Binary Cross Entropy Weighted permite obtener resultados similares.

Por otro lado, se obtuvo un ranking de importancia de descriptores que permite una mejor comprensión de cuáles son las características a visualizar en un compuesto a la hora de predecir su toxicidad en un medio biológico. Los modelos de DL podrían advertir nuevas características que se relacionen con alguna actividad tóxica en particular

y que no eran conocidas previamente. En este trabajo se observó que la presencia de grupos tiocianatos fue la característica más ponderada por la red a la hora de establecer la toxicidad de un compuesto. También entre las características más importantes para la clasificación se encontró a la cantidad presente de dobles enlaces en la estructura de un compuesto y la presencia de agentes metálicos. Muchas veces algunos compuestos químicos pueden ser modificados para disminuir su toxicidad sin afectar su efectividad para actuar contra un blanco determinado. Analizar y conocer las características que afectan más a la clasificación tóxica de las redes neuronales es importante ya que permite conocer de forma localizada qué estructuras generan la toxicidad y de esta manera modificar los compuestos de forma que disminuya su efecto tóxico.

Con respecto al objetivo principal del trabajo, los resultados adecuados de sensibilidad obtenidos demuestran que DL es una herramienta potencial para ser utilizada en la toxicología computacional ya que permite identificar características tóxicas en compuestos biológicos. En particular, en un futuro cercano esta técnica podría ser incorporada en el pipeline de descubrimiento de fármacos, como un filtro adicional de moléculas previo al proceso de VS. De todas formas, es necesario continuar la investigación para seguir optimizando el uso de estos modelos en toxicología, especialmente para mejorar los resultados de las métricas adecuadas y tener clasificadores más efectivos y confiables para detectar compuestos tóxicos.

5. Trabajo a futuro

Como se observa de los resultados obtenidos para los tasks NR-AR-LBD, NR-PPAR-γ y SR-HSE, una forma de mejorar los resultados es realizando una búsqueda de hiper-parámetros separada para cada tarea. En particular, esto permitió mejores resultados en la precisión. En el futuro se podría implementar esta alternativa para las 12 tareas y ver si en general permite mejorar los resultados. También podrían probarse distintos valores del factor β y evaluar como los mismos afectan la matriz de confusión. Si bien se prioriza minimizar los falsos negativos (clasificar a un compuesto no tóxico cuando sí lo es), tener una gran cantidad de falsos positivos tampoco es deseable ya que se están descartando candidatos posibles a nuevos fármacos. Existen a su vez otras técnicas que no se implementaron en este trabajo como el sobre-sampleo de la clase minoritaria que podrían probarse como alternativas para mejorar los resultados.

Por otro lado, sería interesante en trabajos futuros aplicar esta herramienta para realizar un análisis mas local que permita distinguir las zonas, grupos o características de un compuesto que generan la toxicidad y que podrían ser modificados para obtener un compuesto no tóxico. Mirando las características a las que se les asignó más importancia para la clasificación, se podría evaluar qué valores de las mismas o umbrales son los que producen la toxicidad. Por ejemplo, eliminando ciertos grupos de una molécula, agregando o quitando enlaces o elementos, aumentando o disminuyendo su lipofilicidad o cambiando otras características de un compuesto se podría ver como estos cambios se reflejan en la predicción de toxicidad.

Apéndice A

A. Modelos optimizados para cada tarea por separado

En el caso de NR-AR-LBD, se seleccionó un modelo de tres capas ocultas con 128,256 y 308 neuronas respectivamente en cada capa, una capa de entrada de 512 neuronas (modelo triangular) y una capa de salida de 12 neuronas. La capa de salida tiene una activación sigmoidea mientras que la capa de entrada y las ocultas tienen una activación ReLU al igual que en el modelo general. La tasa de aprendizaje que obtuvo los mejores resultados fue de 0.01 para el optimizador Adam. Como regularización, se tienen capas de Dropout de 0.4 para la capa de entrada y 0.2 en las capas ocultas y un valor de regularización l2 de 0.001. Además, como inicializador se encontró que el Glorot Uniform fue el que obtuvo los mejores resultados.

Para NR-PPAR-γ, el modelo que obtuvo el mejor desempeño consistió también en un modelo triangular de una capa de entrada de 512 neuronas y 3 capas ocultas de 128,256 y 308 neuronas respectivamente. En este caso se utilizó una tasa de aprendizaje de 0.05. Los valores de Dropout, de regularización l2 e inicializadores fueron los mismos que para el modelo anterior de NR-AR-LBD. La función de activación de la capa de entrada y ocultas en este caso también fue ReLu.

Con respecto a SR-HSE, se eligió un modelo con 1024 neuronas en la capa de entrada y 256, 512 y 368 neuronas en las 3 capas ocultas. En este caso se utilizó una tasa de Dropout de 0.3 en la capa de entrada y 0.15 en las ocultas. La tasa de aprendizaje fue de 0.05. El resto de los hiperparámetros se mantuvieron con respecto al modelo general.

Apéndice B

B. Código

B.1. Implementación de funciones de costo y métricas

En esta sección se muestran las funciones implementadas en Python para el cálculo de métricas y funciones de costo empleadas sobre un conjunto de datos multi-tarea con etiquetas faltantes:

- La función best_fbeta calcula el F_β score para el umbral óptimo de separación de clases.
- La función multi_label_fbeta calcula el F_β score con el umbral óptimo para cada tarea. Devuelve los umbrales óptimos y scores para cada tarea y el score promedio.
- multi_label_conf_matrix calcula la matriz de confusión para cada tarea.
- Las funciones multi_precision y multi_recall calculan la precisión y el recall para cada tarea respectivamente.
- La función multi_fbeta_test calcula el F_β sobre el conjunto de prueba final usando el umbral óptimo de separación de clases del conjunto Leaderboard.
- Por último, se muestra la implementación de las funciones de costo mencionadas en el texto del trabajo.

[ ]:
```
from matplotlib import pyplot as plt
from sklearn.metrics import fbeta_score


# Funcion para calcular el Fbeta score con distintos
#umbrales de separacion de clases y elegir el optimo.
```

```
def best_fbeta(y_true,y_pred,plot=False,beta=2):
    thresholds = np.arange(100)/100
    scores = np.zeros(thresholds.shape)

    for i, thr in enumerate(thresholds.tolist()):
        y_pred_class = (y_pred>thr)*1
        scores[i] = fbeta_score(y_true,y_pred_class,beta=beta)

    best_score = np.max(scores)
    best_score_idx = np.argmax(scores)
    best_thr = thresholds[best_score_idx]

    if plot:
        plt.plot(thresholds,scores)
        plt.axvline(x=best_thr,color='r')
        plt.xlabel('threshold')
        plt.ylabel('fbeta_score')
    return [best_thr,best_score]
```

```
[ ]: #Funcion para calcular el Fbeta score en un conjunto multi-label
     #con datos faltantes.

     def multi_label_fbeta(y_true, y_pred, plot=False, beta=2):
         fbeta_perlabels = []
         thr_perlabels = []
```

```
    for i in range(y_pred.shape[1]): #calculo el score para cada label␣
↪sin contar los -1:
        pred_label = y_pred[:,i]
        true_label = y_true[:,i]
        # true_label = np.array(true_label)
        index = np.where(true_label == -1)
        pred_label = np.delete(pred_label, index)
        true_label = np.delete(true_label, index)

        th_fbeta, score_fbeta_label = best_fbeta(true_label, pred_label,␣
↪plot=plot, beta=beta)
        fbeta_perlabels.append(score_fbeta_label)
        thr_perlabels.append(th_fbeta)

    # score_auc = np.mean(auc_perlabels)
    mean_score_fbeta = np.mean(fbeta_perlabels)
    return mean_score_fbeta, fbeta_perlabels, thr_perlabels
```

[ ]:
```
#Funcion para calcular las Matrices de Confusion en un conjunto␣
↪multi-label
#con datos faltantes.

from sklearn.metrics import confusion_matrix

def multi_label_conf_matrix(y_true, y_pred, thr):
    # dtype = K.floatx()
```

```
    # mask = K.cast(K.not_equal(y_true, mask_value), dtype)
    # mask = mask.eval(session=tf.compat.v1.Session())
    conf_matrices = []
    for i in range(y_pred.shape[1]): #calculo el score para cada label␣
↪sin contar los -1:
      pred_label = y_pred[:,i]
      true_label = y_true[:,i]

      pred_label = (pred_label>thr[i])*1

      # true_label = np.array(true_label)
      index = np.where(true_label == -1)
      pred_label = np.delete(pred_label, index)
      true_label = np.delete(true_label, index)
      confusion_matlabel = confusion_matrix(true_label, pred_label)
      conf_matrices.append(confusion_matlabel)
    return conf_matrices
```

[ ]:
```
#Funcion para calcular la Precision y el Recall en un conjunto␣
↪multi-label
#con datos faltantes.

from sklearn.metrics import precision_score, recall_score

def multi_precision(y_true, y_pred, thr):
    # dtype = K.floatx()
    # mask = K.cast(K.not_equal(y_true, mask_value), dtype)
```

```
    # mask = mask.eval(session=tf.compat.v1.Session())
    precision = []
    for i in range(y_pred.shape[1]): #calculo el score para cada label␣
 ↪sin contar los -1:
      pred_label = y_pred[:,i]
      true_label = y_true[:,i]

      pred_label = (pred_label>thr[i])*1

      # true_label = np.array(true_label)
      index = np.where(true_label == -1)
      pred_label = np.delete(pred_label, index)
      true_label = np.delete(true_label, index)
      precision_label = precision_score(true_label, pred_label)
      precision.append(precision_label)
    return precision


def multi_recall(y_true, y_pred, thr):
    # dtype = K.floatx()
    # mask = K.cast(K.not_equal(y_true, mask_value), dtype)
    # mask = mask.eval(session=tf.compat.v1.Session())
    recall = []
    for i in range(y_pred.shape[1]): #calculo el score para cada label␣
 ↪sin contar los -1:
      pred_label = y_pred[:,i]
```

```
        true_label = y_true[:,i]

        pred_label = (pred_label>thr[i])*1

        # true_label = np.array(true_label)
        index = np.where(true_label == -1)
        pred_label = np.delete(pred_label, index)
        true_label = np.delete(true_label, index)
        recall_label = recall_score(true_label, pred_label)
        recall.append(recall_label)
    return recall

def multi_fbeta_test(y_true, y_pred, thr):
    # dtype = K.floatx()
    # mask = K.cast(K.not_equal(y_true, mask_value), dtype)
    # mask = mask.eval(session=tf.compat.v1.Session())
    fbeta_test = []
    for i in range(y_pred.shape[1]): #calculo el score para cada label␣
↪sin contar los -1:
        pred_label = y_pred[:,i]
        true_label = y_true[:,i]

        pred_label = (pred_label>thr[i])*1

        # true_label = np.array(true_label)
        index = np.where(true_label == -1)
```

```
      pred_label = np.delete(pred_label, index)
      true_label = np.delete(true_label, index)
      fbetat_label = fbeta_score(true_label, pred_label,beta=2)
      fbeta_test.append(fbetat_label)
    return fbeta_test
```

[ ]:
```
#Funcion para calcular el AUC-PR score en un conjunto multi-label con datos faltantes.

from sklearn.metrics import average_precision_score

def multi_label_aucpr(y_true, y_pred, plot=False):
    auc_perlabels = []
    for i in range(y_pred.shape[1]): #calculo el score para cada label sin contar los -1:
      pred_label = y_pred[:,i]
      true_label = y_true[:,i]
      # true_label = np.array(true_label)
      index = np.where(true_label == -1)
      pred_label = np.delete(pred_label, index)
      true_label = np.delete(true_label, index)
      score_auc_label = average_precision_score(true_label, pred_label)
      auc_perlabels.append(score_auc_label)

    # if plot:
    #     plt.plot(thresholds,scores)
    #     plt.axvline(x=best_thr,color='r')
```

```
    #     plt.xlabel('threshold')
    #     plt.ylabel('fbeta_score')

    mean_score_auc = np.mean(auc_perlabels)
    return mean_score_auc, auc_perlabels
```

[ ]:
```
#Funcion para calcular el Accuracy en un conjunto multi-label con␣
 ↪datos faltantes.

def masked_accuracy(y_true, y_pred):
    dtype = K.floatx()
    total = K.sum(K.cast(K.not_equal(y_true, mask_value), dtype))
    correct = K.sum(K.cast(K.equal(y_true, K.round(y_pred)), dtype))
    return correct / total
```

[ ]:
```
#Funcion para calcular la BCE Loss en un conjunto multi-label con␣
 ↪datos faltantes.

mask_value = -1
def masked_loss_function(y_true, y_pred):
    y_true=tf.cast(y_true, dtype=tf.float32)
    y_pred=tf.cast(y_pred, dtype=tf.float32)
    mask = K.cast(K.not_equal(y_true, mask_value), K.floatx())
    return K.binary_crossentropy(y_true*mask, y_pred*mask)
```

[ ]:
```
#Funcion para calcular la BCE Weighted Loss en un conjunto multi-label␣
 ↪
#con datos faltantes.
```

```
mask_value = -1
def weighted_binary_crossentropy(y_true, y_pred):

        # Original binary crossentropy (see losses.py):
        # K.mean(K.binary_crossentropy(y_true, y_pred), axis=-1)

        # Calculate the binary crossentropy
        # y_true=tf.cast(y_true, dtype=tf.float32)
        # y_pred=tf.cast(y_pred, dtype=tf.float32)
        mask = K.cast(K.not_equal(y_true, mask_value), K.floatx())
        b_ce = K.binary_crossentropy(y_true*mask, y_pred*mask)

        # Apply the weights
        weight_vector = y_true * 0.6 + (1. - y_true) * 0.4
        weighted_b_ce = weight_vector * b_ce

        # Return the mean error
        return K.mean(weighted_b_ce)
```

[ ]:
```
#Funcion para calcular la F2 Loss en un conjunto multi-label con datos␣
 ↪faltantes.

def masked_f2(y_true, y_pred):
    # y_true = K.cast(y_true, 'float')
    # y_pred = K.cast(y_pred, 'float')
    val_pred = K.cast(K.greater(y_pred,0.2), 'float')
```

```
    # val_pred = K.round(y_pred)
    mask = K.cast(K.not_equal(y_true, -1.0), 'float')



    tp = K.sum(K.cast(y_true*mask*val_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*mask*(1-val_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*mask*val_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*mask*(1-val_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f1 = 5*p*r / (4*p+r+K.epsilon())
    f1 = tf.where(tf.math.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)

def f2_loss(y_true, y_pred):
  # beta = 2
    # y_true = K.cast(y_true, 'float')
    # y_pred = K.cast(y_pred, 'float')
    mask = K.cast(K.not_equal(y_true, -1.0), 'float')

    tp = K.sum(K.cast(y_true*mask*y_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*mask*(1-y_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*mask*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*mask*(1-y_pred), 'float'), axis=0)
```

```
    p = tp / (tp + fp + K.epsilon())
    r = tp / (tp + fn + K.epsilon())

    f2 = 5*p*r / (4*p+r+K.epsilon())
    f2 = tf.where(tf.math.is_nan(f2), tf.zeros_like(f2), f2)

    return 1 - K.mean(f2)
```

[ ]:
```
#Callback para el calculo de las metricas durante el entrenamiento
#de los modelos de DL con Keras.

from tensorflow.keras.callbacks import Callback

class MetricsEvaluation(Callback):

    def __init__(self, validation_data =(), interval=10):
        super(Callback, self).__init__()
        self.X_val, self.y_val = validation_data
        self.interval = interval #cada cuantos epochs evaluar las␣
 ↪metricas
        self.auc_pr_scores=[]
        self.fbeta_scores=[]

    def on_train_begin(self,  logs={}):
        self.val_auc_pr = []
        self.val_fbeta = []
```

```
    def on_epoch_end(self, epoch, logs={}):
        if epoch % self.interval == 0:
            y_pred = self.model.predict_proba(self.X_val , verbose=0)
            # print(y_pred)
            mean_score_auc, auc_perlabels = multi_label_aucpr(self.
↪y_val, y_pred)
            mean_score_fbeta, fbeta_perlabels, thr_perlabels =␣
↪multi_label_fbeta(self.y_val, y_pred)
            self.val_auc_pr.append(mean_score_auc)
            self.val_fbeta.append(mean_score_fbeta)
            logs["f_beta"]=mean_score_fbeta
            logs["auc_pr"]=mean_score_auc
            print("interval evaluation - epoch: {:d} - score: {:.6f},
                {:.6f}".format(epoch, mean_score_auc, mean_score_fbeta))

    def on_train_end(self, logs={}):
        self.auc_pr_scores.append(np.max(self.val_auc_pr))
        self.fbeta_scores.append(np.max(self.val_fbeta))

# metrics_callback = MetricsEvaluation(validation_data=(X_val, y_val),␣
↪ interval=1)
```

B.2. Selección del modelo final

B.2.1. Búsqueda de hiperparámetros con Talos

A continuación se muestra el código implementado para la búsqueda de hiperparámetros por Cross Validation utilizando la librería Talos. La función tox21_model es utilizada por la librería mencionada y permite generar modelos de DL a partir de un diccionario de hiperparámetros que se quieren evaluar.

[ ]:
```
def tox21_model(x,y,params):
    model = Sequential()

    # Input layer
    model.add(Dense(params['first_neuron'], input_shape = (X_train.
↪shape[1],),
                    kernel_initializer=params['kernel_initializer']
                    ))

    #Input Activation:
    if params['activation']=='prelu':
            model.add( PReLU(alpha_initializer =␣
↪Constant(value=params['alpha_initializer']) ))
    elif params['activation']=='elu':
            model.add(ELU(alpha=0.5))
    elif params['activation']=='leaky_relu':
            model.add(LeakyReLU())
    else:
            model.add(Activation(params['activation']))
```

```
    # Input Dropout layer
    if(params['dropout']>0):
        model.add(Dropout(params['dropout']))

    # Hidden layers
    layer_neurons = network_shape(params,1)

    for i in range(params['hidden_layers']):

        # Normalization layer
        if params['batch_norm']==True:
            model.add(BatchNormalization())

        # Dense layer
        model.add(Dense(layer_neurons[i],
                        kernel_initializer=params['kernel_initializer'],
                        kernel_regularizer =␣
↪params['kernel_regularizer'],
                        activity_regularizer =␣
↪params['activity_regularizer'],
                        use_bias=True))

        # Activation function
        if params['activation']=='prelu':
            model.add( PReLU(alpha_initializer =␣
↪Constant(value=params['alpha_initializer']) ))
```

```
        elif params['activation']=='elu':
            model.add(ELU(alpha=0.5))
        elif params['activation']=='leaky_relu':
            model.add(LeakyReLU())
        else:
            model.add(Activation(params['activation']))

        # Dropout layer
        if(params['dropout']>0):
            model.add(Dropout(params['dropout']/2))

    # Output Layer
    model.add(Dense(12,␣
↪kernel_initializer='he_uniform',bias_initializer=output_bias, ␣
↪activation = 'sigmoid'))


    # Optimizer
    opt = Adam(
                learning_rate=lr_normalizer(params['lr'],Adam),
                beta_1=0.9,
                beta_2=0.999,
                epsilon=1e-7,
                amsgrad=False)

    # opt = SGD(learning_rate=lr_normalizer(params['lr'],SGD))
```

```
    model.compile(loss=f2_loss, optimizer=opt ,metrics=[masked_f2])#,

    return model

def tox21_talos(x, y, x_v, y_v, params):

    # Create model
    model = tox21_model(x,y,params)

    print("_____________________________________________________________________")
    for key in params:
        print(f"{key:20}{params[key]}")

    print(model.summary())

    # logdir = /content/drive/logs

    es = EarlyStopping(monitor='f_beta',mode='max', verbose=1,
    patience=25, restore_best_weights = True)

    reduceLR=ReduceLROnPlateau(monitor='val_loss',
                               factor=0.2,
                               patience=8,
                               verbose=1,
```

```
                            min_delta=0,
                            cooldown=0, min_lr=0)
    # tb = TensorBoard(logdir, histogram_freq=1)

    history = model.fit(x, y, validation_data = (x_v, y_v),
                        batch_size = params['batch_size'],
                        epochs = params['epochs'],
                        class_weight = params['class_weight'],
                        callbacks=[metrics_callback, es,reduceLR],␣
↪#,TensorBoard(logdir, histogram_freq=1)
                        verbose=2)

    return history, model
```

[ ]:
```
p = {
    'first_neuron': [512, 1024, 3500, 4096],                # Tamaño␣
↪de la primera capa
    'dropout': [0, 0.3, 0.4, 0.5],                    # Valor de␣
↪dropout luego de la primera capa
    'activation': ['relu'],                 # Función de activación
    'alpha_initializer': [0.2],              # PRELU alpha
    'batch_size': [256],#, 512],                       # Tamaño de batch
    'epochs': [400],                         # Number of epochs
    'batch_norm': [True],                    # Batch normalization␣
↪layers
    'hidden_layers':[3, 4],                     # Number of hidde layers
```

```
    'shapes':['brick','triangle', 'funnel'],                    # Shape of hidden layers
    'optimizer': [Adam],    # Optimizer
    'class_weight': [None],
    'kernel_regularizer':[l2(l2=0.001)],            # Kernel (weights) regularization
    #'bias_regularizer':[None],              # Bias (offsets) regularization
    'activity_regularizer':[None],          # Activity (output) regularization
    'kernel_initializer': ['random_normal'], # Kernel (weights) initializer
    'lr': [0.01]                              # Learning rate
}
```

[ ]:
```
p = {
    'first_neuron': [1024],                 # Tamaño de la primera capa
    'dropout': [ 0.3],                    # Valor de dropout luego de la primera capa
    'activation': ['relu', 'PreLu', 'tanh'],                # Función de activación
    'alpha_initializer': [0.2],             # PRELU alpha
    'batch_size': [256],#, 512],                    # Tamaño de batch
    'epochs': [400],                        # Number of epochs
    'batch_norm': [True],                   # Batch normalization layers
```

```
    'hidden_layers':[3],                        # Number of hidde layers
    'shapes':['triangle'],                         # Shape of hidden layers
    'optimizer': [Adam],     # Optimizer
    'class_weight': [None],
    'kernel_regularizer':[l2(l2=0.01), l2(l2=0.001), l2(l2=0.0001)],#␣
↪Kernel (weights) regularization
    #'bias_regularizer':[None],               # Bias (offsets)␣
↪regularization
    'activity_regularizer':[None],           # Activity (output)␣
↪regularization
    'kernel_initializer': ['random_normal', 'glorot_uniform',␣
↪'he_uniform'], # Kernel (weights) initializer
    'lr': [0.01]                                 # Learning rate
}
```

[ ]:
```
%matplotlib inline
%reload_ext tensorboard


scan = talos.Scan(x=X_train,
                  y=y_train,
                  model=tox21_talos,
                  experiment_name='tox21_scan_0203',
                  #fraction_limit=0.004,
                  #round_limit = 200,
                  random_method = 'uniform_mersenne' ,
                  x_val=X_val,
```

```
                    y_val=y_val,
                    params=p,
                    print_params=False,
                    save_weights = True)

#El output de esta funcion se guarda en un archivo csv que permite␣
 ↪visualizar
#los resultados de las metricas para cada combinacion de HP.
```

B.2.2. Cross Validation con Talos

En esta subsección se muestra la implementación del método de Cross Validation en el cual el conjunto de entrenamiento y el conjunto Leaderboard se dividen en tres sub conjuntos diferentes de entrenamiento y validación, para los cuales se evalúa el Fβ-Macro score, como se explica en la sección 3.1.

Se obtiene un archivo csv con las distintas combinaciones de hiperparámetros y métricas para cada conjunto de entrenamiento-validación.

[ ]:
```
from iterstrat.ml_stratifiers import MultilabelStratifiedShuffleSplit,␣
 ↪MultilabelStratifiedKFold
```

[ ]:
```
msss = MultilabelStratifiedShuffleSplit(n_splits=2, test_size=0.2,␣
 ↪random_state=0)
folds = msss.split(descriptorss, tox21_targets.values)
```

[ ]:
```
lista =[]
for i,j in folds:
  lista.append((i,j))
```

[ ]:
```
#Separacion del set de Leaderboard:


train_index = np.arange(11982-293)
test_index = np.arange(11689, 11982)


it =(train_index, test_index)
```

[ ]:
```
lista.append(it)


generator = iter(lista) #Generador con los indices de los 3 distintos␣
 ↪conjuntos
#de Train-Val para Cross Validation. El ultimo conjunto es el set de␣
 ↪Leaderboard.
```

[ ]:
```
from sklearn.preprocessing import StandardScaler
X = descriptorss.values
y = tox21_targets2.values.astype(np.float)
mean_fbetas = []
# fbeta_labels =[]


output_bias = tf.keras.initializers.Constant(initial_bias)


for train_in, test_in in generator:
  X_train, X_val = X[train_in], X[test_in]
  y_train, y_val = y[train_in], y[test_in]


  #Normalizacion:
```

```
desc_scaler = StandardScaler()
X_train = desc_scaler.fit_transform(X_train)
X_val = desc_scaler.transform(X_val)

metrics_callback = MetricsEvaluation(validation_data=(X_val, y_val), ␣
↪interval=1)

#Talos Scan:
%matplotlib inline
%reload_ext tensorboard
scan = talos.Scan(x=X_train,
                  y=y_train,
                  model=tox21_talos,
                  experiment_name='tox21_scan1',
                  #fraction_limit=0.004,
                  #round_limit = 200,
                  random_method = 'uniform_mersenne' ,
                  x_val=X_val,
                  y_val=y_val,
                  params=p,
                  print_params=False,
                  save_weights = False)

mean_fbetas.append(metrics_callback.fbeta_scores)
```

B.3. Importancia de descriptores moleculares

En esta sección se muestra el código implementado para obtener el ranking de importancia de los descriptores moleculares. Para ello se utilizaron las librerías Eli5 y SkLearn.

[ ]:
```
#Funcion para calcular el F2 score en un conjunto de datos multi-label con datos
#faltantes. Se utiliza 0.2 como umbral de separacion de clases, ya que es
#el umbral optimo promedio para la separacion del conjunto Leaderboard con el
#modelo final seleccionado.

def masked_f2_score(clf, x_val, y_true):
    y_pred = clf.predict_proba(x_val)
    # y_true = K.cast(y_true, 'float')
    # y_pred = K.cast(y_pred, 'float')
    val_pred = K.cast(K.greater(y_pred,0.2), 'float')
    # val_pred = K.round(y_pred)
    mask = K.cast(K.not_equal(y_true, -1.0), 'float')


    tp = K.sum(K.cast(y_true*mask*val_pred, 'float'), axis=0)
    tn = K.sum(K.cast((1-y_true)*mask*(1-val_pred), 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*mask*val_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*mask*(1-val_pred), 'float'), axis=0)

    p = tp / (tp + fp + K.epsilon())
```

```
    r = tp / (tp + fn + K.epsilon())

    f1 = 5*p*r / (4*p+r+K.epsilon())
    f1 = tf.where(tf.math.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)
```

[ ]:
```
perm = PermutationImportance(keras_estimator, cv = 'prefit',␣
↪refit=False, random_state=1, n_iter = 12, scoring = masked_f2_score).
↪fit(X_val,y_val)
```

[ ]:
```
features = features.reshape(2393,)
```

[ ]:
```
pesos = eli5.show_weights(perm, feature_names = features)
```

[ ]:
```
#Top Ranking de Descriptores Moleculares:


pesos
```

[ ]: <IPython.core.display.HTML object>

[ ]:
```
importancias = perm.feature_importances_
args= importancias.argsort()
```

[ ]:
```
feature_names = Scaled_descriptors.columns.tolist()
features_sorted = np.array(features)[args]
```

[ ]:
```
features_sorted
```

[ ]:
```
array(['GATS5Z', 'VE3_Dzm', 'AATS5p', ..., 'SubFPC165', 'AATS0m',
       'MACCSFP112'], dtype=object)
```

Referencias

[1] Phrma. Biopharmaceutical research & development: The process behind new medicines brochure. http://phrma-docs.phrma.org/sites/default/files/pdf/rd_brochure_022307.pdf, 2015.

[2] Jean-Louis Reymond, Ruud Deursen, Lorenz Blum, and Lars Ruddigkeit. Chemical space as a source for new drugs. *MedChemComm*, 1, 2010.

[3] S. M. Paul, D. S. Mytelka, C. T. Dunwiddie, C. C. Persinger, B. H. Munos, S. R. Lindborg, and A. L. Schacht. How to improve rd productivity: the pharmaceutical industry's grand challenge. *Nature Reviews Drug Discovery*, 9(3):203–214, 2010.

[4] Claudio Cavasotto and Andew W. Orry. Ligand docking and structure-based virtual screening in drug discovery. *Current Topics in Medicinal Chemistry*, 7(10):1006–1014, 2007.

[5] Andrey A. Toropov, Alla P. Toropova, Ivan Raska Jr, Danuta Leszczynska, and Jerzy Leszczynski. Comprehension of drug toxicity: Software and databases. *Elseiver. Computers in Biology and Medicine*, 45:20–25, 2014.

[6] Genome Research Limited. https://www.yourgenome.org/facts/how-are-drugs-designed-and-developed.

[7] B. G. Katzung and A. J. Trevor. *Basic and Clinical Pharmacology 13 E.* McGraw-Hill Education/Medical New York, 2014.

[8] BIO Industry Analysis. Clinical development succes rates 2006-2015. https://www.bio.org/sites/default/files/legacy/bioorg/docs/Clinical%20Development%20Success%20Rates%202006-2015%20-%20BIO,%20Biomedtracker,%20Amplion%202016.pdf, 2016.

[9] M. E. Andersen and D. Krewski. Toxicity testing in the 21st century: Bringing the vision to life. *Toxicological Sciences*, 107(2):324–330, 2008.

[10] D. Krewski, D. Acosta, M. Andersen, H. Anderson, J. C. Bailar, and K. Boekelheide. Toxicity testing in the 21st century: A vision and a strategy. *Journal of Toxicology and Environmental Health*, 13(2-4):51–138, 2010.

[11] M. A. Dorato and L. A. Buckley. Toxicology testing in drug discovery and development. *Current Protocols in Toxicology*, Chapter 19, 2007.

[12] Jeffrey Kramer, John Sagartz, and Dale Morris. The application of discovery toxicology and pathology towards the design of safer pharmaceutical lead candidates. *Nature reviews. Drug discovery*, 6(8):636–49, 2007.

[13] Ton J. Cleophas and Aeilko H. Zwinderman. *Machine Learning in Medicine – a complete overview*. Springer, 2015.

[14] Claudio N Cavasotto and Juan I Di Filippo. Artificial intelligence in the early stages of drug discovery. *Archives of Biochemistry and Biophysics*, 698:108730, 2021.

[15] E. B. Lenselink, N. Ten Dijke, B. Bongers, G. Papadatos, H. W. Van Vlijmen, W. Kowalczyk, A. P. IJzerman, and G. J. Van Westen. Beyond the hype: deep neural networks outperform established methods using a chembl bioactivity benchmark set. *Journal of cheminformatics*, 9:1–14, 2017.

[16] Igor I. Baskin. Machine learning methods in computational toxicology. *Methods in Molecular Biology(Clifton, N.J.)*, 1800:119–139, 2018.

[17] Luis G Valerio Jr. In silico toxicology models and databases as fda critical path initiative toolkits. *Human Genomics BMD.*, 5:200–7, 2011.

[18] Kit-Kay Mak and Mallikarjuna Rao Pichika. Artificial intelligence in drug development: present status and future prospects. *Drug Discovery Today*, 24(3):773–780,

2018.

[19] Huang Ruili, Xia Menghang, Nguyen Dac-Trung, Zhao Tongan, Sakamuru Srilatha, Zhao Jinghua, Shahane Sampada A., Rossoshek Anna, and Simeonov Anton. Tox21challenge to build predictive models of nuclear receptor and stress response pathways as mediated by exposure to environmental chemicals and drugs. *Frontiers in Environmental Science*, 3, 2016.

[20] Hartmut Jaeschke, Mitchell McGill, and Anup Ramachandran. Oxidant stress, mitochondria, and cell death mechanisms in drug-induced liver injury: lessons learned from acetaminophen hepatotoxicity. *Drug Metabolism Reviews*, 44:88–106, 2012.

[21] National Center for Advancing Translational Sciences (NCATS). Tox 21 data challenge 2014. https://tripod.nih.gov/tox21/challenge/index.jsp, 2014.

[22] Andreas Mayr, Günter Klambauer, Thomas Unterthiner, and Sepp Hochreiter. Deep tox: Toxicity prediction using deep learning. *Frontiers in Environmental Science*, 3:80, 2016.

[23] Jan Wenzel, Hans Matter, and Friedemann Schmidt. Predictive multitask deep neural network models for adme-tox properties: Learning from large data sets. *Journal of Chemical Information and Modelling*, 59, 2019.

[24] Y. Xu, Z. Dai, F. Chen, S. Gao, J. Pei, and L. Lai. Deep learning for drug-induced liver injury. *Journal of Chemical Information and Modeling*, 55(10):2085–2093, 2015.

[25] M Fernandez, F Ban, G Woo, M Hsing, T Yamazaki, E LeBlanc, and A. Cherkasov. Toxic colors: The use of deep learning for predicting toxicity of compounds merely from their graphic images. *Journal of Chemical Information and Modeling*, 58:1533–1543, 2018.

[26] Hughes TB and Swamidass SJ. Deep learning to predict the formation of quinone species in drug metabolism. *Chemical Research in Toxicology*, 30:642–656, 2017.

[27] Gabriel Idakwo, Sundar Thangapandian, Joseph Luttrell, Zhaoxian Zhou, Chaoyang Zhang, and Ping Gong. Deep learning-based structure-activity relationship modeling for multi-category toxicity classification: A case study of 10k tox21 chemicals with high-throughput cell-based androgen receptor bioassay data. *Frontiers in Physiology*, 10, 2019.

[28] Xiang Li, Yj Clinux, Luhua Lai, and Jianfeng Pei. Prediction of human cytochrome p450 inhibition using a multitask deep autoencoder neural network. *Molecular Pharmaceutics*, 15, 2018.

[29] I. Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 2000.

[30] Yann LeCun, Y. Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521:436–44, 2015.

[31] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[32] Limeng Pu, Misagh Naderi, Tairan Liu, Hsiao-Chun Wu, Supratik Mukhopadhyay, and Michal Brylinski. etoxpred: a machine learning-based approach to estimate the toxicity of drug candidates. *BMC Pharmacology and Toxicology*, 20, 2019.

[33] Sigma Aldrich. Lopac 1280 (library of pharmacologically active compounds). https://www.sigmaaldrich.com/catalog/product/sigma/lo1280?lang=es&region=AR.

[34] Juan Jose Berlanga Chiquero. Respuesta al estrés celular. implicaciones en envejecimiento y otras patologías relacionadas con la edad. *Departamento de Biología*

Molecular. Universidad Autónoma de Madrid, 2016.

[35] R.A. Abagyan, M.M. Totrov, and D.N. Kuznetsov. Icm - a new method for protein modeling and design. applications to docking and structure prediction from the distorted native conformation. *Journal of Computational Chemistry*, 15:488–506, 1994.

[36] N M O'Boyle, M Banck, C A James, C Morley, T Vandermeersch, and G R Hutchison. Open babel: An open chemical toolbox. *Journal of Cheminformatics.*, 2011.

[37] J. Godwin. Multi-task learning in tensorflow. https://jg8610.github.io/Multi-Task/, 2016.

[38] Michael Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv*, 2020.

[39] Dan C. Fara and Tudor I. Oprea. Basics: Molecular descriptors and fingerprints. http://datascience.unm.edu/biomed505/Course/Cheminformatics/basic/descs_fingers/.

[40] Joseph Durant, Burton Leland, Douglas Henry, and James Nourse. Reoptimization of mdl keys for use in drug discovery. *Journal of chemical information and computer sciences*, 42:1273–80, 2002.

[41] David Rogers and Mathew Hahn. Extended-connectivity fingerprints. *Journal of Chemical Information and Modeling*, 50:742–754, 2010.

[42] N. Rhodes, D. E. Clark, and P. Willett. Similarity searching in databases of flexible 3d structures using autocorrelation vectors derived from smoothed bounded distance matrices. *Journal of Chemical Information and Modeling*, 46:615–619, 2006.

[43] Yap CW. Padel-descriptor: An open source software to calculate molecular descriptors and fingerprints. *Journal of Computational Chemistry.*, 2011.

[44] Moriwaki H, Tian Y-S, Kawashita N, and Takagi T. Mordred: a molecular descriptor calculator. *Journal of Cheminformatics.*, 2018.

[45] Greg Landrum. Rdkit: Open-source cheminformatics. http://www.rdkit.org.

[46] Christopher A. Lipinski, Franco Lombardo, Beryl W. Dominy, and Paul J. Feeney. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Advanced Drug Delivery Reviews*, 64:4–17, 2012.

[47] Keras. https://keras.io/.

[48] Seyed Raein Hashemi, Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, Sanjay P Prabhu, Simon K Warfield, and Ali Gholipour. Asymmetric loss functions and deep denselyconnected networks for highly-imbalanced medical image segmentation: Application to multiple sclerosis lesion detection. *IEEE Access*, 7:1721–1735, 2019.

[49] Google Colaboratory. Introduction. https://colab.research.google.com/notebooks/welcome.ipynb.

[50] Autonomio talos [computer software]. http://github.com/autonomio/talos, 2019.

[51] Konstantinos Sechidis, Grigorios Tsoumakas, and Ioannis Vlahavas. *On the Stratification of Multi-label Data*. Machine Learning and Knowledge Discovery in Databases. Springer, 2011.

[52] Guangtao Wang, Qinbao Song, Heli Sun, Xueying Zhang, Baowen Xu, and Yuming Zhou. A feature subset selection algorithm automatic recommendation method. *Journal of Artificial Intelligence Research*, 47, 2013.

[53] Eduardo Gaitan. Goitrogens, environmental. thiocyanate. *Encyclopedia of Endocrine Diseases*, 2004.

[54] Kunal Roy and Gopinath Ghosh. Exploring qsars with extended topochemical atom (eta) indices for modeling chemical and drug toxicity. *Current pharmaceutical design*, 16:2625–39, 2010.

[55] Organización Mundial de la Salud. Diez sustancias químicas que constituyen una preocupación para la salud pública. https://www.who.int/ipcs/assessment/public_health/chemicals_phc/es/.

[56] Felix Lasta Georgios Douzas, Fernando Bacaoa. Improving imbalanced learning through a heuristic oversampling method based on k-means and smote. *Information Sciences.*, 2018.