Análisis y optimización de un autoencoder variacional semisupervisado para diseño molecular condicionado

Autor: Gonzalo Colmenarejo Sánchez
Máster Universitario en Bioinformática y Bioestadística
Área: Deep Learning

Consultor: Esteban Vegas Lozano

Fecha: 31-Diciembre-2018

Dedicado a Cristina, Teresa y Blanca

Dedicado a mi familia

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Análisis y optimización de un autoencoder variacional semisupervisado para diseño molecular condicionado*
Nombre del autor:	*Gonzalo Colmenarejo Sánchez*
Nombre del consultor/a:	*Esteban Vegas Lozano*
Nombre del PRA:	*Alexandre Sánchez Pla*
Fecha de entrega (mm/aaaa):	01/2019
Titulación::	*Máster Universitario en Bioinformática y Bioestadística UOC-UB*
Área del Trabajo Final:	*Deep Learning*
Idioma del trabajo:	*Español*
Palabras clave	*Deep Learning, Molecular Design, Variational Autoencoders*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

Se ha analizado y optimizado el reciente Autoencoder Variacional SemiSupervisado (SSVAE) de Kang y Chao (2018, J. Chem. Inf. Model., Article ASAP DOI: 10.1021/acs.jcim.8b00263) como modelo de Deep Learning de QSAR inverso para diseño molecular condicionado. El objetivo es caracterizar el output del modelo (corrección, diversidad y novedad, distribución de propiedades) en función de distintos factores: tamaño y diversidad del conjunto de entrenamiento, tamaño del output, tipo de conjunto de moléculas ("drug-like" vs productos naturales), y propiedades de condicionamiento (MWt, logP y QED vs TPSA, MR y LASA). Se ha utilizado TensorFlow en las simulaciones y RDKit y chemfp como librerías quimioinformáticas. El modelo, en su modo incondicionado, genera colecciones de moléculas con alta diversidad (medida como moléculas únicas, número de clusters y número de frameworks) y relativamente baja novedad (medida como porcentaje de moléculas sin análogos y porcentaje de frameworks nuevos), disminuyendo la diversidad y aumentando la novedad en el condicionado. La corrección aumenta en el modo condicionado, siendo siempre muy alta (> 90%). La diversidad aumenta con el tamaño del conjunto de output, más lentamente en las moléculas condicionadas, y no depende del tamaño del conjunto de entrenamiento. La novedad disminuye significativamente con el tamaño del conjunto de entrenamiento, y aumenta con el del conjunto de output. La diversidad y novedad aumentan con la diversidad intensiva del conjunto de entrenamiento.

Además, se ha modificado el SSVAE para generar con éxito tanto productos

naturales como análogos condicionados, vía condicionamiento multiobjetivo, extendiendo las capacidades originales del SSVAE para diseño molecular.

Abstract (in English, 250 words or less):

The recent SemiSupervised Variational Autoencoder (SSVAE) of Kang & Chao (2018, J. Chem. Inf. Model., Article ASAP DOI: 10.1021/acs.jcim.8b00263) has been analyzed and optimized as Deep Learning inverse QSAR model for conditional molecular design. The aim is to characterize the output of the model (correctness, diversity and novelty, properties distribution) based on different factors: size and diversity of the training set, size of output, type of molecules ("drug-like" vs natural products) and conditioning properties (MWt, logP and QED vs TPSA, MR and LASA). TensorFlow has been used for the simulations and RDKit and chemfp as chemoinformatic libraries. The model, in its unconditioned mode, generates sets of molecules with high diversity (measured as number of unique molecules, number of clusters, and number of frameworks) and relatively low novelty (measured as percentage of molecules with no analogs, and percentage of new frameworks), while in the conditioned mode the diversity decreases and the novelty increases. Correction is slightly higher in the conditioned mode, but always showing very high values (>90%). Diversity increases with the size of the output set (at a lower rate in conditioned mode), and is not dependent on the size of the training set. Novelty decreases with the size of the training set, and increases with that of the output set. Both diversity and novelty increase with the intensive diversity of the training set.

Moreover, the SSVAE has been modified to generate natural products and conditioned analogs (via multiobjective conditioning) thus extending the original molecular design capabilities of the model.

Indice

Lista de figuras

1. Introducción

1.1 Contexto y justificación

Diseño Molecular *in silico*

El diseño molecular de fármacos es un área de intensa investigación, tanto en el ámbito académico como en la industria farmacéutica, dada la necesidad de obtención de nuevos fármacos para patologías no resueltas (e.g. cáncer, ALS, VIH, Alzheimer) o para minimizar efectos adversos o resistencias de los actuales.[1,2] Similares consideraciones se pueden aplicar a otros campos donde el diseño molecular puede ser aplicado, como es la nutrición, la agroquímica, la industria cosmética, la creación de catalizadores y la industria energética (tanto en su vertiente de almacenamiento como de generación de energía). Se estima que el espacio químico de moléculas pequeñas con potencial farmacológico abarca alrededor de 10^{60} moléculas.[3-5] Sin embargo, la capacidad de probar experimentalmente la actividad biológica de moléculas por métodos automatizados (mediante lo que se llama *High-Throughput Screening*) viene a ser del orden de 10^6 moléculas.[6] En este sentido, los métodos *in silico* o de simulación computacional aparecen como una herramienta insustituible, ya que permiten, con un coste muy reducido, el análisis y prueba de enormes cantidades de moléculas mediante cálculos computacionales. De esta forma se puede explorar el espacio químico de una manera rápida y de bajo coste comparativo, al objeto de generar un número reducido de hipótesis o estructuras químicas tentativas que luego se probarán experimentalmente.

Una de las metodologías más utilizadas de diseño molecular *in silico* emplea los denominados modelos de QSAR (de Quantitative Structure-Activity Relationship; para una revisión reciente, ver (7)). Estos modelos se basan en la utilización de diferentes técnicas estadísticas para obtener, a partir de datos experimentales, una relación matemática entre la estructura molecular, codificada mediante una representación de distintos tipos, y la actividad biológica o una propiedad fisicoquímica. Como input el modelo utiliza la estructura molecular codificada, y como output obtenemos la actividad biológica o la propiedad. De esta forma, podemos utilizar el modelo como una plataforma de screening virtual, en la cual hacemos pasar estructuras químicas presentes en una base de datos, y obtenemos un valor de actividad o propiedad para cada una de ellas. Aquellas moléculas que presenten la actividad o propiedad más cercana a la buscada serán las que se seleccionen para ser probadas experimentalmente para su posible confirmación.

Este proceso, por tanto, es muy útil, ya que nos permite extrapolar la información de estructura/actividad obtenida para un conjunto de moléculas con datos experimentales conocidos, a un espacio químico más extenso, especialmente si el modelo se ha derivado con un conjunto diverso de moléculas. Sin embargo, tiene la desventaja de que necesita probar en el modelo una gran cantidad de moléculas virtuales, con el correspondiente coste computacional en algunos casos, ya que a priori no tenemos idea de qué estructuras nos darán el valor de actividad o propiedad deseada.

En este sentido, existe un interés creciente en el desarrollo de metodologías de *QSAR inverso*,[8-10] en las cuales el modelo es capaz de utilizar como input un valor de actividad o propiedad, y de ahí generar automáticamente una o más estructuras que presenten un valor aproximado al deseado. Nosotros estudiaremos en este TFM un modelo de este tipo.

Aparte de la existencia de modelos de QSAR directo o inverso, los modelos de QSAR pueden variar en cuanto a la representación molecular que se utilice como input: descriptores fisicoquímicos, fingerprints, densidades electrónicas, farmacóforos, descriptores topológicos y electrotopológicos, etc (para una introducción a las representaciones moleculares y otros aspectos de la Quimioinformática, ver **Anexo I. Bases de Quimioinformática**). Además, las técnicas estadísticas que se han utilizado para generar modelos de QSAR son también muy variables: regresión lineal, regresión logística, análisis de discriminante, etc., que posteriormente han ido derivando hacia la utilización masiva de técnicas de Machine Learning, dada la naturaleza altamente no lineal de muchas de las relaciones estructura-actividad: random forest, support vector machines, redes neuronales, clasificadores bayesianos, etc.[7] Más recientemente se han empezado a utilizar técnicas de Deep Learning en este área.[11-14] En este trabajo estamos especialmente interesados en estas técnicas, ya que utilizaremos un modelo de Deep Learning de QSAR, por lo que pasaremos a comentar brevemente en qué consiste el Deep Learning.

Deep Learning

El Deep Learning son una serie de métodos de modelado estadístico resultado de la evolución de las redes neuronales, consistentes en la utilización de una gran cantidad de capas de neuronas.[15-16] Se observa que el uso de un gran número de capas resulta en modelos cuyo rendimiento se escala con el tamaño del input, es decir, que a diferencia de otros métodos, cuyo rendimiento se “satura” al aumentar el tamaño de los datos de entrenamiento, el Deep Learning sigue mejorando su rendimiento tanto más cuanto mayor sea el conjunto de entrenamiento.
Además, el uso de muchas capas de neuronas resulta en una capacidad de aprendizaje jerárquico de características (“features”) de los datos, de forma que deja de ser necesaria la derivación de variables de input “específicas de dominio”, es decir, variables reales o categóricas que

correspondan a aproximaciones inteligentes de datos de naturaleza complicada. En este sentido, el Deep Learning ha mostrado su mayor rendimiento en el modelado de datos de tipo no-tabular, como es el caso de datos de audio, imágenes, vídeo, textos, etc.[16] El modelo de Deep Learning es capaz de generar en sus capas iniciales representaciones de los datos con diferentes niveles de características, que luego son utilizadas en capas más profundas del tipo tradicional para realizar el entrenamiento.

La utilización de gran número de capas ha sido posible tras resolver una serie de problemas computacionales y de cálculo numérico, dado el gran número de parámetros a ajustar y la tendencia de los gradientes en el "backpropagation" (el algoritmo de ajuste de pesos de la red) a tomar valores extremos.[16-17] Por un lado, el aumento de la potencia de cálculo durante los últimos años, y más aún la posibilidad de utilización de GPUs, han sido fundamentales. Por otro lado, se han encontrado formas más efectivas de inicialización de los pesos de las redes,[18-19] nuevas funciones de activación (e.g. ReLU,[20] leaky ReLU[21] y ELU[22]), nuevos optimizadores (e.g. stochastic gradient descent, Adam,[23] AdaGrad,[24] RMSProp[25]), nuevas estrategias de entrenamiento (e.g. normalización por batch[26]), y nuevas formas de regularización (e.g. drop-out o la parada de entrenamiento temprana).

Hay diferentes tipos de modelos de Deep Learning, que pueden ser aplicados para diferentes tipos de datos.[15-17] Por ejemplo, las *Redes Neuronales Convolucionales* ("Convolutional Neural Networks", CNN) son principalmente utilizadas para el modelado de imágenes. Las *Redes Neuronales Recurrentes* ("Recurrent Neural Networks", RNN) son apropiadas para el modelado de secuencias, por ejemplo, texto o series temporales. Estas últimas utilizan neuronas especialmente diseñadas para converger rápido y detectar dependencias a largo plazo en los datos: son las neuronas LSTM[27,28] (Long Short-Term Memory) y las neuronas GRU[29] (Gated Recurrent Unit).

También existe una rama del Deep Learning dedicada a lo que se llama *modelos generativos*, es decir, modelos que aprenden de los datos de entrenamiento a generar nuevas versiones de los mismos, a veces de forma guiada. Entre estos, destacan principalmente las *GAN*[30] (Generative Adversarial Networks) y los *VAE*[31] (Variational Autoencoders o Autoencoders Variacionales).

Los modelos generativos de Deep Learning se están empezando a utilizar como forma de hacer QSAR inverso, que hemos visto más arriba es de especial interés para el diseño molecular *in silico* (ver revisión en (32)). En la mayoría de los casos, el modelo incluye una RNN para modelar secuencias de caracteres de tipo SMILES,[33] que codifican la estructura del grafo molecular (ver **Anexo I**. **Bases de Quimioinformática** para una explicación de los códigos SMILES). Por ejemplo, se han desarrollado modelos de RNN, en los cuales se condiciona el diseño molecular mediante re-entrenamiento del modelo

con moléculas apropiadas,[34] o bien que mediante “aprendizaje por refuerzo” (Reinforcement Learning, o RL) para condicionar la generación de moléculas.[35,36] Otros modelos utilizan VAEs y luego optimización Bayesiana para condicionar la generación de moléculas.[37-39] Hay también modelos que utilizan GANs con RL,[40,41] o híbridos GAN-VAE (Adversarial Autoencoders,[42,43] AAE). Finalmente, hay trabajos recientes que modelan la molécula como grafo directamente y no como cadena de caracteres, dentro de esquemas de VAEs.[44-47]

Muy recientemente se ha publicado un nuevo modelo de QSAR inverso (Kang y Cho, 2018)[48] que utiliza VAEs semisupervisados (Semi-Supervised Variational AutoEncoders,[49] SSVAE) para generar moléculas condicionadas. Entre las ventajas de esta aproximación está su relativa simplicidad matemática y programática, así como su relativa rapidez de ejecución en comparación con los modelos anteriores. Además, los autores proporcionan el código fuente. En su artículo, Kang y Cho presentan el modelo, lo entrenan con un conjunto al azar de 300000 moléculas “drug-like“ (ver **Anexo I** para definición de moléculas “drug-like”) sacadas al azar de la base de datos ZINC,[50,51] y comprueban que son capaces de guiar la generación de moléculas a valores predeterminados de peso molecular, logP (logaritmo del coeficiente de partición octanol/agua, una medida de la lipofilicidad de la molécula) de Wildman y Crippen,[53] y el QED (Quantitative Estimation of Drug-likeness,[16] una medida de la “druggability” de la molécula).

Sin embargo, los autores no hacen una investigación sistemática de las moléculas que su modelo genera, en cuanto al espacio químico cubierto y su relación con el espacio químico del conjunto de entrenamiento. Cabe preguntarse por la diversidad química del output y su novedad respecto del conjunto de entrenamiento, y así como por posibles diferencias entre las distribuciones de propiedades del input y el output. ¿Estamos generando pequeñas variaciones del espacio químico inicial, o el modelo es capaz de generar estructuras completamente nuevas? ¿El modelo muestrea un amplio espacio químico, o se restringe a unos pocos quimiotipos? ¿Cómo se restringe la diversidad y novedad al condicionar la generación de moléculas? Parece interesante también estudiar la dependencia del output del modelo con el conjunto de moléculas de entrenamiento: ¿cómo es esto afectado con el tamaño y la diversidad del conjunto de entrenamiento?

Por otra parte, Kang y Cho[48] utilizaron moléculas “drug-like”. ¿Es posible adaptar el modelo para generar otros tipos de moléculas, como son los productos naturales? Y también podemos preguntarnos sobre el comportamiento del modelo al condicionar la generación por otras propiedades no utilizadas por los autores. ¿Es posible modificar el modelo para generar moléculas condicionadas a dos o más propiedades? Esto sería especialmente útil, ya que el proceso de optimización de moléculas para generar candidatos a fármacos es altamente multidimensional. En particular, ¿podemos generar análogos a un compuesto con propiedades modificadas?

El presente TFM intentará responder a toda esta gran cantidad de preguntas, mediante el análisis y optimización del modelo SSVAE aplicado a diseño molecular condicionado (QSAR inverso, ver abajo la sección **1.2. Objetivos**). Al objeto de proporcionar el suficiente contexto, el modelo SSVAE se describirá para finalizar esta sección.

Autoencoders variacionales semisupervisados (SSVAE) para diseño molecular condicionado. El modelo de Kang y Cho

Un VAE[31] es un modelo probabilístico que combina un codificador (*encoder*, una red neuronal profunda en nuestro caso) para transformar un input **x** en una media **μ** y desviación estándar **σ**, que sirven para muestrear al azar un punto en una distribución normal multidimensional con esos parámetros, que es decodificado mediante un decodificador (*decoder,* otra red neuronal profunda), que genera un output **x**'. El conjunto es entrenado mediante inferencia variacional para minimizar la distancia entre los datos input y output, y simultáneamente forzar a los datos codificados a seguir una distribución estándar multivariada (el espacio latente **z**, minimizado la distancia de Kullback-Leibler entre la distribución empírica de datos codificados y una distribución estándar multivariada). De esta forma se mapea el espacio de los datos en el espacio latente de forma compacta y ordenada.

Es fácil ver cómo puede utilizarse un VAE como modelo generativo: simplemente se muestrean puntos en el espacio latente, y se decodifican dando outputs **x'** en el espacio de los datos. Ahora bien, estamos interesados en generar de forma *condicionada* estos outputs, es decir, haciendo que presenten valores cercanos a un valor determinado de una variable **y** (que puede ser multidimensional, es decir, contener varias propiedades de condicionamiento). Esto se consigue añadiendo un predictor (de nuevo, una red neuronal profunda) que se entrena junto con el resto del modelo. El resultado es el SSVAE.[49] El adjetivo de semisupervisado hace referencia a que los datos utilizados están en parte "etiquetados" (conocemos los valores de **y** para ellos) y en parte no.

En este caso, la función de pérdida minimizada contiene dos términos: uno para datos etiquetados, L(**x**,**y**), correspondiendo al límite inferior variacional de logp(**x**, **y**), siendo p(**x**, **y**) la probabilidad de una instancia etiquetada, y otro para los datos sin etiquetar, U(**x**), que corresponden al límite inferior variacional de logp(**x**), siendo p(**x**) la probabilidad de una instancia sin etiquetar. Un parámetro β controla la proporción con que cada término se incluye en la función de pérdida total. De esta forma se optimiza el modelo para predecir **y** a partir de **x** correctamente, y se fuerza a **z** a seguir una distribución estándar multivariante, permitiendo obtener una distribución posterior $p_\theta(\mathbf{x}|\mathbf{y},\mathbf{z})$ a partir de la cual se generarán los outputs condicionados.

Para trabajar con moléculas, Kang y Cho[48] utilizaron tres RNN para modelar cadenas SMILES. Estos a su vez se codificaron con el método *one-hot*, el cual consiste en crear un diccionario de posibles caracteres, y codificar cada carácter del SMILES como un vector de 0s con un 1 en la posición correspondiente al carácter. Para tratar SMILES de distintas longitudes se añadieron 0s al final de los códigos (*zero-padding*) hasta llegar a una longitud máxima mayor que el más largo de los SMILES. El resultado para un SMILES será una matriz dispersa de 0s y 1s, de dimensión n*m, donde n es la longitud del diccionario y m la longitud de los vectores codificadores de caracteres tras el *zero-padding*, y eso será in input **x** para la red. Para las RNN utilizaron a su vez neuronas GRU, bidireccionales para el predictor y para el codificador, y unidireccionales para el decodificador. El esquema del modelo es el siguiente:

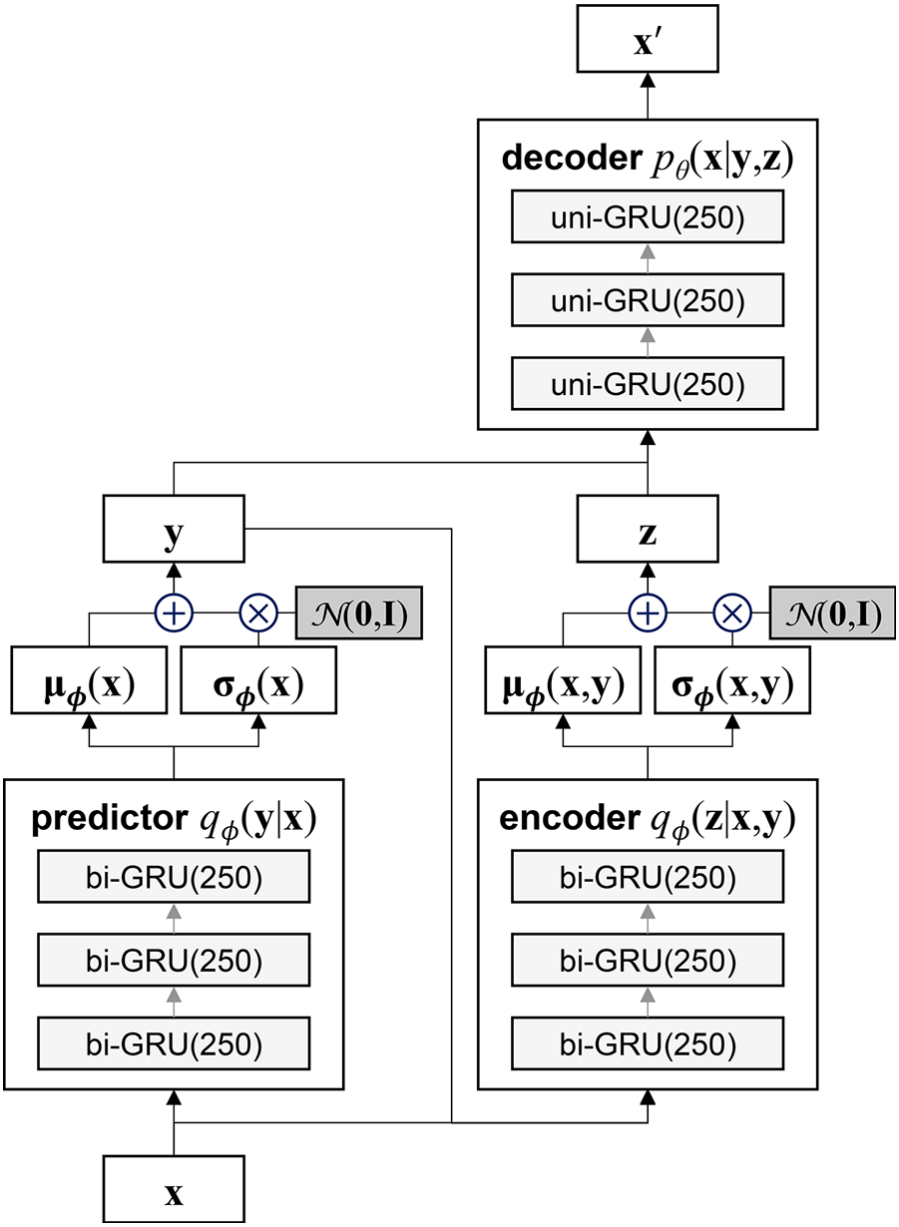


Figura 1. Modelo SSVAE utilizado por Kang y Cho. Tanto el predictor como el codificador generan para cada instancia **x** una media y una desviación estándar, a

partir de las cuales se genera **y** y **z**, que se utilizan como input del decodificador. Reproducido de (48).

Para generar una molécula a partir de **y** y **z** utilizaron *beam search* basándose en la distribución posterior condicionada $p_\theta(\mathbf{x}|\mathbf{y},\mathbf{z})$.

El programa para implementar el modelo fue escrito en Python utilizando la librería TensorFlow.[55] Para el entrenamiento utilizaron 310000 moléculas "drug-like" de ZINC, que fueron divididas en 300000 moléculas para entrenamiento *per se* y 10000 moléculas para validación. Utilizaron el siguiente diccionario de 35 caracteres de código SMILES para realizar la codificación *one-hot*:

{1, 2, 3, 4, 5, 6, 7, 8, 9, + ,=,#,(,), [,], H, B, C, N, O, F, Si, P, S, Cl, Br, Sn, I, c, n, o, p, s}

La dimensión de **z** fue establecida en 100.

Pasamos a enumerar los objetivos del TFM.

1.2 Objetivos

Los objetivos del TFM serán cinco:

- Análisis de la corrección, novedad, diversidad y distribución de propiedades del output del SSVAE entrenado con el conjunto de entrenamiento original, tanto incondicional como condicional.
- Análisis de la corrección, novedad, diversidad y distribución de propiedades del output, incondicional y condicional, en función de:
    - Tamaño del conjunto de entrenamiento
    - Diversidad del conjunto de entrenamiento
- Adaptación del SSVAE para generar productos naturales; análisis de la corrección, novedad, diversidad y distribución de propiedades con el conjunto de productos naturales de ZINC como conjunto de entrenamiento.
- Prueba del SSVAE con propiedades alternativas: TPSA[56] (Total Polar Surface Area), refractividad molar (MR) y Labute ASA[57] (LASA); análisis de corrección, novedad, diversidad y distribución de propiedades.
- Adaptación del SSVAE para condicionar la generación de moléculas con dos o más propiedades, en particular para generar análogos con propiedades modificadas.

1.3 Materiales and Métodos

Ambiente de simulación: hardware y software

Todas las simulaciones y análisis fueron realizados en dos workstations de Linux, una con 32 Gb de RAM y Ubuntu Xenial como sistema operativo, y otra con 64 Gb de RAM y Debian Jessie como sistema

operativo. En la primera había hasta 40 hilos de cálculo en paralelo disponibles, y en la segunda hasta 32 hilos.

Todas las simulaciones utilizaron Python 3.5 con la librería TensorFlow. Se creó un ambiente de Anaconda (tf35) en el cual se instaló TensorFlow, NumPy, y scikit-learn.

Todos los análisis utilizaron Python 2.7 (debido a incompatibilidades de la librería chemfp[58]). Se creó un ambiente de Anaconda para análisis quimioinformáticos (cix) en el cual se instaló RDKit,[59] jupyter, NumPy, Pandas, MatplotLib, y chemfp. Los cálculos quimioinformáticos utilizaron las toolkits RDKit y chemfp.

Es de destacar que, a pesar de estar utilizando workstations con procesadores Xeon con extensiva capacidad de paralelización, junto con una gran cantidad de RAM, la ejecución del SSVAE es bastante costosa computacionalmente y por tanto lenta de obtener resultados. Por ejemplo, la simulación en las condiciones de Kang y Cho[48] (300K moléculas de input, 5K + 5K moléculas de output) puede tardar en estas máquinas del orden de 1-2 días. Se hicieron algunas pruebas en ordenadores con Windows y utilizando CPUs los cálculos se volvían interminables. Únicamente fue relativamente exitosa una prueba realizada utilizando TensorFlow para GPUs en un laptop con 64 Gb de RAM, y aun así el cálculo tardó 2-3 días. Por tanto, se aconseja ejecutar estos cálculos en workstations altamente paralelizables y con gran cantidad de RAM, y si es posible, utilizando GPUs.

Repositiorio Git

Al objeto de mantener un correcto control de versiones, y permitir la reproducción pública de las simulaciones y análisis se creó un repositorio público en GitHub con copias locales sincronizadas en ambas workstations. La dirección del mismo es:

https://github.com/gcolmenarejo/cmd

Este repositorio contiene el código del repositorio público original de Kang y Cho,[48] más todas las modificaciones de código, resultados de simulaciones y notebooks de análisis de este TFM. Se organiza en forma de directorios exp0, exp1, exp2, etc. Cada uno de ellos contiene un experimento o área de trabajo, con uno o más ficheros de tipo **`myrunxx.py`** que sirven para realizar la simulación, y uno o más notebooks de Python de tipo **`ExpxxAnalysis.ipynb`** donde se realiza el análisis correspondiente. En el directorio raíz está el fichero **`run.py`**, correspondiente al código de Python de control de simulación originario, junto con un módulo **`SSVAE.py`** que contiene el código del SSVAE y un módulo **`preprocessing.py`** que realiza la codificación *one-hot* de los SMILES y otras funciones iniciales de procesado.

Los diferentes directorios de experimentos creados son:

exp0: simulación original de Kang y Cho,[48] junto con análisis de output: corrección, novedad, diversidad, distribución de propiedades, tanto incondicional como condicionalmente.

exp1: simulación similar a la anterior, pero iterando sobre conjuntos de entrenamiento cada vez mayores, parte o la totalidad del conjunto de entrenamiento original. También se probó a generar tamaños crecientes de conjuntos de output: 5000, 10000 y 20000 moléculas.

exp3: simulación de generación de productos naturales, con análisis de corrección, novedad, diversidad, y distribución de propiedades, tanto incondicional como condicionalmente. Se itera sobre conjuntos de entrenamiento cada vez mayores, parte o totalidad del conjunto de productos naturales de ZINC,[50,51] previamente procesado para poder ser utilizado con el SSVAE. Contiene un notebook adicional (**`generateNP.ipynb`**) para generar el conjunto de productos naturales de input, así como un módulo **`SSVAE.py`** modificado para poder tratar productos naturales con el SSVAE.

exp4: simulaciones de generación de moléculas a partir de conjuntos de entrenamiento de moléculas de máxima diversidad (ortogonales) y mínima diversidad (clusterizadas). Se iteró sobre conjuntos crecientes de moléculas, y se analizó comparativamente la corrección, novedad, diversidad y distribución de propiedades del output, tanto incondicional como condicionalmente. Para crear los conjuntos ortogonal y clusterizado de input, se creó un notebook de Jupyter (**`Exp4DivSample.ipynb`**) que utilizó como fuente de moléculas las moléculas “lead-like” o “drug-like” de ZINC.[50,51]

exp5: simulación de generación de moléculas condicionadas a valores específicos de TPSA, refractividad molar (MR) y LASA. Utiliza el módulo **`mySSVAE.py`** (en el directorio raíz) que contiene el SSVAE modificado para optimización multiobjetivo.

exp6: simulación de generación de análogos condicionados a un valor específico de logP. Utiliza el módulo **`mySSVAE.py`** (en el directorio raíz) que contiene el SSVAE modificado para optimización multiobjetivo.

Para los análisis estadísticos se utilizó R.

Módulo myfuncs.py

Se creó un módulo de Python (**`myfuncs.py`**) con múltiples funciones de análisis, clusterización, visualización de moléculas, gráficas, muestreo químico, etc. Dicho módulo se encuentra reproducido en el **Anexo II**. Las funciones en este módulo se utilizaron en los notebooks de análisis.

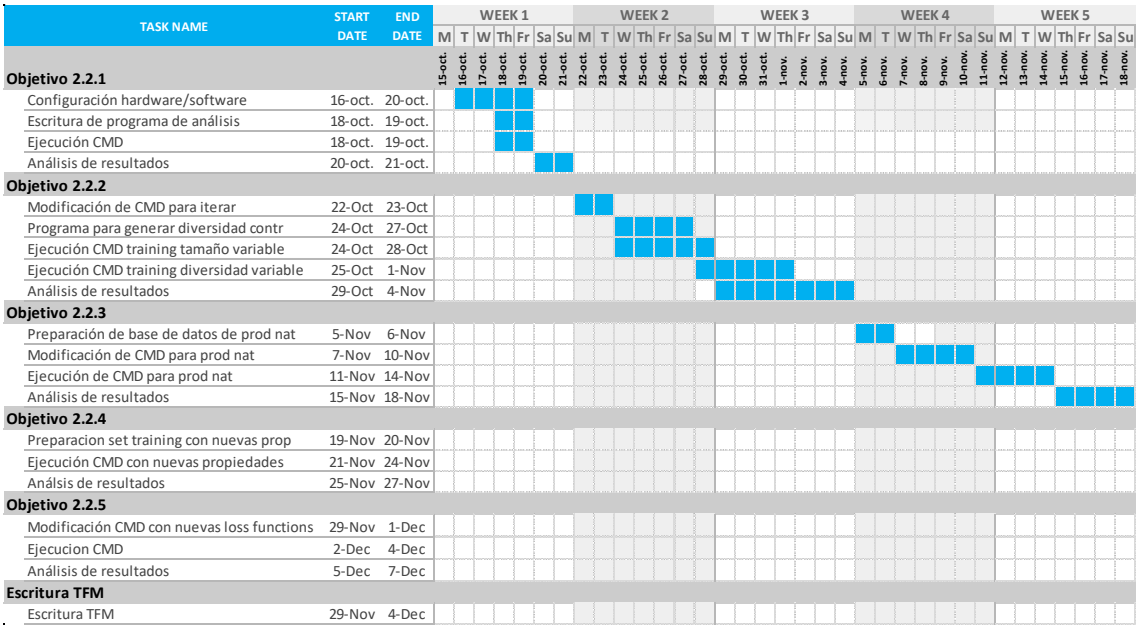
Notebooks de Jupyter

Para cada análisis se creó un notebook de Jupyter, que incluye cálculos y gráficas y están ampliamente comentados. También se crearon notebooks de Jupyter para generar diferentes conjuntos de moléculas de entrenamiento. Estos notebooks están recogidos en el repositorio GitHub arriba mencionado.

Las gráficas se realizaron con Matplotlib de Python o Spotfire.[60]

1.4 Planificación del trabajo

La planificación del trabajo se hizo de acuerdo a un Gantt chart generado al inicio del proyecto. Se muestra a continuación:



TASK NAME	START DATE	END DATE
Objetivo 2.2.1		
Configuración hardware/software	16-oct.	20-oct.
Escritura de programa de análisis	18-oct.	19-oct.
Ejecución CMD	18-oct.	19-oct.
Análisis de resultados	20-oct.	21-oct.
Objetivo 2.2.2		
Modificación de CMD para iterar	22-Oct	23-Oct
Programa para generar diversidad contr	24-Oct	27-Oct
Ejecución CMD training tamaño variable	24-Oct	28-Oct
Ejecución CMD training diversidad variable	25-Oct	1-Nov
Análisis de resultados	29-Oct	4-Nov
Objetivo 2.2.3		
Preparación de base de datos de prod nat	5-Nov	6-Nov
Modificación de CMD para prod nat	7-Nov	10-Nov
Ejecución de CMD para prod nat	11-Nov	14-Nov
Análisis de resultados	15-Nov	18-Nov
Objetivo 2.2.4		
Preparacion set training con nuevas prop	19-Nov	20-Nov
Ejecución CMD con nuevas propiedades	21-Nov	24-Nov
Análsis de resultados	25-Nov	27-Nov
Objetivo 2.2.5		
Modificación CMD con nuevas loss functions	29-Nov	1-Dec
Ejecucion CMD	2-Dec	4-Dec
Análisis de resultados	5-Dec	7-Dec
Escritura TFM		
Escritura TFM	29-Nov	4-Dec

Figura 2. Gantt chart del proyecto de TFM (semanas 1-5).

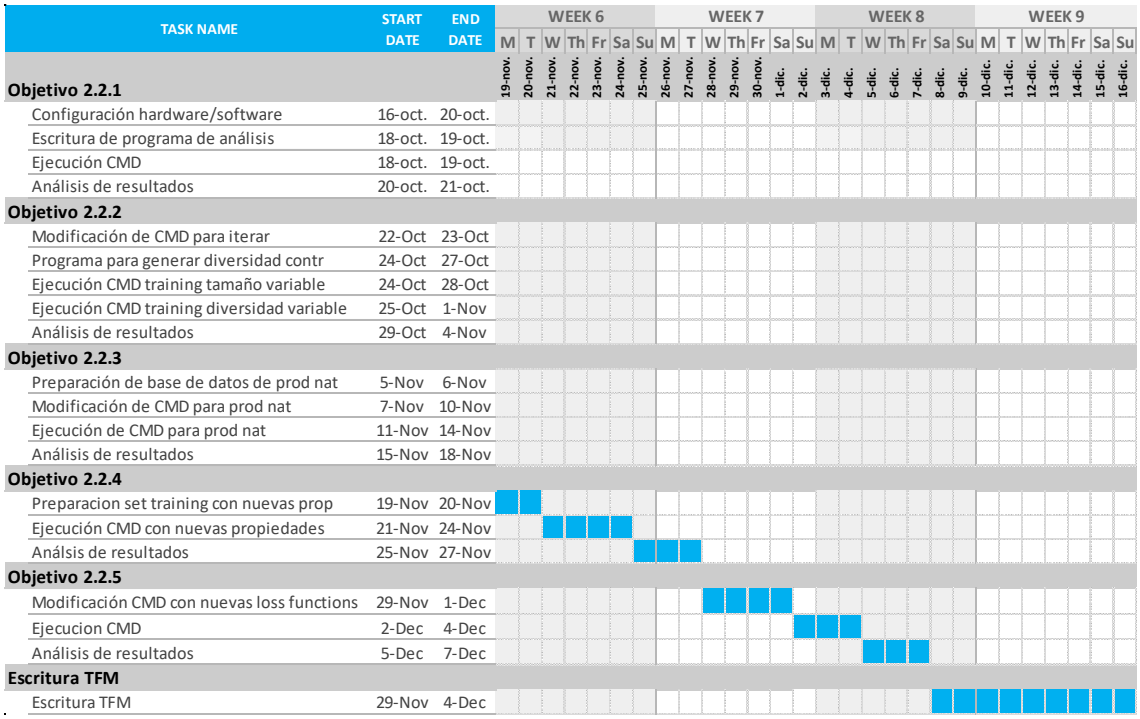


Figura 3. Gantt chart del proyecto de TFM (semanas 6-9).

1.5 Breve descripción del producto

En este TFM se ha analizado sistemáticamente el modelo SSVAE de Kang y Cho[48] de diseño molecular condicionado. Diferentes experimentos se han realizado al objeto de **estudiar la dependencia del output con el input, tanto en su extensión, diversidad y tipo de moléculas, así como propiedades condicionantes**. Además, se han realizado modificaciones en el SSVAE de forma que se ha conseguido:

- **generar moléculas de tipo producto natural**.
- **generar moléculas condicionadas a más de una propiedad**
- **generar análogos condicionados a una o más propiedades.**

1.6 Breve descripción de la memoria

La memoria a continuación se organiza en forma de una sección de **Resultados y Discusión**, en la cual se realizará una descripción de todos los experimentos y análisis realizados, los resultados obtenidos y se discutirán en referencia a la bibliografía en el área. Seguirá una sección de **Conclusiones** que resumirán los principales hallazgos de este trabajo. Finalmente, habrá una par de **Anexos**, uno con una introducción resumida a la Quimioinformática (**Bases de Quimioinformática**), y otro con material suplementario con el código del módulo **myfuncs.py** de Python generado para facilitar los análisis.

2. Resultados y Discusión

2.1. Análisis del output del SSVAE con el entrenamiento original

En su artículo,[48] Kang y Cho describen el entrenamiento y ejecución del SSVAE con un conjunto de 310000 moléculas, concretamente 300000 para el entrenamiento de la red *per se* y 10000 para su validación. Estas 310000 moléculas son un conjunto al azar tomado de las moléculas "drug-like" de ZINC.[50,51] Tras el entrenamiento, generaron 5000 moléculas incondicionadas y otras 5000 moléculas condicionadas a tener un peso molecular de 250 Da.

La presente sección describe la reproducción de la simulación de Kang y Cho, y se analiza el output de dicha simulación. En particular, se estudia la *corrección*, *diversidad* y *novedad* de las moléculas resultantes, así como su *distribución de propiedades* en comparación con el conjunto de entrenamiento.

Para determinar la corrección de las moléculas, se creó una función (**`corrsmis`**, dentro del módulo **`myfuncs.py`**) que toma una lista de SMILES y separa aquéllos que son sintácticamente correctos (a partir de ellos se puede reconstruir el grafo molecular) de aquéllos que no lo son, siendo retornados en sendas listas, junto con el número total de SMILES, y el número de los correctos. Para ello utiliza la función **`MolFromSmiles`** de RDKit. Esta función se aplicó a cada conjunto de SMILES output analizados.

En cuanto a la diversidad, se calcularon tres parámetros: el número de estructuras únicas generadas (no es raro que el SSVAE genere moléculas repetidas), el número de clusters presentes en las moléculas resultantes, y el número de frameworks de Murcko[61] únicos (ver **Anexo I. Bases de Quimioinformática** para una descripción de los mismos). Para las dos últimas se creó una función (**`divan`**, dentro del módulo myfuncs.py) que toma un dataframe de Pandas `smidf` con una columna de SMILES y otra de identificador, y lo clusteriza (llamando a la función **`clusmidf`**, dentro de **`myfuncs.py`**) mediante el algoritmo de Taylor-Butina[62,63] utilizando un umbral de similitud de 0.8, que habitualmente resulta en clusters con moléculas intuitivamente similares. El clustering se realiza mediante la toolkit quimioinformática chemfp, que es capaz de generar la lista de vecinos a muy alta velocidad.

La función **`divan`** también genera la lista de frameworks de Murcko únicos utilizando la función **`GetScaffoldForMol`** de RDKit.

En cuanto a la novedad química, se creó una función (**`novan`**, dentro del módulo **`myfuncs.py`**) que por una parte calcula, utilizando chemfp, el número de moléculas sin similitudes iguales o superiores a 0.7 en el conjunto de entrenamiento (moléculas con similitud por debajo de 0.7

suelen aparecer intuitivamente como de diferentes quimiotipos). Esto es una medida del número de moléculas sin análogos en el conjunto de entrenamiento y que por tanto corresponden a quimiotipos nuevos. También calcula el número de frameworks de Murcko no presentes en el conjunto de entrenamiento, una medida alternativa de la novedad química, en este caso enfocándose en la estructura del núcleo de la molécula en vez de la molécula entera (moléculas nuevas con el mismo framework pero distintas cadenas laterales no será contadas como novedad).

Para la distribución de propiedades, siguiendo a Kang y Cho se entrenó a la red utilizando peso molecular, logP y QED como propiedades adicionales. Para nuestro análisis de distribución de propiedades nos fijamos sólo en el peso molecular y el logP, dada la alta dependencia del QED en estas dos propiedades. A continuación, describiremos en tres subsecciones el conjunto de entrenamiento, el output incondicionado y el output condicionado.

Conjunto de entrenamiento

Si nos fijamos en el conjunto de entrenamiento, vemos que contiene un total de 194527 clusters (según el algoritmo de Taylor-Butina con umbral de 0.8), correspondiendo a un promedio de 1.54 moléculas por cluster, y un total de 161518 frameworks de Murcko (promedio de 1.86 moléculas por framework). La distribución de peso molecular vs logP tiene el siguiente aspecto (**Fig. 4**):

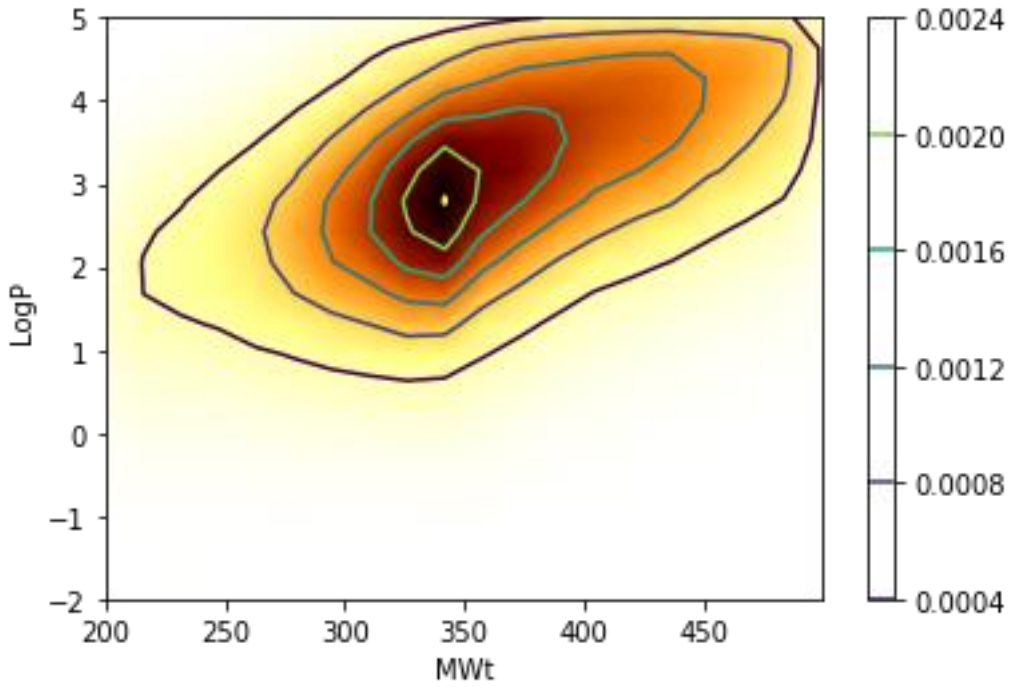


Figura 4. Distribución de peso molecular (MWt en inglés) vs logP del conjunto de entrenamiento de Kang y Cho (300000 moléculas). A la derecha, escala de contornos de igual densidad.

Vemos que está limitada por arriba a un peso molecular de 500 Da y un logP de 5, correspondiendo por tanto a moléculas “drug-like” como se esperaba. Se observa cierta correlación entre ambas propiedades, que puede cuantificarse en un coeficiente de Pearson de 0.43. El máximo de la distribución se encuentra en un peso molecular de aproximadamente 350 Da y en un logP de aproximadamente 3, y las mayores densidades se dan entre valores de 325-400 Da y 2-4 de logP, habituales en moléculas “drug-like”. En cuanto a la distribución de clusters, vemos una dependencia de tipo ley de potencia, abarcando distintos órdenes de magnitud los tamaños de los clusters (hay clusters de hasta casi 900 moléculas, y luego tamaños decrecientes hasta una larga cola de singletons), con una dependencia lineal en las representaciones doble-logarítmicas:

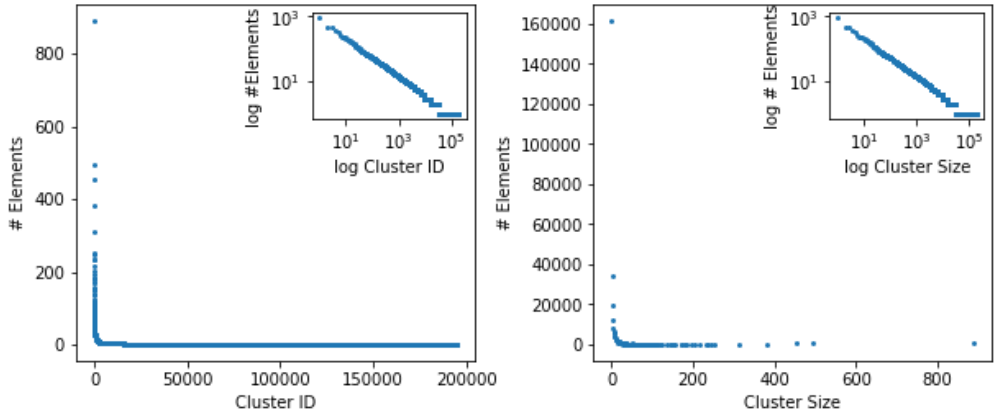


Figura 5. Distribución clusters y de tamaños de clusters de conjunto de entrenamiento de Kang y Cho (300000 moléculas). En la gráfica de la izquierda se representa el número de elementos de cada cluster (en escala doble-logarítmica en el inset). En la gráfica de la derecha se representa el número de elementos para cada tamaño de cluster (en escala doble-logarítmica en el inset), es decir, cada punto representa la agregación de todos los clusters del mismo tamaño, resultando en menos puntos; el punto más a la izquierda son los singletons, que corresponden a la cola derecha de la gráfica de la izquierda.

Este comportamiento es típico para distribuciones heterogéneas y grandes de compuestos.[64]

Generación incondicionada de moléculas

Tras ejecutar el SSVAE en con los parámetros de Kang y Cho, se obtuvieron los siguientes resultados para la generación incodicionada de moléculas (**Tabla 1**):

# mol out	%corr	# mol unicas	# clusters	# frames	% nuevas estr	% nuevos frames
5000	94.68	4218	3352	2973	23.14	49.31

Tabla 1. Resultados de corrección, diversidad y novedad de la generación incondicionada tras entrenamiento en las condiciones de Kang y Cho. # mol out **=** número de moléculas del output; % corr = porcentaje de moléculas sintácticamente

correctas; # clusters = número de clusters; # frames = número de frameworks de Murcko únicos; % nuevas estr = porcentaje de estructuras nuevas (sin análogos en el conjunto de entrenamiento; % nuevos frames = porcentaje de frameworks nuevos (no presentes en el conjunto de entrenamiento).

Vemos en primer lugar que el porcentaje de moléculas generadas sintácticamente correctas es del 94.7%, lo cual es remarcable en su alto valor dada la complejidad que a veces puede alcanzar la codificación de una cadena SMILES, con interacciones entre posiciones muy alejadas en la cadena. Parece que la red ha aprendido muy bien a generar estructuras químicas realistas. También vemos que ha habido algunas repeticiones de moléculas, puesto que el número de moléculas únicas (4218) está ligeramente por debajo del de correctas (4735, lo que corresponde a aproximadamente un 11% de repeticiones).

En cuanto a la diversidad química, la simulación incondicionada resultó en una colección bastante diversa, pues hay 3352 clusters (un promedio de 1.26 moléculas por cluster) y 2973 diferentes frameworks de Murcko (1.42 moléculas por framework en promedio). La distribución de clusters resultantes es de nuevo de tipo ley de potencia (**Fig. 6**):

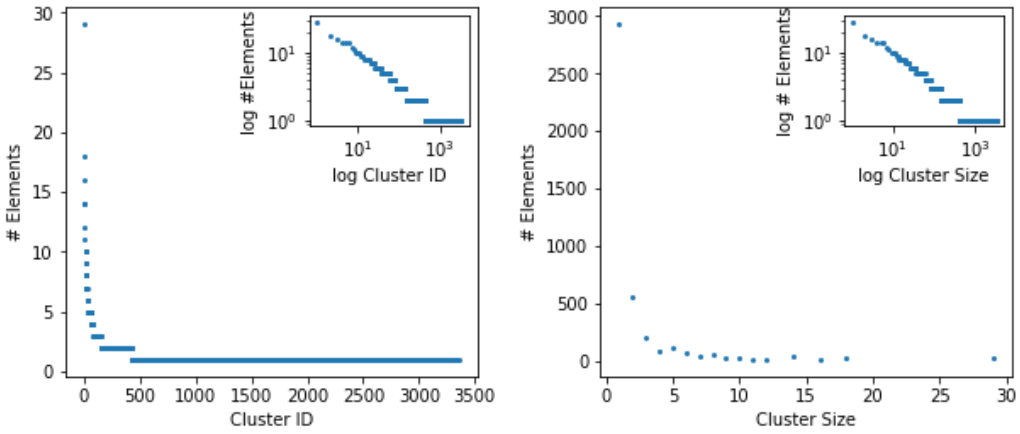


Figura 6. Distribución clusters y de tamaños de clusters de conjunto de output incondicionado tras entrenamiento con condiciones de Kang y Cho. Gráficas como en la Fig. 5.

En cuanto a la novedad química, vemos que hay un porcentaje de sólo el 23.14% de moléculas que podamos categorizar como no análogos del conjunto de entrenamiento, es decir, quimiotipos nuevos. Si nos ceñimos a los frameworks, el porcentaje es mayor, del 49.31%, aunque de nuevo es relativamente bajo. Parece que el SSVAE aprende no sólo a generar moléculas correctamente, sino también a generar moléculas según los quimiotipos presentes en el conjunto de entrenamiento en bastante medida. Esto no debería sorprendernos, pues los modelos generativos de tipo VAE en su versión incondicionada lo que hacen es muestrear puntos del espacio latente y decodificarlos, por lo que los datos que se generan de output tienen en principio bastante similitud con los originales, si bien es interesante el observar que también hay una generación no desdeñable de quimiotipos completamente nuevos.

La distribución de peso molecular vs logP de las moléculas generadas incondicionalmente es la siguiente (**Fig. 7**):

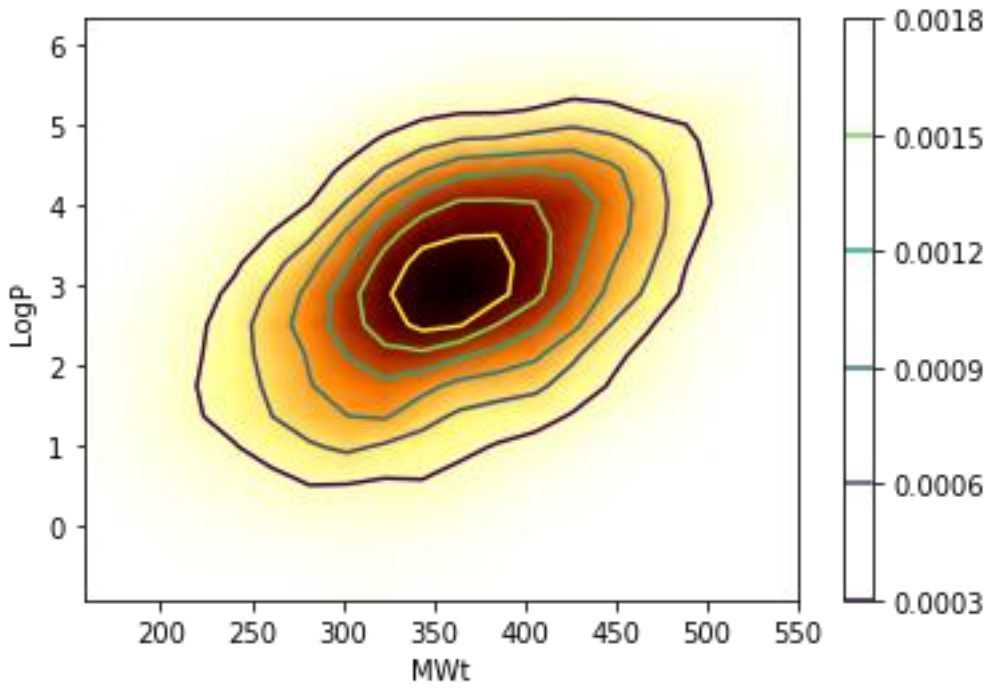


Figura 7. Distribución de peso molecular vs logP del output incondicionado tras entrenamiento con condiciones de Kang y Cho.

De nuevo vemos una distribución centrada en el peso molecular de 350 Da y el logP de alrededor de 3 (coeficiente de correlación de Pearson de 0.42), similar al conjunto de entrenamiento.

Generación condicionada de moléculas

Por otra parte, la generación condicionada a peso molecular alrededor de 250 Da nos dio los siguientes resultados (**Tabla 2**):

# mol out	%corr	# mol unicas	# clusters	# frames	% nuevas estr	% nuevos frames
5000	96.4	2590	2277	1525	38.26	43.21

Tabla 2. Resultados de corrección, diversidad y novedad de la generación condicionada tras entrenamiento en las condiciones de Kang y Cho

En este caso vemos que la corrección de moléculas no parece haber sido afectada y sigue estando en un valor muy alto. Por otra parte, el número de moléculas únicas ha disminuido considerablemente, resultando en un 46.3% de repetición en las moléculas correctas. El número de clusters ha disminuido en un 32%, y el de frameworks en un 49%. La distribución de clusters muestra la ley de potencias típica, aunque con clusters más pequeños (**Fig. 8**):

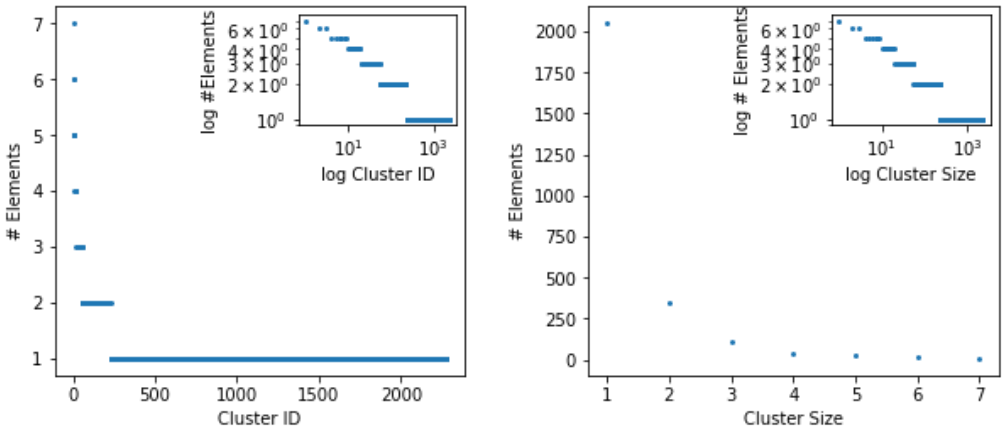


Figura 8. Distribución clusters y de tamaños de clusters de conjunto de output condicionado tras entrenamiento con condiciones de Kang y Cho. Gráficas como en la Fig. 5.

Esta observación de clara disminución de la diversidad química del output puede explicarse en base a que estamos forzando al modelo a generar moléculas con unas propiedades restringidas, en particular, con un peso molecular de 250 Da, por lo que estamos muestreando una región restringida del espacio latente. Si observamos la **Fig. 4** vemos que el porcentaje de moléculas del conjunto de entrenamiento original con esa característica es muy bajo, por lo que el modelo tiene poca información sobre este tipo de moléculas. Por consiguiente, resulta lógico que aumente el número de moléculas repetidas y que la diversidad de las moléculas resultantes de la generación condicionada sea menor.

En cuanto a la novedad química, se observa un aumento claro de la misma al analizar el porcentaje de nuevas estructuras (38.26% vs 23.14% en el entrenamiento incondicionado), y un número ligeramente inferior, no sabemos si significativo, al analizar el porcentaje de frameworks nuevos (43.21% vs 49.31%). La observación del aumento de estructuras no análogas puede racionalizarse de nuevo si consideramos que estamos condicionando en una región del espacio latente con pocos representantes en el conjunto de entrenamiento. Por ello, estamos forzando al modelo a generar moléculas poco representadas en aquél en su globalidad, y por tanto aumentando la probabilidad de generar quimiotipos nuevos, aunque sea a costa de una disminución de la diversidad del output y un aumento de las repeticiones.

Finalmente mostramos la distribución de peso molecular vs logP del conjunto de output condicionado (**Fig. 9**):

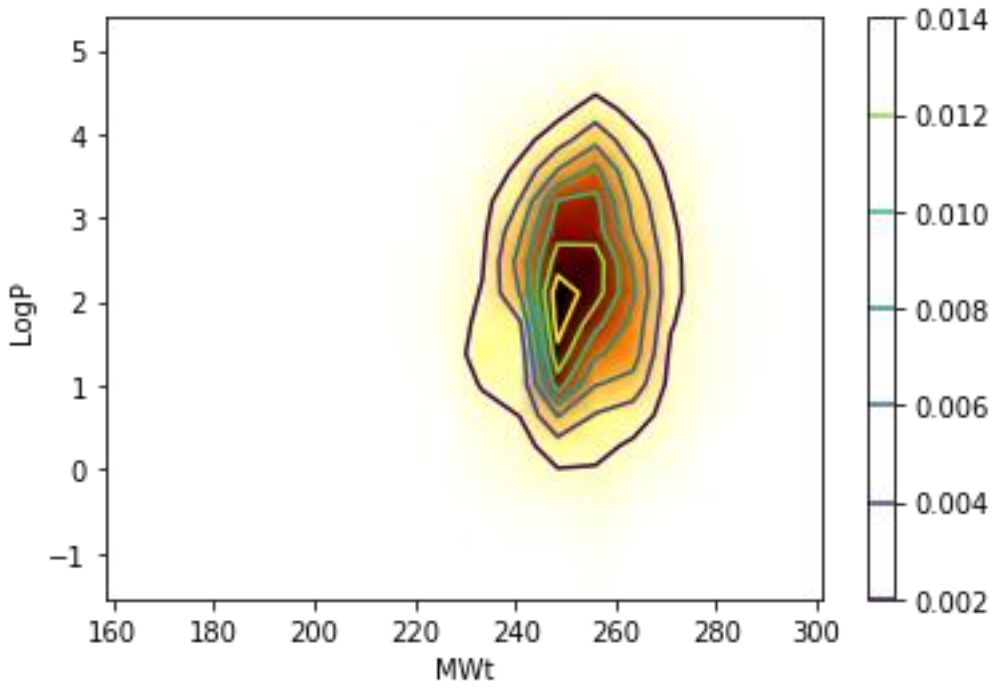


Figura 9. Distribución de peso molecular vs logP del output condicionado tras entrenamiento con condiciones de Kang y Cho. Obsérvese la disminución de variabilidad del peso molecular en comparación con el logP.

Observamos que la generación condicionada ha sido un éxito, puesto que la distribución marginal de peso molecular está centrada muy estrechamente en 250 Da. Es interesante observar que en este caso la distribución marginal de logP no se centra en 3, como en el conjunto de entrenamiento, sino en aproximadamente 2, y es más ancha. Esto puede explicarse si consideramos la **Fig. 4**, en la que podemos ver que en el conjunto de entrenamiento la mayoría de las moléculas con peso molecular alrededor de 250 Da tienen un logP centrado en aproximadamente 2. El modelo, entonces, al muestrear el espacio latente condicionado a peso molecular de 250 Da estará generando puntos que al ser decodificados resultarán en moléculas con tendencia a tener logP de 2.

Concluimos que el SSVAE genera moléculas con una alta diversidad química y cierta novedad. En el caso incondicionado, esta novedad es menor, pero la diversidad es mayor. Lo contrario ocurre en el caso condicionado, con mayor novedad pero menor diversidad. En cualquier caso, es capaz de generar nuevo espacio químico, y lo que es más importante, condicionado a unos valores de propiedad, por lo que parece tener capacidad de extrapolación para generar nuevas moléculas con un valor determinado de una propiedad, actuando como modelo efectivo de QSAR inverso.

2.2. Análisis del output del SSVAE en función del tamaño del conjunto de entrenamiento y del tamaño del output

A continuación, nos preguntamos si el output del SSVAE, en cuanto a corrección, diversidad y novedad podía depender el tamaño del conjunto de entrenamiento. Cabría esperar que al aumentar el tamaño de dicho conjunto la red tuviera más diversidad química para aprender y por tanto generara un output más correcto, diverso y nuevo. Por ello, realizamos simulaciones del SSVAE con subconjuntos crecientes del conjunto de entrenamiento de Kang y Cho, concretamente de 50K, 100K, 150K, 200K, 250K y 300K moléculas (esta última es la simulación de la sección anterior). Asimismo, probamos a generar, para cada una de ellas, conjuntos de moléculas incondicionadas y condicionadas de tamaño creciente, en concreto de 5000, 10000 y 20000 moléculas, al objeto de ver si la diversidad y novedad de los outputs crecían linealmente con el tamaño del output, o por el contrario se producían fenómenos de "saturación", etc.

Corrección

En primer lugar, analizamos la dependencia de la corrección de moléculas con el tamaño del conjunto de entrenamiento y del conjunto de output. En la **Fig. 10** mostramos dicha dependencia para todos los experimentos juntos:

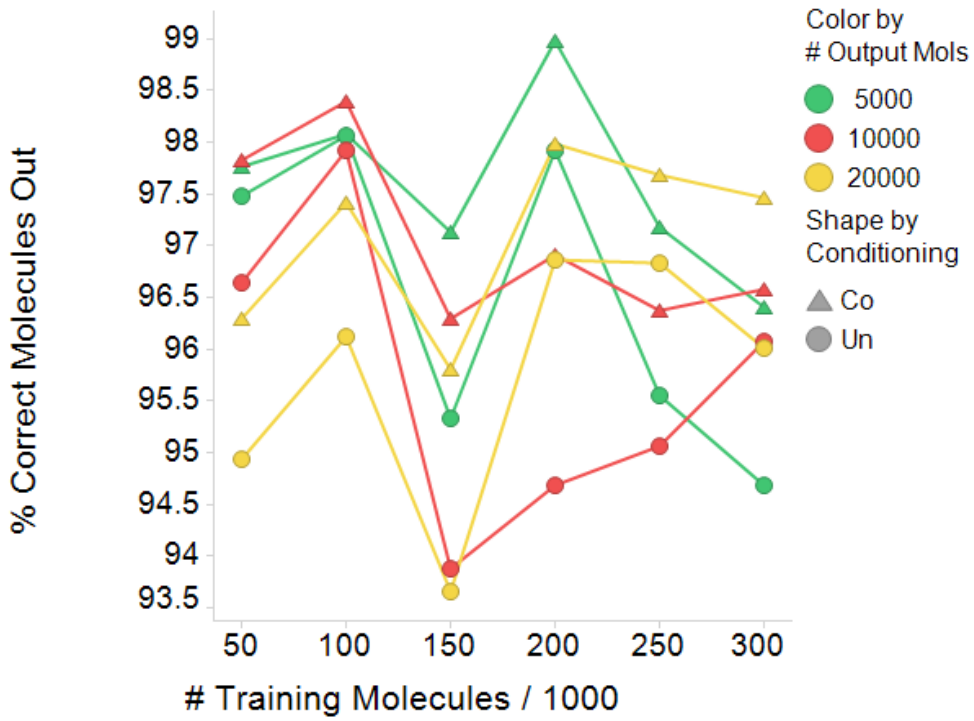


Figura 10. Porcentaje de moléculas correctas del conjunto de output en función del tamaño del conjunto de entrenamiento. Color por tamaño del conjunto de output, y símbolo por condicionamiento.

No parece haber una tendencia muy clara con el tamaño del conjunto de entrenamiento. Si bien es cierto que hay ligeras subidas y bajadas en general para valores concretos del eje de abscisas, todas se dan alrededor de valores muy altos del porcentaje de corrección, y sin una

tendencia global clara con el tamaño del conjunto de entrenamiento (subida generalizada, bajada generalizada, saturación, etc). En cuanto al posible efecto del tamaño del conjunto de output, tampoco se ve una tendencia clara pues los distintos colores se cruzan. Finalmente, vemos una ligera tendencia de los outputs condicionados (triángulos) a ser más correctos que los outputs incondicionados (círculos), pues siempre estos últimos quedan por debajo de los otros.

Diversidad química

Si analizamos las variables de diversidad (número de estructuras únicas, número de clusters y número de frameworks) observamos lo siguiente (**Fig. 11**):

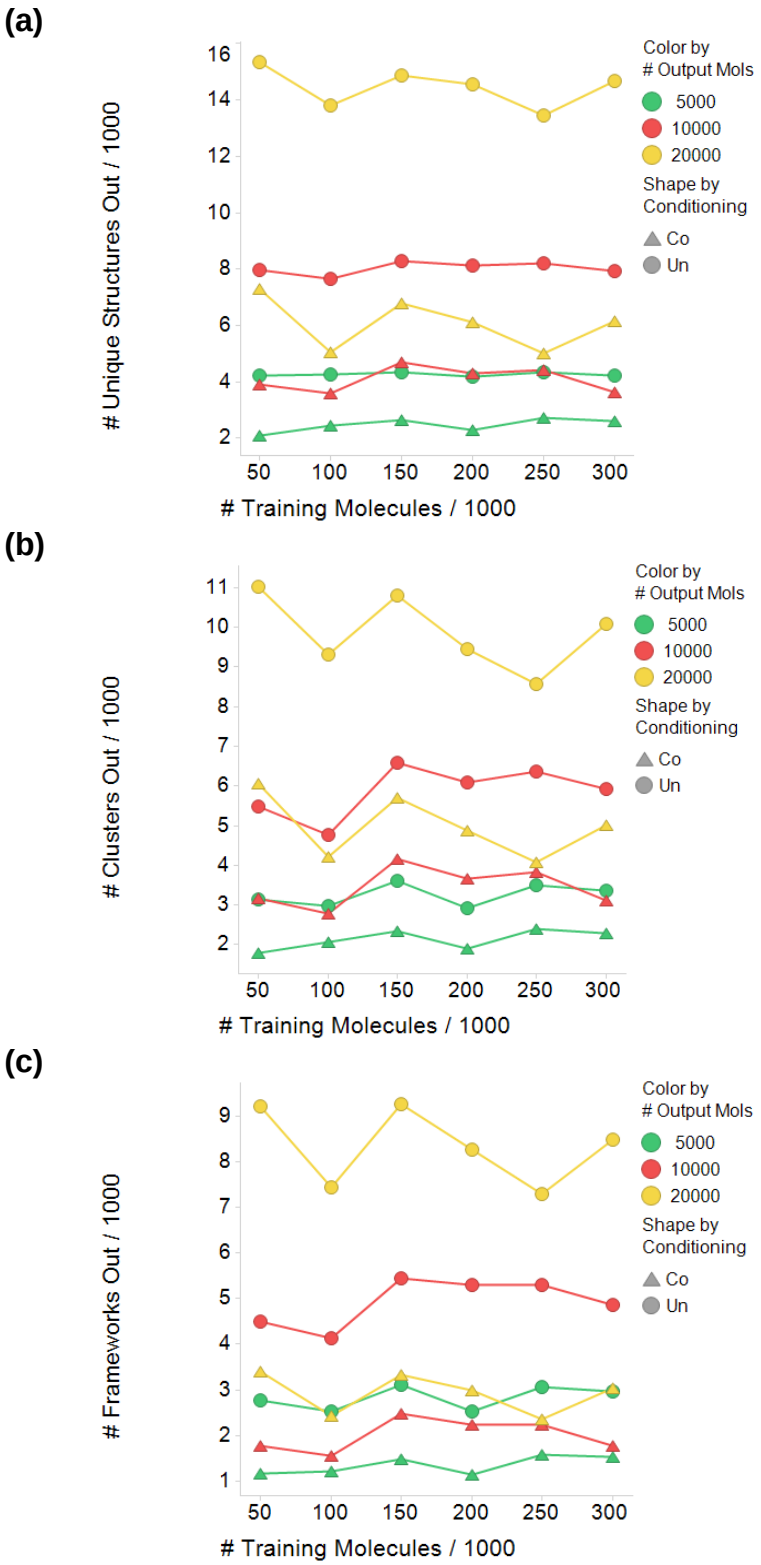


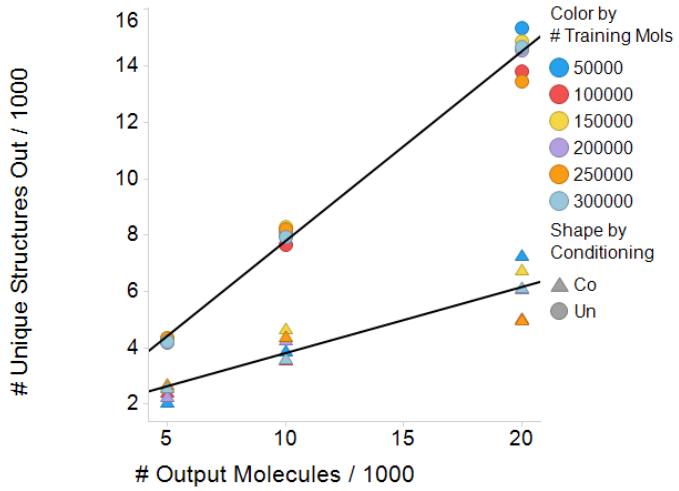
Figura 11. Diversidad de output vs tamaño de conjunto de entrenamiento. (a) Número de estructuras únicas; (b) Número de clusters; (c) Número de frameworks. Coloreado por número de moléculas del output. Símbolos por condicionamiento.

Vemos que no hay tendencia clara a cambiar la diversidad con el tamaño del conjunto de entrenamiento para un tamaño de output y condicionamiento constantes: hay subidas y bajadas pero sin una

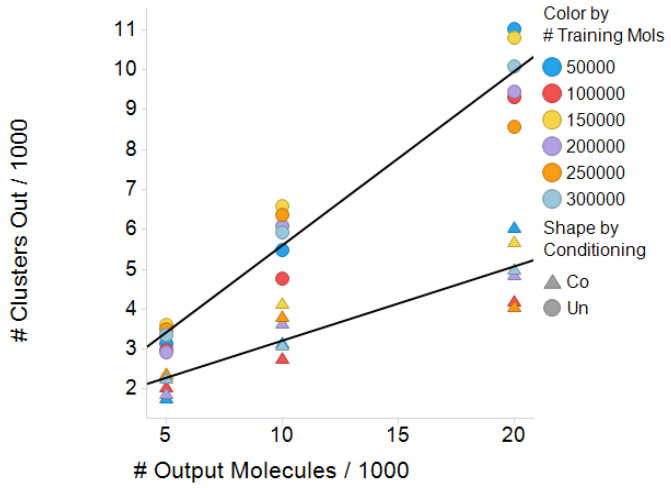
tendencia clara al aumento o a la disminución. Esto significaría que para entrenar el SSVAE sería necesario en principio un conjunto relativamente pequeño de moléculas, del orden de decenas de miles, al menos en cuanto a la generación de un conjunto diverso de moléculas (otra cosa sería en cuanto a la generación condicionada, que según qué valores de las variables condicionantes impongamos podría ser necesario aumentar el número de moléculas de entrenamiento a valores mayores).
En todos los casos observamos que la diversidad del output condicionado es claramente menor que la del output incondicionado, como habíamos visto en la **Sección 2.1**.

Si reanalizamos estos datos en función del tamaño del conjunto de output obtendremos lo siguiente (**Fig. 12**):

(a)



(b)



(c)

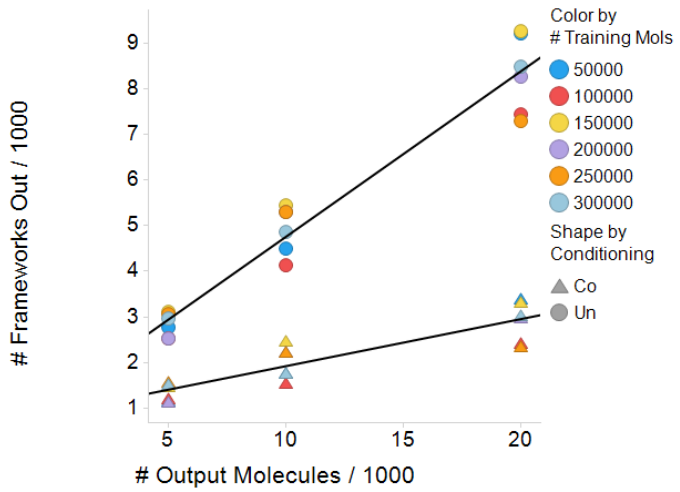


Figura 12. Diversidad de output vs tamaño de conjunto de output. (a) Número de estructuras únicas; (b) Número de clusters; (c) Número de frameworks. Coloreado por tamaño del conjunto de entrenamiento. Símbolos por condicionamiento.

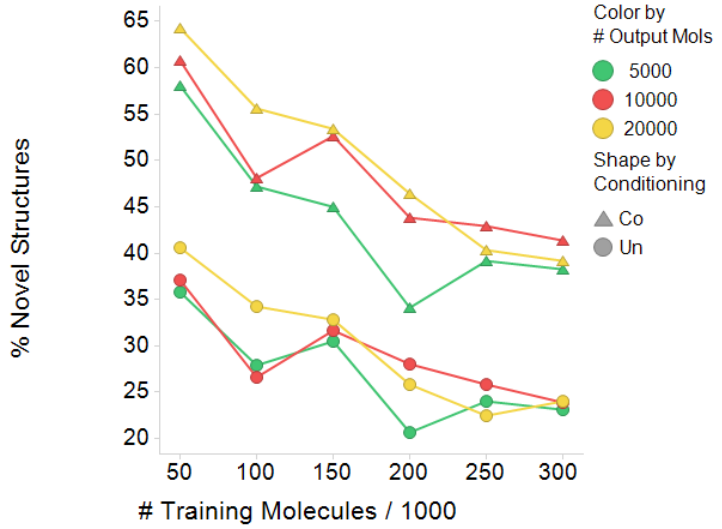
En este caso vemos una tendencia clara lineal a aumentar la diversidad del output con el tamaño del conjunto de output, tanto en el caso incondicionado como en el condicionado. Concluiríamos por tanto que el modelo, al menos con los tamaños de output probados, está lejos de "saturarse" en cuanto a su capacidad de generar diversidad química. Esto, hasta cierto punto, es esperable, dado que estamos generando un número de moléculas output bastante menor que el conjunto de entrenamiento. Si bien estamos muestreando en principio todo el espacio latente, con estos tamaños de output cabe la posibilidad de estar dejando muchos quimiotipos del conjunto de entrenamiento por representar en el output. Realmente una prueba de estrés de la capacidad de generar diversidad química del SSVAE sería la generación de conjuntos de output del orden o más del conjunto de entrenamiento, aunque tal cálculo resultaría extraordinariamente costoso y excede los medios y tiempo de que disponemos para este TFM.

Es de destacar que la pendiente para los outputs incondicionados es mayor que la de los condicionados, de nuevo reflejo de la mayor dificultad de generar moléculas condicionadas, puesto que siempre hay una proporción mayor en este caso de moléculas repetidas.

Novedad química

Si analizamos las variables de novedad (porcentaje de moléculas "nuevas", es decir, sin análogos en el conjunto de entrenamiento, y porcentaje de nuevos frameworks) observamos lo siguiente (**Fig. 13**):

(a)



(b)

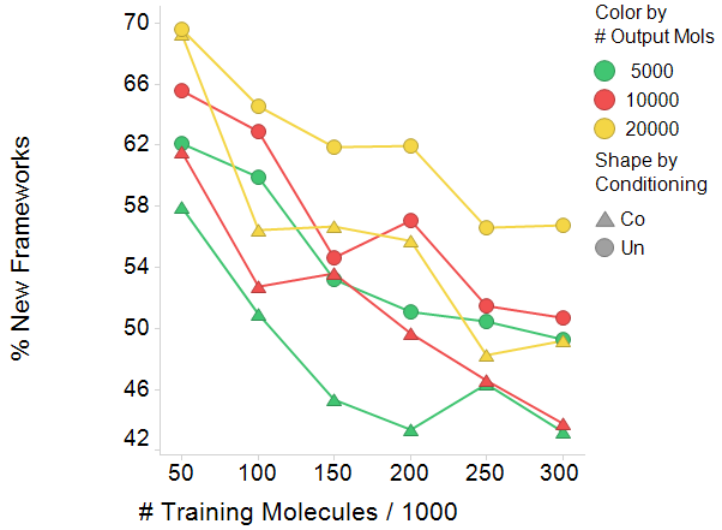


Figura 13. Novedad de output vs tamaño de conjunto de entrenamiento. (a) Porcentaje de estructuras nuevas; (b) Porcentaje de nuevos frameworks. Coloreado por tamaño del conjunto de output. Símbolos por condicionamiento.

Vemos que hay una tendencia clara a la disminuición del porcentaje de estructuras y frameworks nuevos conforme aumentamos el tamaño del conjunto de entrenamiento. Esta observación podría racionalizarse en base a que la generación de estructuras nuevas se hace cada vez más costosa, ya que la nueva molécula debe compararse simultáneamente a un número cada vez mayor de moléculas del conjunto de entrenamiento. Generamos más diversidad al aumentar el tamaño del conjunto de entrenamiento, pero este crece más rápido por lo que disminuye la probabilidad de generar nuevas estructuras.

Si nos enfocamos en el porcentaje de estructuras nuevas, es mayor en la generación condicional que en la incondicional de forma sistemática, y podría explicarse en base a que estamos forzando al modelo a generar moléculas poco representativas del conjunto de entrenamiento (ver "Generación condicionada de moléculas" en la **Sección 2.1**). En el caso

del porcentaje de nuevos frameworks la tendencia se invierte, aunque no hay una separación tan clara como en el caso anterior.

En cuanto a la dependencia de la novedad con el tamaño del output, es posible ver cierta tendencia al aumento del porcentaje de estructuras nuevas y frameworks nuevos conforme aumentamos el mismo (**Fig. 14**):

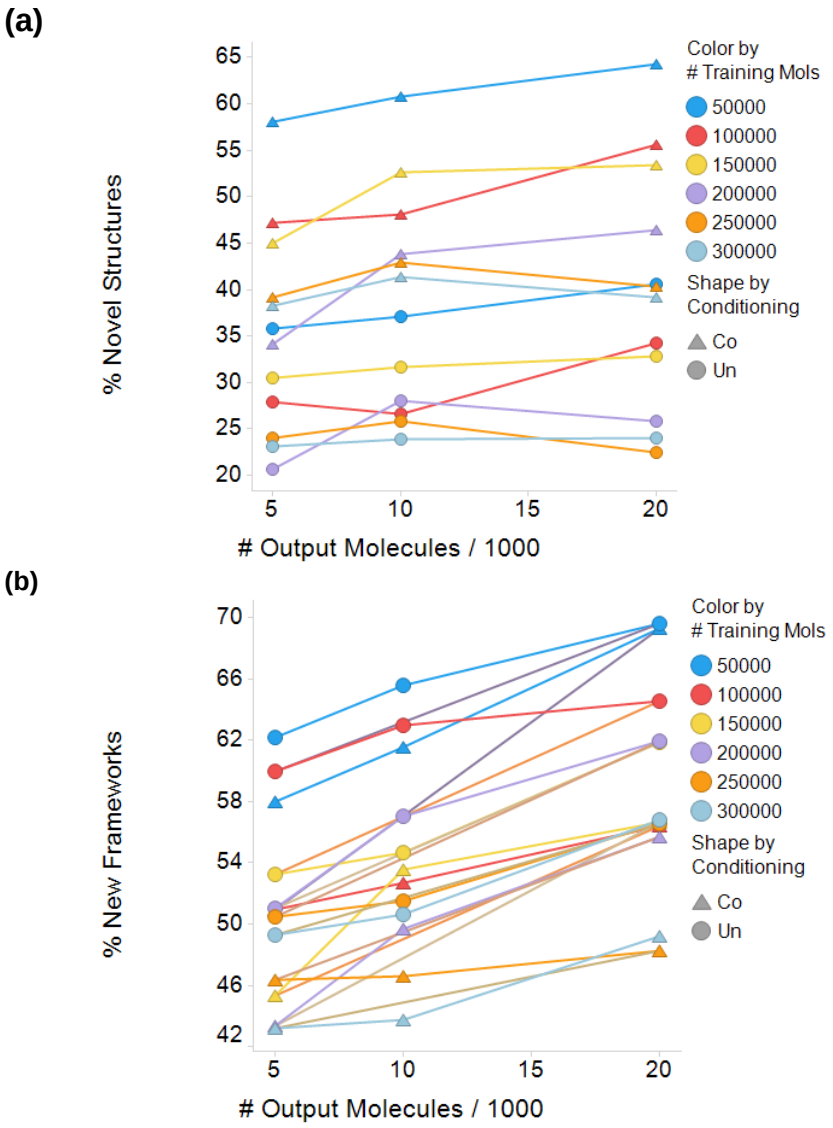


Figura 14. Novedad de output vs tamaño de conjunto de output. (a) Porcentaje de estructuras nuevas; (b) Porcentaje de nuevos frameworks. Coloreado por tamaño del conjunto de entrenamiento. Símbolos por condicionamiento

En este caso estaríamos viendo el efecto contrario, es decir, conforme aumentamos el tamaño del conjunto de output aumentamos la probabilidad de generar moléculas que no estén representadas en el conjunto de entrenamiento. De nuevo, para el porcentaje de nuevas estructuras, la generación condicionada muestra mayores valores, mientras que esta tendencia se invierte para el porcentaje de nuevos frameworks.

Análisis estadístico de los datos

Al objeto de identificar predictores signifcativos para explicar la variabilidad de los parámetros de corrección, diversidad y novedad, se ajustaron modelos lineales a los datos. Se consideraron tres modelos, de complejidad creciente: primero un modelo sin interacciones, de tipo **variable ~ n.train+n.out+con**, donde "n.train" es el tamaño del conjunto de entrenamiento, "n.out" el tamaño del conjunto de output y "con" una variable categórica de condicionamiento. El siguiente modelo que se consideró fue del tipo **variable ~ n.train+n.out*con**, donde el asterisco indica una interacción. Como modelo de máxima complejidad se consideró uno del tipo **variable ~ n.train*n.out*con**, es decir, una triple interacción. Los modelos fueron comparados de menor a mayor complejidad, de forma que se pasaba a uno de mayor complejidad si el correspondiente test F de comparación de los modelos salía significativo. La **Tabla 3** muestra los resultados de este análisis estadístico para todas las variables de diversidad y novedad:

Variable respuesta	Mejor modelo	n.train	n.out	cond	n.out:cond
p.corr.out	p.corr.out~n.train+n.out+con			*	NA
n.un.out	n.un.out~n.train+n.out*con		*		*
n.clus.out	n.clus.out~n.train+n.out*con		*		*
n.fram.out	n.fram.out~n.train+n.out*con		*		*
p.new.str	p.new.str~n.train+n.out+con	*	*	*	NA
p.new.fram	p.new.fram~n.train+n.out+con	*	*	*	NA

Tabla 3. Resultados de análisis estadístico de corrección, diversidad y novedad en función de tamaño de conjunto de entrenamiento y output, y condicionamiento. Las casillas con asterisco indican significación (para un nivel de significancia de 0.05) de la variable predictora en el correspondiente modelo. Las casillas con NA indican la no presencia de esa variable predictora en el modelo. p.corr.out = porcentaje de moléculas correctas en conjunto output; n.un.out = número de moléculas únicas; n.clus.out = número de clusters; n.fram.out = número de frameworks; p.new.str = porcentaje de estructuras nuevas; p.new.fram = porcentaje de nuevos frameworks.

Vemos que para el porcentaje de moléculas output correctas el mejor modelo carece de interacciones y la variable de condicionamiento es significativa, no así n.train y n.out. En el caso de las variables de diversidad química, el mejor modelo es del tipo n.train+n.out*con, y tanto el tamaño del conjunto output como la interacción n.out:con son significativas. Finalmente, para las variables de novedad química el mejor modelo es el más simple sin interacciones, y las tres variables, n.train, n.out y con son significativas.

Concluimos por tanto que la corrección química del output del SSVAE depende del condicionamiento (es significativamente mayor con las moléculas condicionadas). La diversidad química depende significativamente del tamaño del conjunto de output (aumenta linealmente con el mismo), pero de forma diferente según el condicionamiento: más rápido con moléculas incodicionadas que con las

condicionadas. Finalmente, la novedad química depende significativamente tanto del tamaño del conjunto de entrenamiento (disminuye con el mismo), el del conjunto output (aumenta con el mismo) y el condicionamiento (la intercepta es mayor con la generación incondicionada para el porcentaje de moléculas nuevas, y menor para el porcentaje de frameworks nuevos).

2.3. Análisis del output del SSVAE en función de la diversidad "intensiva" del conjunto de entrenamiento

Hemos visto en la sección anterior que el aumento del tamaño del conjunto de entrenamiento, y por tanto, de su diversidad, no parece aumentar la diversidad del output del SSVAE, al menos para los tamaños de output que estamos generando. Pero cabe preguntarse si esta constancia se observaría también si partiéramos, para un mismo tamaño del conjunto de entrenamiento, de situaciones de "diversidad intensiva" diferente. Por esto entendemos conjuntos de entrenamiento del mismo tamaño pero con diferente redundancia interna. En un extremo, podríamos partir de un conjunto de entrenamiento en el que ninguna de las moléculas fueran análogas entre sí, es decir, que ninguna ellas mostrara similitud igual o mayor a 0.7 con ninguna otra de las moléculas del conjunto. Este sería el conjunto "*ortogonal*". En el otro extremo, podríamos partir de un conjunto de entrenamiento en el que las moléculas estuvieran organizadas en grandes clusters. Este sería el conjunto "*clusterizado*". Compararíamos los resultados con ambos conjuntos de entrenamiento para ver posibles diferencias.

Este experimento se realizó partiendo del conjunto de compuestos "clean lead-like" de ZINC-12, un total de 4591276 moléculas ("clean" significa que son un subconjunto del que han sido eliminadas moléculas "lead-like" con determinadas subestructuras reactivas/promiscuas). Para generar el conjunto ortogonal se creó un algoritmo y función (**`divsamp0`**, dentro del módulo **`myfuncs.py`**). Este algoritmo selecciona la primera molécula dentro de un conjunto de moléculas previamente aleatorizado (incluye una semilla para el caso en que se quieran obtener resultados reproducibles) y la añade al conjunto ortogonal. Entonces va calculando la similitud de Tanimoto de esta molécula con las sucesivas moléculas hasta que encuentra una con una similitud por debajo de un umbral (e.g. 0.7). Entonces esta molécula se añade al conjunto ortogonal. Este proceso se repite, en cada iteración evaluando la similitud de la siguiente molécula con todas las moléculas del conjunto ortogonal hasta el momento, hasta que éste llega a tener un tamaño previamente definido. En nuestro caso, generamos un conjunto de entrenamiento de 310000 moléculas con un umbral de 0.7.

Para el conjunto clusterizado se clusterizó el conjunto "clean lead-like" de ZINC-12 mediante la función **`clusmidf`** que implementa el algoritmo de Taylor-Butina, utilizando un umbral de 0.8. El output de esta función proporciona los clusters organizados de mayor a menor, por lo que se

cogieron las primeras 310000 moléculas al objeto de obtener un conjunto de compuestos con los mayores clusters posibles.

En ambos casos se calcularon el peso molecular, el logP y el QED de todas las moléculas. Con estos dos conjuntos de entrenamiento se entrenó el SSVAE, generando 10000 moléculas incondicional y condicionalmente (en este caso, poniendo como objetivo un peso molecular de 250 Da). Se hicieron en ambos casos sucesivas iteraciones con subconjuntos crecientes de los dos conjuntos de entrenamiento, de 100K, 200K y 300K moléculas. El resultado se analizó con modelos lineales múltiples (ver subsección Análisis Estadístico al final).

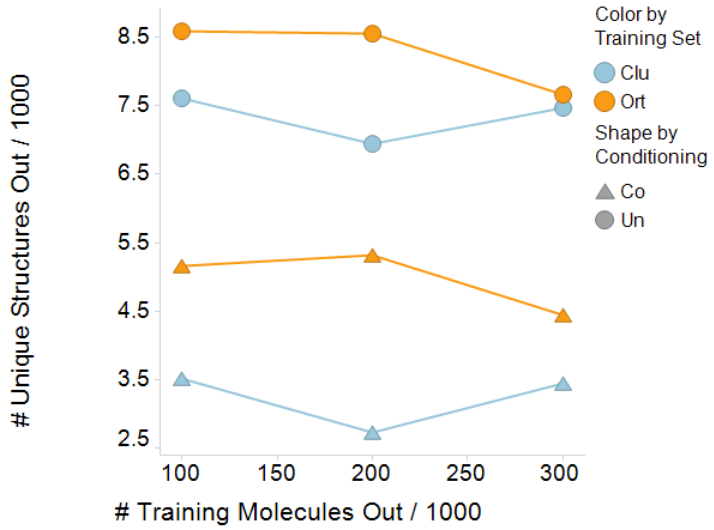
Corrección química

El porcentaje de moléculas correctas no dependió significativamente ni del grupo de diversidad intensiva de entrenamiento, ni del tamaño del conjunto de entrenamiento, ni del condicionamiento (datos no mostrados).

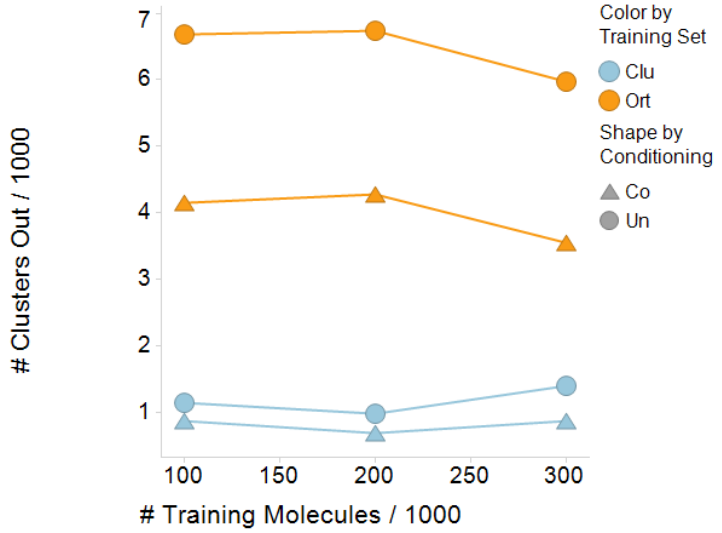
Diversidad química

Presentamos a continuación los parámetros de diversidad química frente al tamaño del conjunto de entrenamiento, para el experimento ortogonal vs el clusterizado, con generación incondicional y condicional (**Fig. 15**):

(a)



(b)



(c)

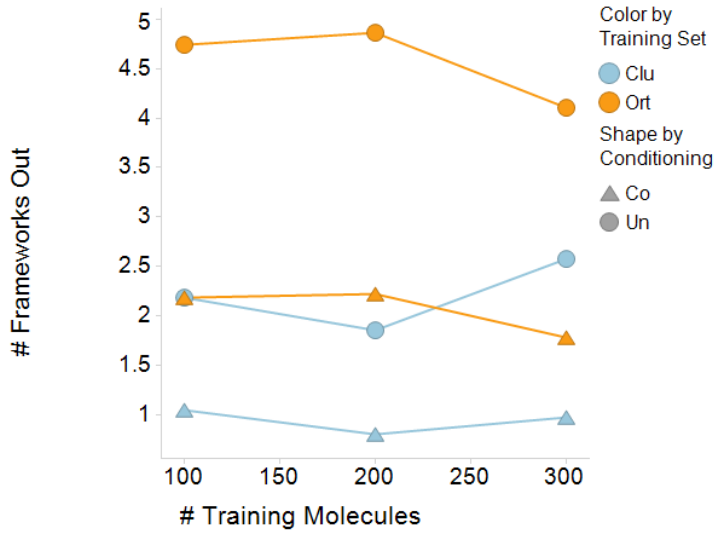


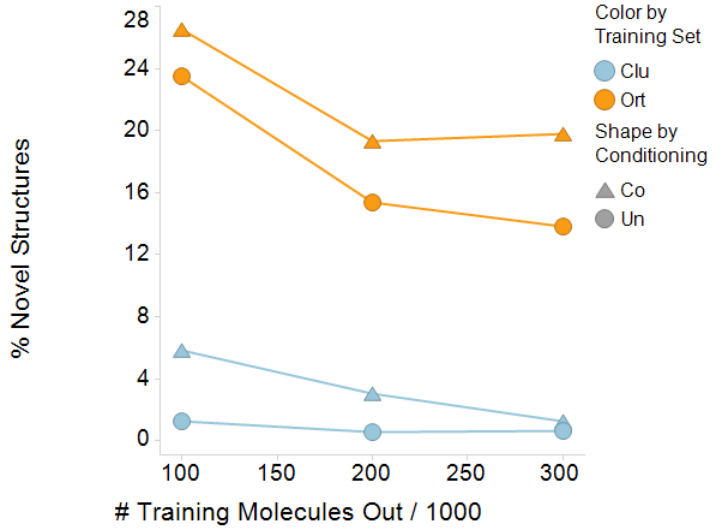
Figura 15. Diversidad de output vs diversidad intensiva de conjunto de entrenamiento, para tres tamaños de conjunto de entrenamiento. (a) Número de estructuras únicas; (b) Número de clusters; (c) Número de frameworks. Coloreado por tipo de conjunto de entrenamiento. Símbolos por condicionamiento.

Se observa una disminución dramática de la diversidad química en el conjunto clusterizado vs el conjunto ortogonal, especialmente en los casos de los clusters y los frameworks. La generación incondicionada también muestra mayor diversidad en todos los casos, aunque en el caso de clusters nuevos del grupo clusterizado los valores de incondicionado y condicionado están muy cercanos.

Novedad química

Las variables de novedad química se presentan a continuación (**Fig. 16**):

(a)



(b)

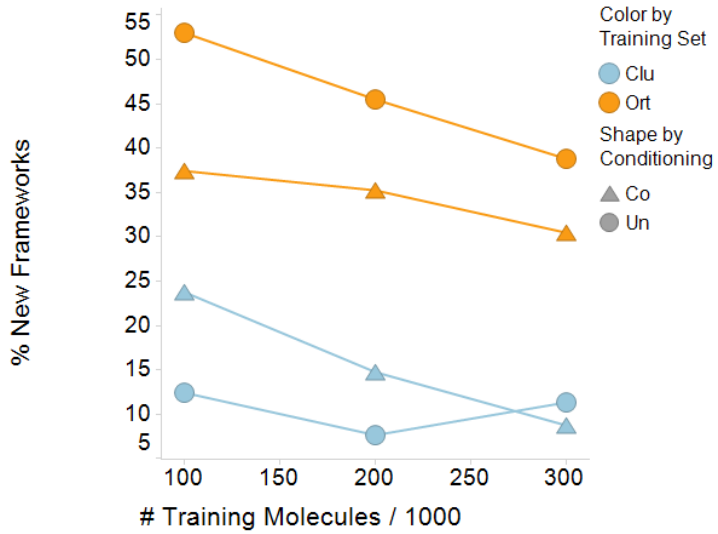


Figura 16. Diversidad de output vs diversidad intensiva de conjunto de entrenamiento, para tres tamaños de conjunto de entrenamiento. (a) Porcentaje de estructuras nuevas; (b) Porcentaje de nuevos frameworks. Coloreado por tipo de conjunto de entrenamiento. Símbolos por condicionamiento.

De nuevo se observa una disminución clara de la novedad química del conjunto clusterizado vs el ortogonal. Como se observó anteriormente, la generación condicionada para las nuevas estructuras muestra mayor novedad que la incondicionada, mientras que la tendencia se invierte con el porcentaje de nuevos frameworks.

Análisis estadístico de los datos

Se realizó un análisis por ANCOVA múltiple de los datos similar al de la sección anterior. En este caso, las variables predictoras consideradas fueron el tamaño del conjunto de entrenamiento (n.train), el grupo de diversidad química (ortogonal vs clusterizado, gr) y el condicionamiento (con). Se consieraron tres modelos de creciente complejidad: un modelo sin interacciones (**variable ~ n.train+gr+con**), un modelo con interacción gr*con (**variable ~ n.train+gr*con**), y un modelo con

interacción triple (**variable ~ n.train*gr*con**). La selección del modelo final para cada variable fue por test F, comparando la suma de cuadrados residuales del modelo de menor complejidad con el del siguiente en complejidad y aceptando el último si el test resultaba significativo ($p < 0.05$). Mostramos los resultados (**Tabla 4**):

var	best.mod	n.train	gr	con	gr:con
p.corr.out	p.corr.out~n.train+gr+con				NA
n.un.out	n.un.out~n.train+gr+con		*	*	NA
n.clus.out	n.clus.out~n.train+gr*con		*		*
n.fram.out	n.fram.out~n.train+gr*con		*	*	*
p.new.str	p.new.str~n.train+gr+con	*	*	*	NA
p.new.fram	p.new.fram~n.train+gr*con	*	*		*

Tabla 4. Resultados de análisis estadístico de corrección, diversidad y novedad en función de la diversidad intensiva del conjunto de entrenamiento, su tamaño y el condicionamiento. Las casillas con asterisco indican significación (para un nivel de significancia de 0.05) de la variable predictora en el correspondiente modelo. Las casillas con NA indican la no presencia de esa variable predictora en el modelo. gr = grupo de diversidad (ortogonal vs clusterizado); resto de variables como en **Tabla 3**.

Como decíamos arriba, el porcentaje de moléculas correctas no depende de ninguna variable, en particular el grupo de diversidad intensiva. Para el resto de los casos el grupo de diversidad intensiva es significativo: hay mayor diversidad y novedad con el conjunto de entrenamiento ortogonal. En el caso de las variables de diversidad hay distintos comportamientos: para el número de moléculas únicas no hay interacciones significativas, y el condicionamiento también es significativo (el número es significativamente menor para la generación condicionada); en el caso del número de clusters y frameworks, se da una interacción del grupo de diversidad con el condicionamiento: aunque se dan valores mayores con la generación incondicionada, la diferencia incondicionada-condicionada es mayor con el entrenamiento ortogonal que con el entrenamiento clusterizado. Finalmente, para las variables de novedad n.train también es significativo (como vimos en la **Sección 2.2**). En el caso del porcentaje de estructuras nuevas, no hay interacciones y también es significativo el condicionamiento: la generación condicionada resulta en valores repetidamente mayores de porcentaje. El porcentaje de frameworks nuevos muestra una interacción grupo de diversidad con el condicionamiento: para el grupo ortogonal, la generación incondicionada resulta en valores mayores, y esta tendencia se invierte en el grupo clusterizado.

Podemos concluir que la diversidad intensiva del conjunto de entrenamiento tiene una influencia decisiva en la diversidad y novedad del output: para generar un output diverso y novedoso es necesario partir de un conjunto de entrenamiento con moléculas muy diversas, y no sería tan importante el tamaño del conjunto de entrenamiento. Es posible que este último fuera importante para la generación de outputs del orden del conjunto de training o mayores (cientos de miles o millones de moléculas) pero el tiempo de cálculo requerido excede el disponible para

este TFM. Además, se da en algunos casos una interacción significativa entre la diversidad intensiva y el condicionamiento; tal es el caso del número de clusters y de frameworks, y así como del porcentaje de frameworks nuevos.

2.4. Modificación del SSVAE para generar productos naturales

Los productos naturales son moléculas obtenidas a partir de organismos vivos.[65,66] Especialmente interesantes para el descubrimiento de fármacos son los llamados metabolitos secundarios, que son moléculas producidas para funciones de protección y competición frente a otras especies, y que no son necesarias para la supervivencia inmediata del organismo productor. Estas moléculas han sido seleccionadas para atravesar membranas biológicas e interaccionar con dianas biológicas, y por ello muestran, aparte de una alta diversidad y diferencias estructurales respecto de las moléculas sintéticas, propiedades fisicoquímicas interesantes para guiar el diseño de moléculas. Entre ellas están un menor logP, una distribución más amplia de TPSA, así como una mayor hidrofilicidad y un número mayor de estererocentros.[66] También suele haber una mayor abundancia de moléculas de alto peso molecular, alto número de ramificaciones y sistemas de anillos complicados.

Es por ello que sus correspondientes códigos SMILES muestran una complejidad mayor, pues en ellos abundan los símbolos de estereoquímica, así como números enteros (que codifican el inicio y final de anillos, ver **Anexo I**) elevados. Es por esto que el SSVAE de Kang y Cho, con un diccionario de sólo 35 símbolos, no puede modelar productos naturales, y es necesario modificarlo de cara a poder trabajar con este tipo de moléculas.

En este sentido, por un lado se creó un notebook de Python (**`generateNP.ipynb`**) para definir un conjunto de productos naturales lo más amplio posible que pudiera ser modelable adaptando el SSVAE. Se partió del conjunto de productos naturales de ZINC-12, un total de 85200 moléculas. Los SMILES correspondientes se recodificaron en su versión no isomérica, para eliminar los símbolos de esteroisomería en los mismos. A continuación, se identificaron las moléculas con números enteros de tres cifras (que corresponden a anillos trifurcados) y se eliminaron, resultando en 84605 moléculas. Posteriormente, se identificaron los números enteros de dos cifras representados en menos de 50 moléculas (un total de 65), y las moléculas que los contenían también se eliminaron, resultando en un total final de 84384 moléculas. De esta forma, se redujo ligeramente (menos del 1%) el conjunto de entrenamiento de productos naturales pero a su vez se eliminaron una gran cantidad de símbolos en los SMILES con pocas instancias en el conjunto de entrenamiento que hubieran aumentado enormemente el diccionario del SSVAE y hubieran sido muy difíciles de entrenar.

La otra parte fue, entonces, modificar el diccionario del SSVAE para incluir los nuevos símbolos, principalmente los enteros de dos cifras. El diccionario resultante contiene un total de 59 símbolos:

{'',1,2,3,4,5,6,7,8,9,,#,(,),[,],+,=,B,Br,c,C,Cl,F,H,I,N,n,O,o,P,p,S,s,Si,Sn,12
,13,14,15,21,23,24,25,31,32,34,35,36,41,42,43,45,46,52,53,54,56,65,67}

Con estas modificaciones el SSVAE pudo modelar productos naturales. Se realizaron entrenamientos con los primeros 50K productos naturales y con el total de los productos naturales, y se generaron 5K moléculas incondicional y condicionalmente (peso molecular de 250 Da).

Conjunto de entrenamiento

La **Fig. 17** muestra la distribución de peso molecular vs logP del conjunto de entrenamiento, donde podemos ver que en este caso el peso molecular está centrado en aproximadamente 400 Da y el logP en alrededor de 3, si bien cabe destacar el amplio rango de valores tanto de peso molecular como de logP que hay en el conjunto, llegando a pesos moleculares alrededor del 1000 Da y de logP de hasta 20 o -10:

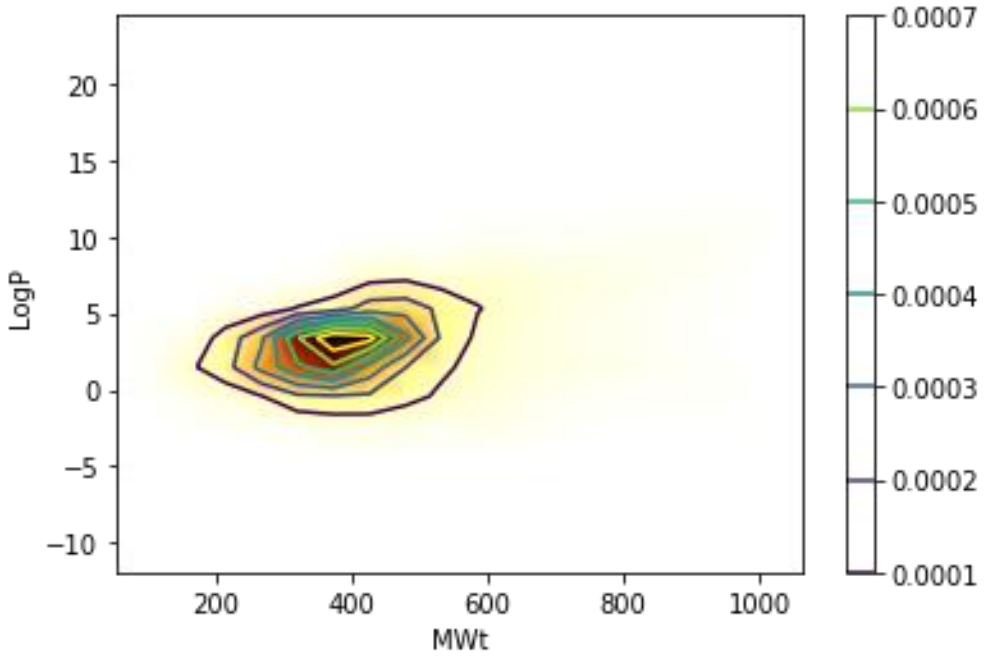


Figura 17. Distribución de peso molecular vs logP del conjunto de entrenamiento de productos naturales

Generación incondicionada

La **Fig. 18** muestra la distribución de peso molecular vs logP del conjunto de output incondicionado:

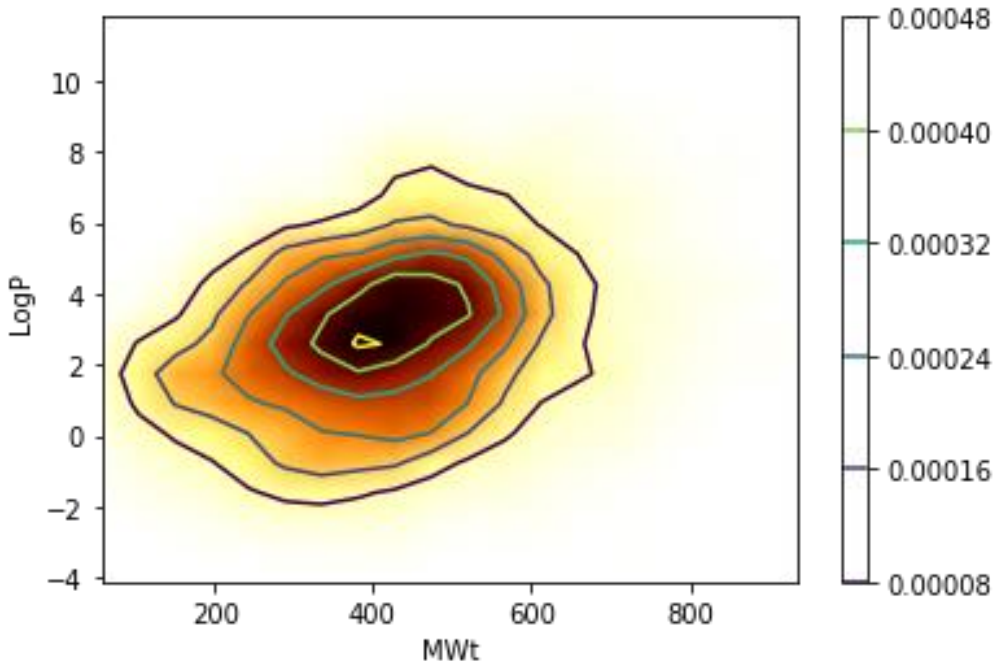


Figura 18. Distribución de peso molecular vs logP del conjunto de output incondicionado de productos naturales

Vemos que de nuevo la distribución incondicionada es similar a la del conjunto de input. Las variables de diversidad y novedad resultantes fueron (**Tabla 5**):

# mol input	%corr	# mol unicas	# clusters	# frames	% nuevas estr	% nuevos frames
50000	95.04	2948	1496	1744	6.65	15.25
81661	95.18	3054	1596	1825	5.83	14.41

Tabla 5. Resultados de corrección, diversidad y novedad de la generación incondicionada tras entrenamiento con productos naturales (5000 moléculas de output)

Se observa un gran número de moléculas repetidas, alrededor de un 35%, resultando en un número de clusters y frameworks comparativamente menor. Asimismo, el porcentaje de nuevas estructuras y frameworks es bastante bajo, en comparación al conjunto de entrenamiento de Kang y Cho. Como posible explicación a esta baja diversidad y novedad podríamos aducir la complejidad de los productos naturales, de forma que además, con un conjunto de entrenamiento tan pequeño es difícil entrenar a la red para generar una gran diversidad de estructuras complejas. Cabe por otra parte destacar que se observan más frameworks que clusters, lo cual puede ser debido a que al generar este tipo de moléculas complejas se produzca una mayor variabilidad en el posible framework en comparación con la similitud global de la molécula, de forma que podría haber moléculas en el mismo cluster con diferente framework.

Al objeto de hacerse una idea de las moléculas que el SSVAE estaba generando, se creó un notebook (**`Exp3MolVisualize.ipynb`**) para visualizar los distintos clusters de moléculas. Se representaron los 10 primeros y se pudo ver que en todos los casos las moléculas eran de tipo producto natural. La **Fig. 19** muestra algunas moléculas del primer cluster, mostrando la alta complejidad característica de los productos naturales:

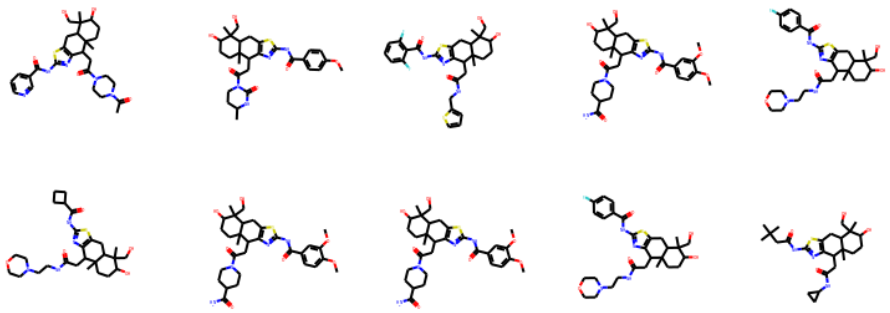


Figura 19. Estructuras de moléculas del primer cluster del output incondicionado de productos naturales

Generación condicionada (peso molecular 250 Da)

La distribución de peso molecular vs logP es dramáticamente alterada en la generación condicionada (**Fig. 20**), con la media del peso molecular alrededor de 250 Da y el logP cerca de 2:

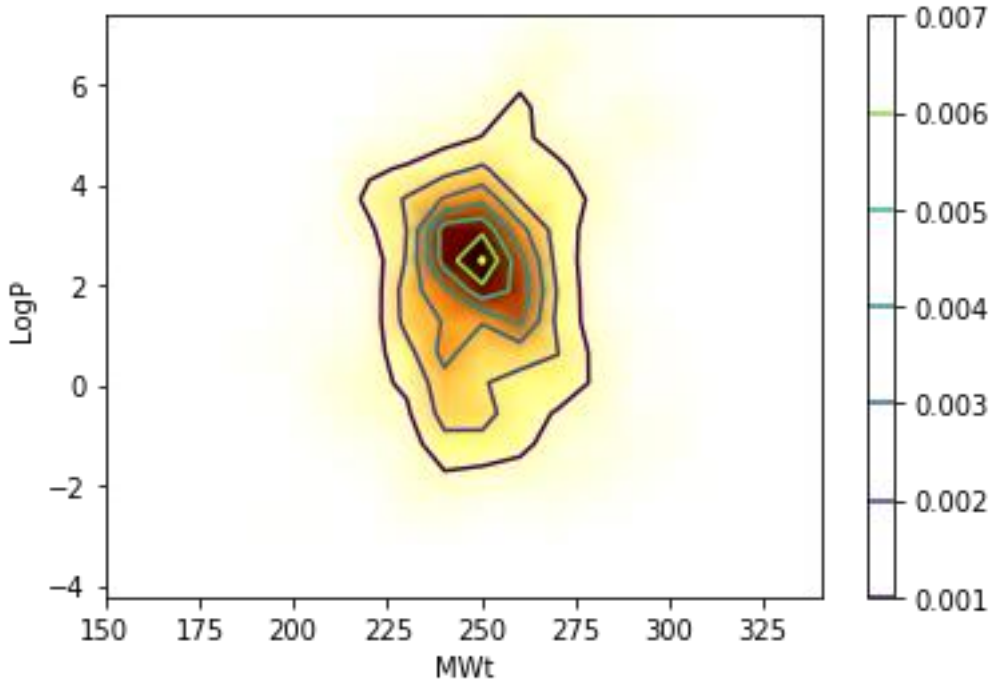


Figura 20. Distribución de peso molecular vs logP del conjunto de output condicionado de productos naturales

Las variables de diversidad y novedad resultantes fueron (**Tabla 6**):

# mol input	%corr	# mol unicas	# clusters	# frames	% nuevas estr	% nuevos frames
50000	95.34	1100	744	534	17.09	26.59
81661	96.88	1039	723	516	16.84	31.4

Tabla 6. Resultados de corrección, diversidad y novedad de la generación condicionada tras entrenamiento con productos naturales (peso molecular diana de 250 Da, 5000 moléculas de output)

De nuevo la diversidad disminuye aún más en la generación condicionada, aunque la novedad aumenta. Visualizando los 10 primeros clusters se observa que sólo la mitad son de tipo producto natural, lo cual es esperable, dado que estamos condicionando el peso molecular al típico de fragmentos o leads. Por ejemplo, en el cluster 8 observamos estas estructuras (**Fig. 21**), que son claramente de tipo “lead-like”:

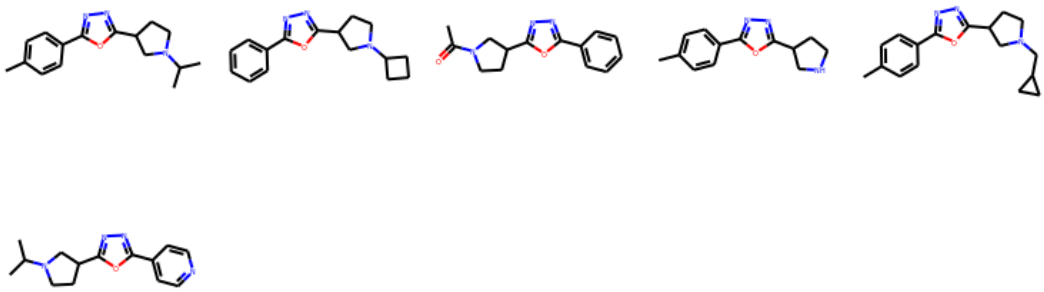


Figura 21. Estructuras de moléculas del cluster 8 del output condicionado de productos naturales

En conclusión, podemos decir que la modificación del SSVAE para generar productos naturales ha sido un éxito, aunque se trata de moléculas complejas y con un conjunto de entrenamiento menor, por lo que la diversidad y novedad del ouput están disminuidas en comparación con las moléculas “drug-like”. La posibilidad de crear versiones condicionadas (e.g. menores pesos moleculares) de productos naturales es muy interesante, ya que podríamos beneficiarnos de las buenas propiedades de éstos (polaridad, abundancia de esterocentros) eliminando las indeseadas (gran tamaño y complejidad).

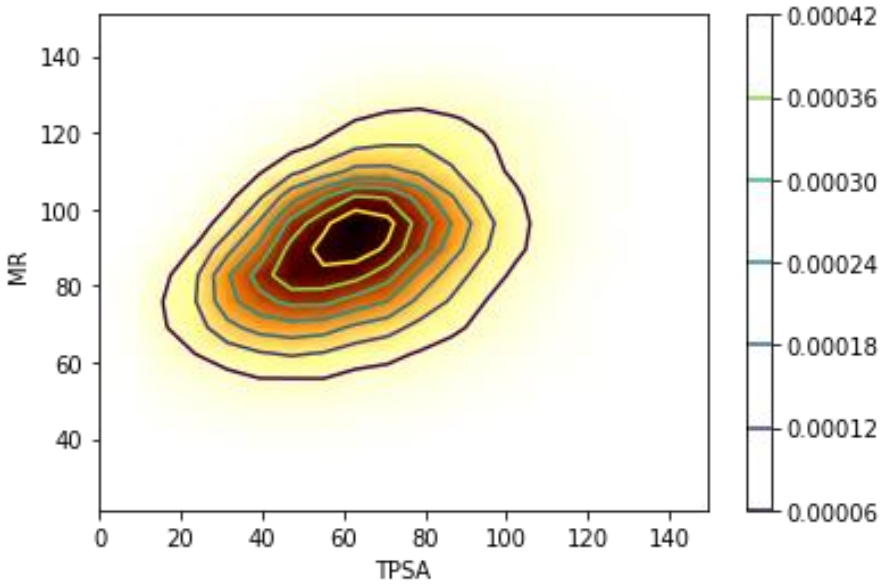
2.5. Entrenamiento del SSVAE con propiedades alternativas: TPSA, MR y LASA

Hasta ahora hemos estado utilizado como propiedades para entrenar al SSVAE el peso molecular, el logP y el QED, tal y como hicieron Kang y Cho. Cabe preguntarnos acerca del comportamiento de la red con otras propiedades.

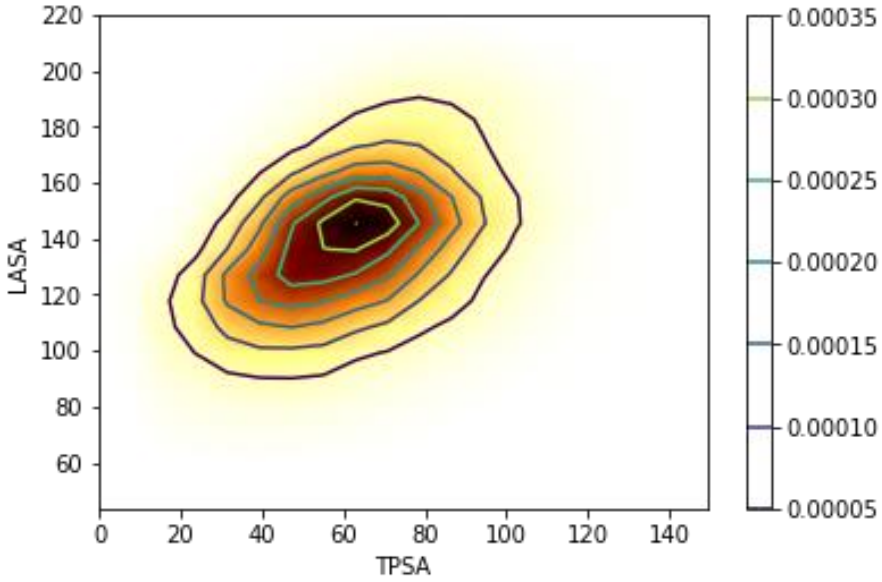
En este sentido, se experimentó con las propiedades TPSA (Topological Polar Surface Area,[56] una medida de la superficie de los átomos polares en una molécula), MR (Molar Refractivity, una medida de la polarizabilidad de un molécula) y Labute ASA (Approximate Surface

Area de Labute[57]). Para ello, se creó un notebook (**`GenerateExp5Set.ipynb`**) en el cual, partiendo de las moléculas "clean drug-like" de ZINC-12, seleccionaron al azar un conjunto de 310K moléculas y se calcularon estas propiedades con RDKit. Las distribuciones de propiedades observadas en este conjunto de entrenamiento son las siguientes (**Fig. 22**):

(a)



(b)



(c)

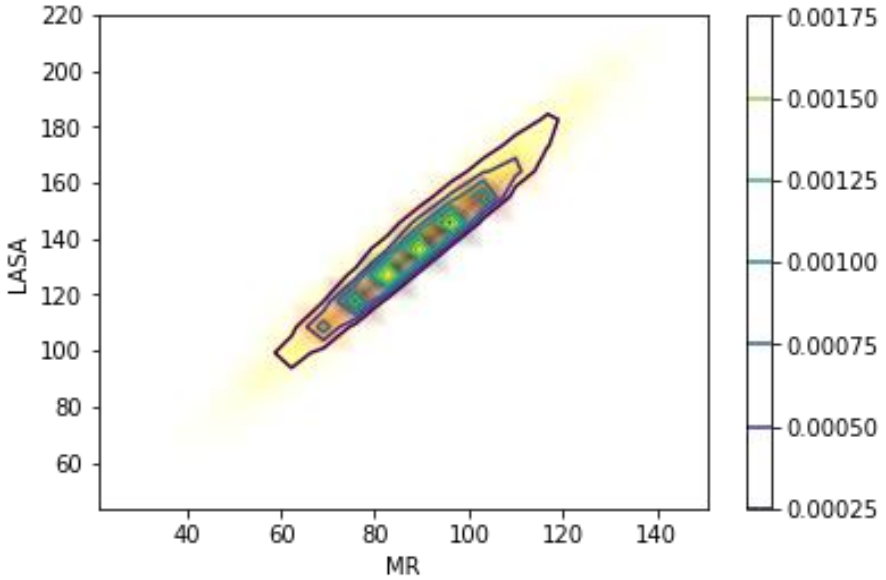


Figura 22. Distribución de propiedades de conjunto de entrenamiento con TPSA, MR y LASA: (a) TPSA vs MR; (b) TPSA vs LASA; (c) MR vs LASA

Puede verse que la TPSA está centrada aproximadamente en 60, la MR en 90, y la LASA en 140. Se observa cierta correlación entre TPSA y MR (coeficiente de correlación de Pearson de 0.32), y entre TPSA y LASA (correlación de 0.40) , y mucha entre MR y LASA (correlación de 0.98).

Se analizó el comportamiento del SSVAE entrenado con estas propiedades. Para la generación condicionada, se utilizó un doble condicionamiento: TPSA de 100 y MR de 80 (la modificación del SSVAE para tratar condicionamientos con dos o más variables se describe en la sección **2.6**). Se generaron 10K moléculas de output, tanto incondicionada como condicionadamente.

Generación incondicionada

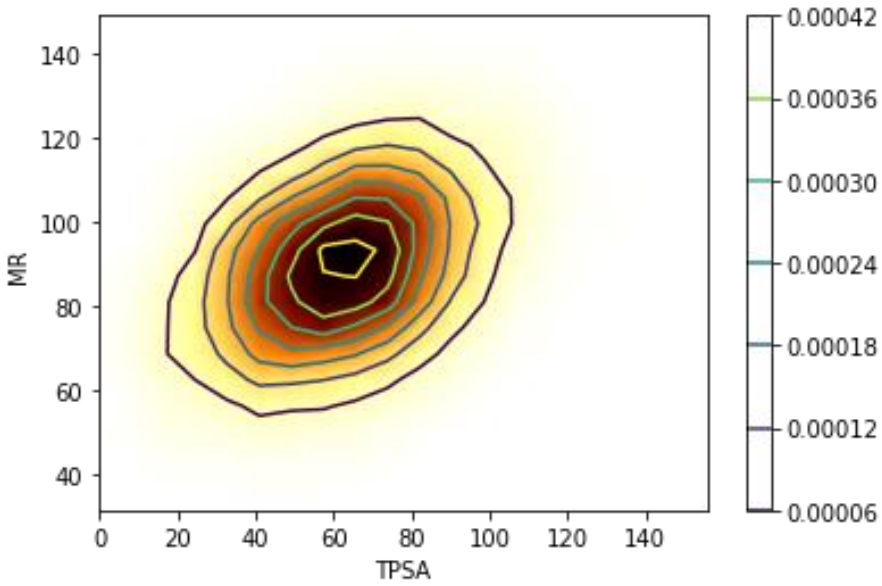
En la **Tabla 7** se muestran los resultados de corrección, diversidad y novedad de la generación incondicionada:

# mol input	%corr	# mol unicas	# clusters	# frames	% nuevas estr	% nuevos frames
10000	99.35	9527	7552	5645	13.81	48.38

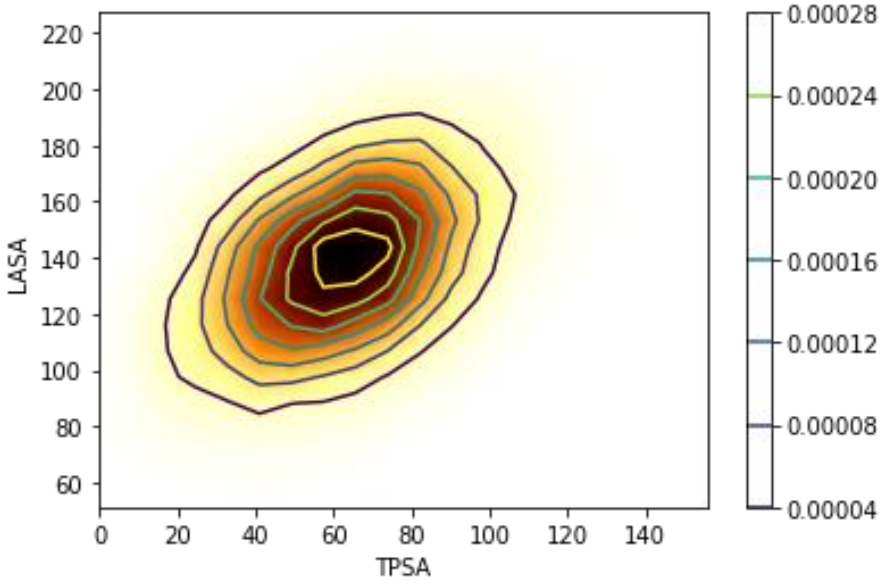
Tabla 7. Resultados de corrección, diversidad y novedad de la generación incondicionada tras entrenamiento con TPSA, MR y LASA

En este caso vemos que hay muy pocas moléculas repetidas (alrededor de un 4%) y bastante diversidad química. La novedad es relativamente baja. Son resultados similares a los obtenidos con peso molecular, logP y QED. Las distribuciones de propiedades con moléculas generadas incondicionalmente son similares a las del conjunto de entrenamiento (**Fig. 23**):

(a)


MR
TPSA
0
20
40
60
80
100
120
140
40
60
80
100
120
140
0.00042
0.00036
0.00030
0.00024
0.00018
0.00012
0.00006

(b)


LASA
TPSA
0
20
40
60
80
100
120
140
60
80
100
120
140
160
180
200
220
0.00028
0.00024
0.00020
0.00016
0.00012
0.00008
0.00004

(c)

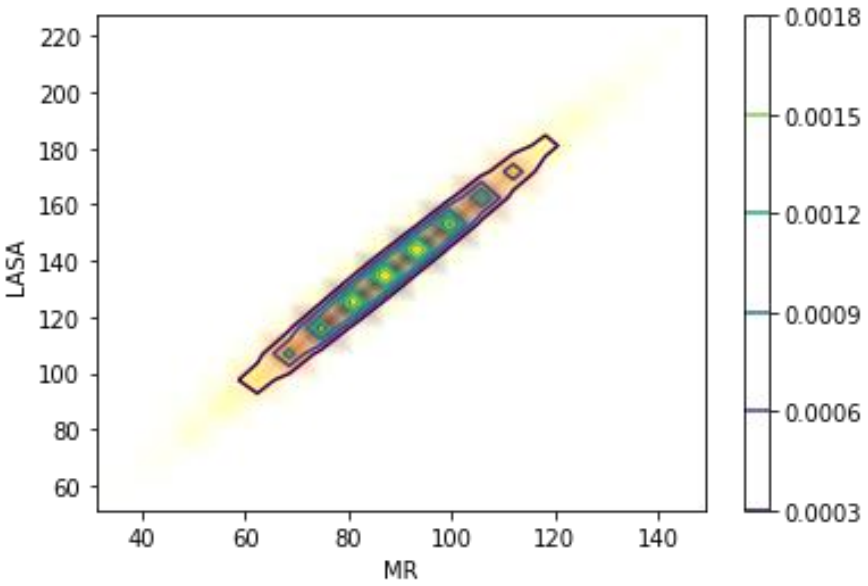


Figura 23. Distribución de propiedades de conjunto de output incondicionado para entrenamiento con TPSA, MR y LASA: (a) TPSA vs MR; (b) TPSA vs LASA; (c) MR vs LASA

Generación condicionada (TPSA = 100 y MR = 80)

La **Tabla 8** muestra los resultados de corrección, diversidad y novedad de la generación condicionada:

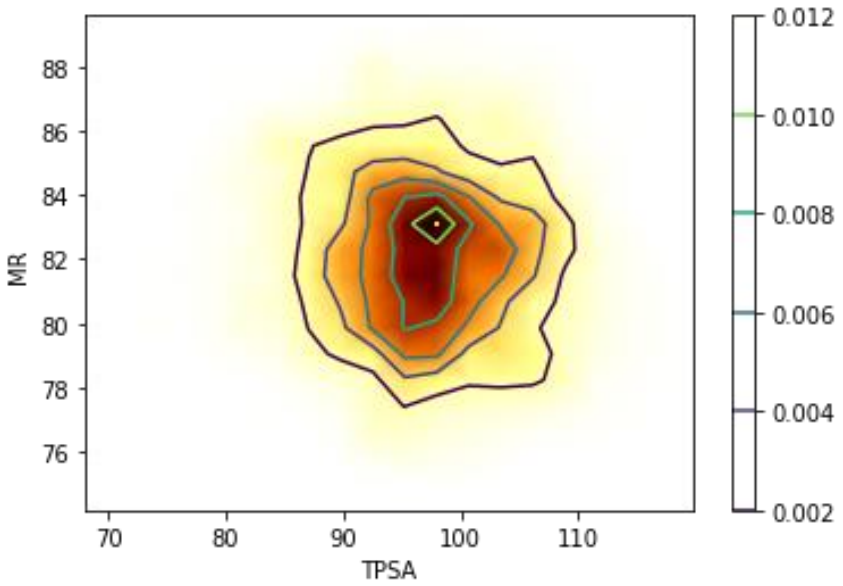
# mol input	%corr	# mol unicas	# clusters	# frames	% nuevas estr	% nuevos frames
10000	97.99	3271	2370	1692	43.53	55.26

Tabla 8. Resultados de corrección, diversidad y novedad de la generación condicionada tras entrenamiento con TPSA, MR y LASA

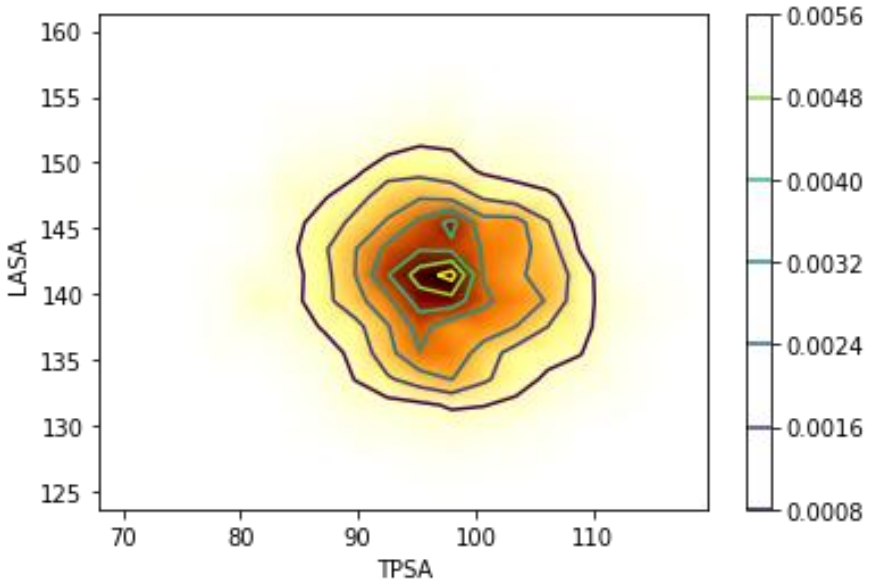
Se observa la habitual disminución de diversidad de la generación condicionada. En el caso de la novedad, vemos un aumento, tanto de las nuevas estructuras como de los nuevos frameworks.

Finalmente, a partir de las distribuciones de propiedades del output podemos ver que la generación doble-condicionada ha sido un éxito (**Fig. 24**):

(a)


MR
TPSA
88
86
84
82
80
78
76
70
80
90
100
110
0.012
0.010
0.008
0.006
0.004
0.002

(b)


LASA
TPSA
160
155
150
145
140
135
130
125
70
80
90
100
110
0.0056
0.0048
0.0040
0.0032
0.0024
0.0016
0.0008

(c)

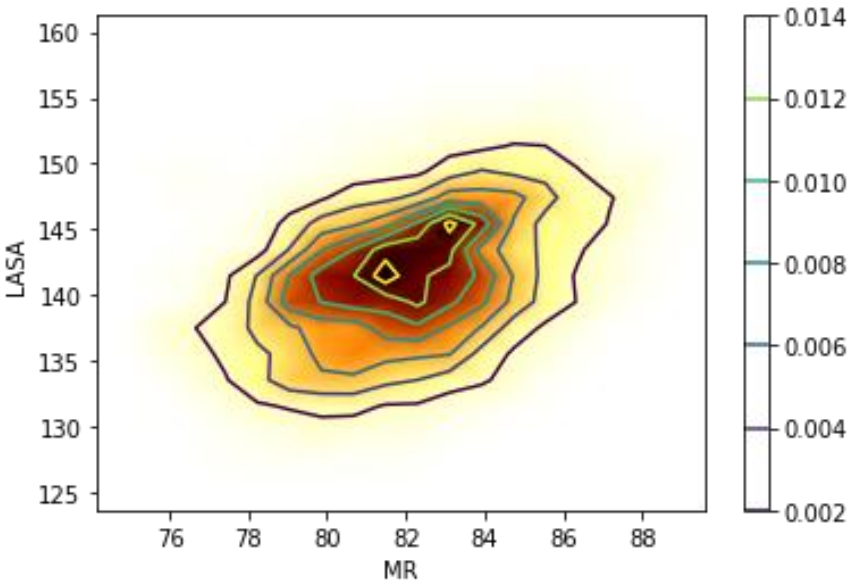


Figura 24. Distribución de propiedades de conjunto de output condicionado para entrenamiento con TPSA, MR y LASA: (a) TPSA vs MR; (b) TPSA vs LASA; (c) MR vs LASA

La TPSA se ha movido completamente hacia un valor de alrededor de 100. La MR muestra una distribución bimodal, con un máximo alrededor de 80 y otro alrededor de 84. Ambas distribuciones se han estrechado enormemente. Como consecuencia del condicionado de la TPSA y la MR, la LASA también ha estrechado su distribución, situándose prioritariamente entre 130 y 150, que es el rango de valores que en el conjunto de entrenamiento tiene esta propiedad para TPSA = 100 y MR = 80.

Como conclusión podemos decir que el SSVAE parece funcionar igualmente bien con otras propiedades, y que es capaz de generar moléculas condicionadas a dos o más propiedades.

2.6. Modificación del SSVAE para generar moléculas condicionadas a dos o más propiedades. Generación de análogos condicionados.

En su artículo, Kang y Cho condicionan la generación de moléculas del SSVAE a un peso molecular de 250 Da. En su código la red predictora es entrenada con tres propiedades. Sin embargo, la función de muestreo condicionado que implementan (**`sampling_conditional`**) solo admite una única propiedad condicionadora.

El diseño molecular es un proceso altamente multidimensional, en el que hay que optimizar simultáneamente distintas variables de distinta naturaleza, a veces con tendencias contrapuestas. Es por ello que es de especial interés un sistema de QSAR inverso multietiqueta, en el que se

pueda condicionar la generación de moléculas a más de dos propiedades.

Al objeto de utilizar el SSVAE condicionado a múltiples propiedades, se modificó la función de muestreo condicionado para poder condicionar por múltiples propiedades. La función resultante es **`mysampling_conditional`**, dentro del nuevo módulo **`mySSVAE.py`**. La modificación básicamente consistió en modificar una variable de entrada de selección de propiedad condicionante a una lista en la que se podían añadir múltiples propiedades. Ya hemos visto en la **Sección 2.5** que la modificación realizada resulta en una versión del SSVAE con capacidad de condicionamiento múltiple, en ese caso condicionando por TPSA y MR.

Una aplicación especialmente interesante de la generación condicionada por múltiples propiedades es la **generación de análogos condicionada**. Una de las propiedades a condicionar sería la similitud a una o más moléculas, y las otras propiedades serían las propiedades de condicionamiento de generación de análogos. Esta idea se probó con el SSVAE modificado.

Al objeto de generar el conjunto de entrenamiento se creó el notebook **`GenerateExp6.ipynb`**. En él, se importaron las 310K del conjunto de entrenamiento de Kang y Chao y se clusterizaron. El centroide del cluster más grande resultante (**Fig. 25**) se utilizó como "diana" de la generación de análogos.

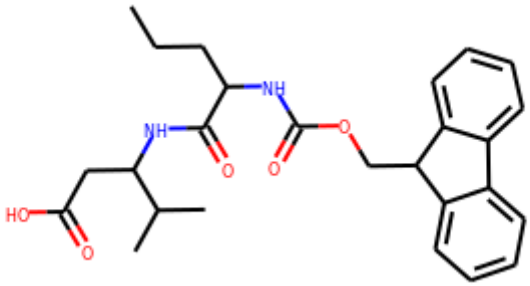
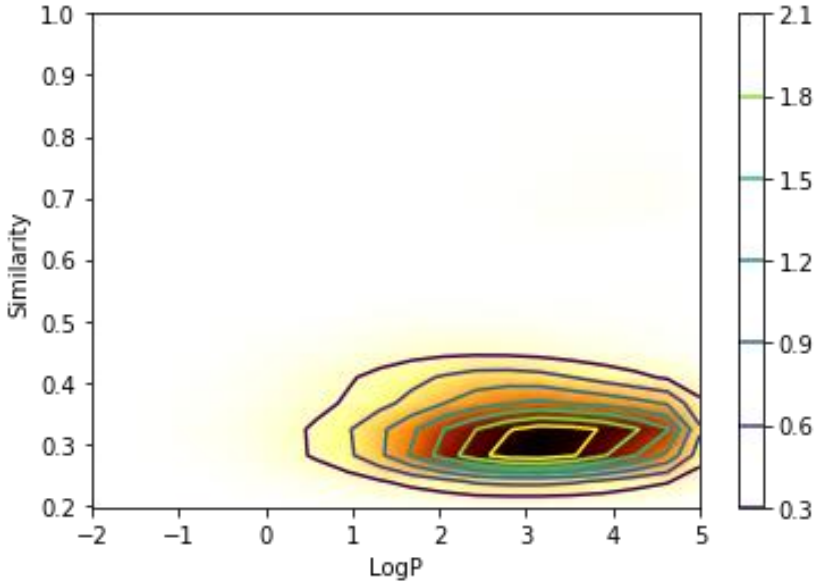


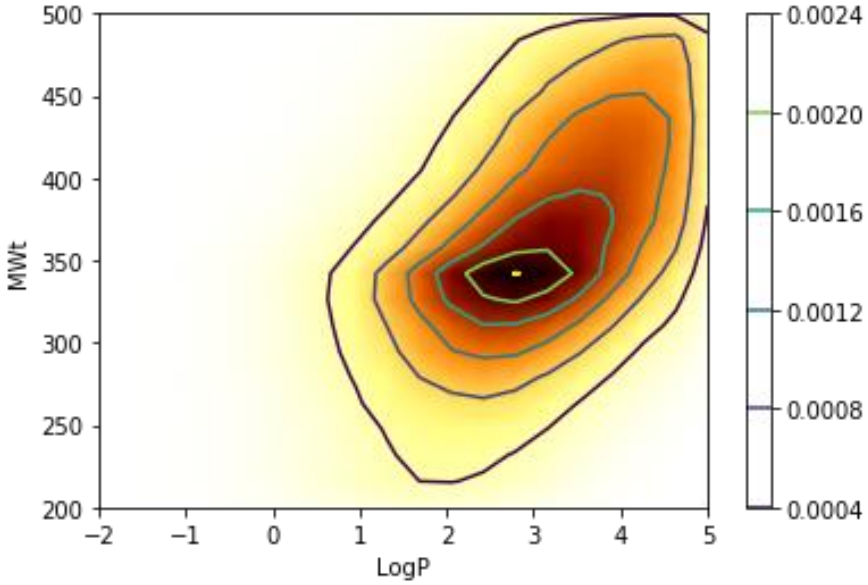
Figura 25. Centroide del primer cluster del conjunto de entrenamiento

Para ello se calculó la similitud de Tanimoto de todas las moléculas a esta molécula, lo que se utilizó como primera propiedad condicionante. También se calcularon el peso molecular y el logP de todas las moléculas, como las dos propiedades adicionales de entrenamiento del SSVAE. La **Fig. 26** muestra las distribuciones por pares de logP, similitud y peso molecular del conjunto de entrenamiento:

(a)


Similarity
LogP
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
−2
−1
0
1
2
3
4
5
2.1
1.8
1.5
1.2
0.9
0.6
0.3

(b)


MWt
LogP
500
450
400
350
300
250
200
−2
−1
0
1
2
3
4
5
0.0024
0.0020
0.0016
0.0012
0.0008
0.0004

(c)

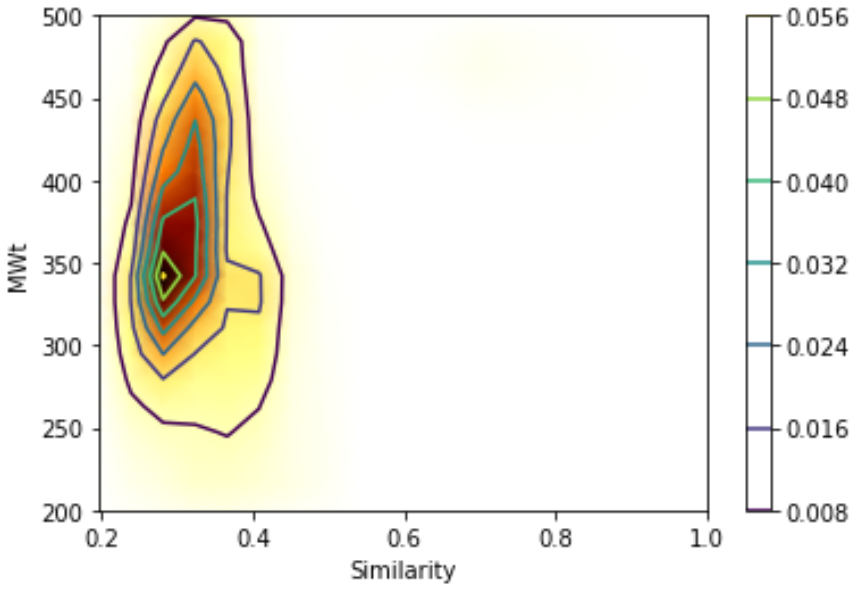


Figura 26. Distribución de propiedades de conjunto entrenamiento para generar análogos condicionados: (a) logP vs similitud; (b) logP vs peso molecular; (c) similitud vs peso molecular

Puede verse que globalmente que las similitudes son muy bajas, y se centran alrededor de 0.3, mientras que las otras propiedades muestran los valores típicos “drug-like” que ya vimos en la **Fig 4**.

La distribución de logP del primer cluster del conjunto de entrenamiento (el que contiene las moléculas más similares a la molécula diana, con una similitud > 0.8) se muestra en la **Fig. 27**:

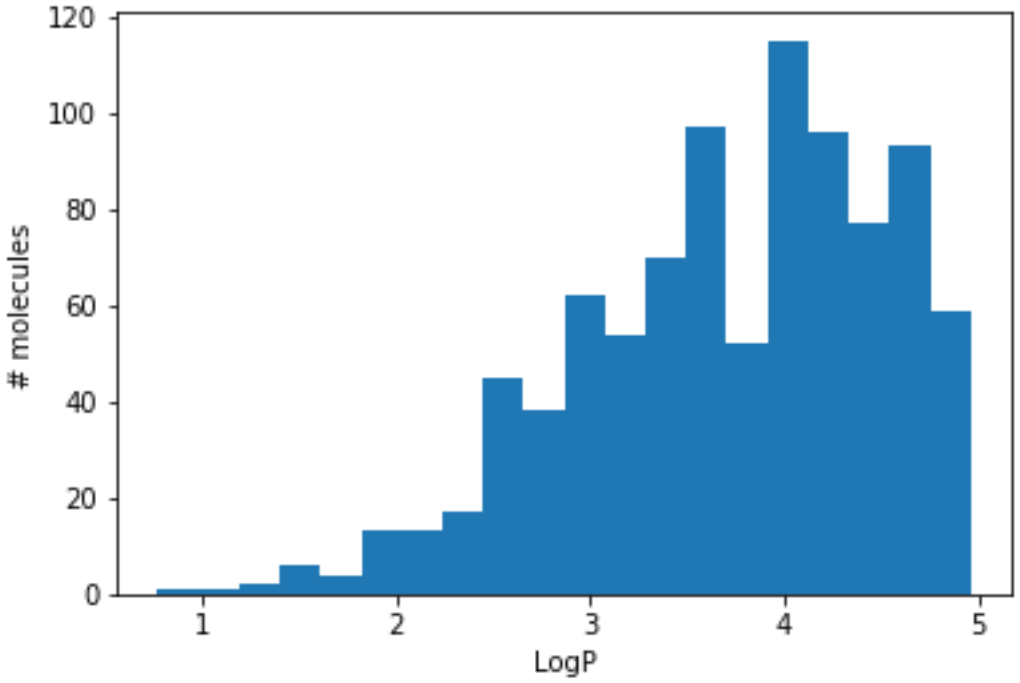


Figura 27. Distribución logP del primer cluster del conjunto de entrenamiento

La media de logP está centrada en 4 aproximadamente, y el mínimo logP del cluster es de 0.77. Se decidió entonces condicionar la generación de moléculas del SSVAE a una similitud de 1 y un logP de 0.5, al objeto de generar análogos con propiedades de lipofilicidad muy alejadas del resto de los análogos. Se generaron 10000 moléculas.

La **Tabla 9** muestra los parámetros de corrección, diversidad y novedad de la generación condicionada:

# mol input	%corr	# mol unicas	# clusters	# frames	% nuevas estr	% nuevos frames
10000	97.1	2693	1622	496	61.75	63.1

Tabla 9. Resultados de corrección, diversidad y novedad de la generación condicionada de análogos

Vemos que ha habido un gran número de duplicados (un 72%) y la diversidad es baja. Sin embargo, es remarcable el gran aumento en novedad, sin duda efecto de condicionar la generación a la similitud de la molécula diana.

La distribución de logP vs similitud del output se muestra en la **Fig. 28**:

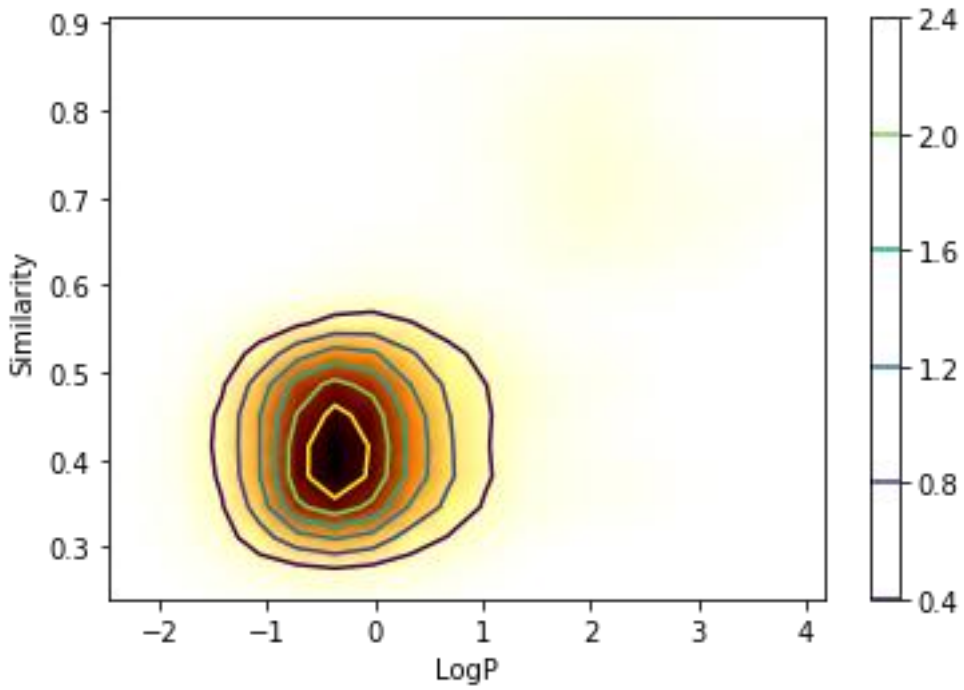


Figura 28. Distribución de propiedades de output de generación de análogos condicionados

Vemos que se el logP está centrado en 0.5 aproximadamente, como cabía esperar. La similitud global ha aumentado, estando centrada ahora alrededor de 0.4. Si miramos la distribución marginal de la similitud, veremos que ha aparecido una cola larga a la derecha, de altas similitudes (**Fig. 28**):

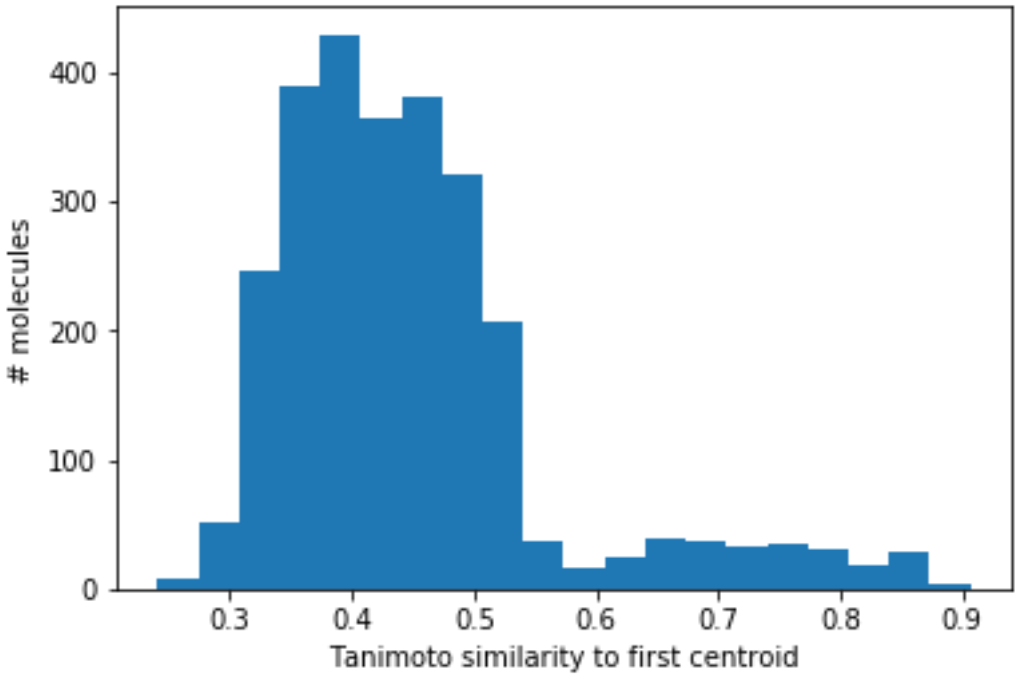


Figura 28. Distribución de similitudes a la molécula diana del output de generación de análogos condicionados. Véase la cola de análogos a la derecha.

Parece por tanto que ha aumentado la generación de análogos, a pesar de la muy pequeña cantidad de análogos que había presente en conjunto de entrenamiento (aproximadamente un 0.3%). Buscamos dentro de estos análogos moléculas con bajo logP, cercano a 0.5. La **Tabla 10** muestra las 6 moléculas del conjunto output con similitud > 0.8 y logP < 1:

smiles	id	logp	mwt	similarity
NC(=O)CNC(=O)C(CC(=O)O)NC(=O)OCC1c2ccccc2-c2ccccc21	s49	0.97	411.41	0.859
O=C(O)CC(NC(=O)OCC1c2ccccc2-c2ccccc21)C(=O)NCC(O)C(=O)O	s357	0.93	442.42	0.858
O=C(NC(CC(=O)O)C(=O)NCC(O)C(=O)O)OCC1c2ccccc2-c2ccccc21	s1170	0.93	442.42	0.858
O=C(NC(CC(=O)O)C(=O)NC(CO)C(=O)O)OCC1c2ccccc2-c2ccccc21	s1837	0.93	442.42	0.851
NC(=O)C(NC(=O)OCC1c2ccccc2-c2ccccc21)C(=O)NCC(O)C(=O)O	s2510	-0.06	427.41	0.8
NCC(O)CNC(=O)C(CC(=O)O)NC(=O)OCC1c2ccccc2-c2ccccc21	s2528	0.80	427.46	0.852

Tabla 10. Moléculas del output de generación de análogos condicionados con similitud > 0.8 y logP < 1

Vemos que de hecho una de las moléculas (s2510, **Fig. 29**) tiene un logP bastante inferior al mínimo logP (-0.06 vs 0.77):

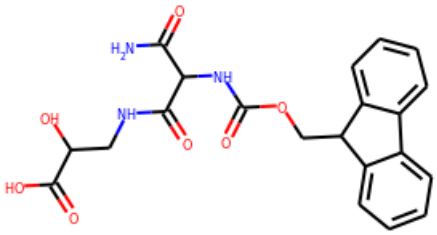


Figura 29. Estructura de s2510, análogo de molécula diana con logP = -0.06

En s2510 la cadena alquílica que estaba situada entre la amida y el carbamato de la molécula diana ha sido sustituida por una amida, resultando en una versión más polar de la molécula diana y por tanto con menor logP.

Concluimos por tanto que el SSVAE modificado para condicionar por varias propiedades permite generar análogos condicionados, lo cual resulta especialmente interesante para el diseño y optimización molecular multiobjetivo. Este método se presenta como una herramienta para optimizar hits (ver **Anexo I**) o leads para obtener candidatos a ensayos clínicos con propiedades optimizadas adecuadamente.

3. Conclusiones

Como conclusiones de este TFM, podemos enumerar las siguientes:

- El SSVAE genera moléculas con una alta diversidad química y cierta novedad. En el caso incondicionado, esta novedad es menor, pero la diversidad es mayor. En el caso condicionado, perdemos diversidad química, pero la novedad del output aumenta, en algunos casos enormemente. El hecho de que tenga capacidad de generar nuevo espacio químico, y lo que es más importante, este pueda ser condicionado a unos valores de propiedades, indica que tiene potencialidad de extrapolación guiada del mismo y por tanto lo valida como modelo efectivo de QSAR inverso.
- La corrección química del output del SSVAE depende del condicionamiento (es significativamente mayor con las moléculas condicionadas), pero no del tamaño del conjunto de entrenamiento ni del de output. En cualquier caso es muy alta, superior al 90%.
- La diversidad química depende significativamente del tamaño del conjunto de output (aumenta linealmente con el mismo), pero de forma diferente según el condicionamiento (se da una interacción significativa entre ellos): más rápido con moléculas incodicionadas que con las condicionadas. No parece depender del tamaño del conjunto de entrenamiento, al menos para los tamaños de output probados en este TFM. Es bastante posible que en el caso de generarse outputs del orden o más del conjunto de entrenamiento vieramos una dependencia de la diversidad con el tamaño del mismo, siempre y cuando se trabaje con un espacio latente suficientemente grande.
- La novedad química depende significativamente tanto del tamaño del conjunto de entrenamiento (disminuye con el mismo), como del tamaño del conjunto output (aumenta con el mismo) y del condicionamiento; respecto a este último, la intercepta es mayor con la generación condicionada para el porcentaje de moléculas nuevas, y menor para el porcentaje de frameworks nuevos, si bien en productos naturales en ambos casos la intercepta es mayor para la generación condicionada.
- La diversidad *intensiva* del conjunto de entrenamiento sí tiene una influencia decisiva en la diversidad y novedad del output. Si partimos, para un mismo tamaño de conjunto de entrenamiento, de moléculas muy diversas, para un mismo tamaño de conjunto de output obtendremos una diversidad y novedad química mayor que si partimos de un conjunto de entrenamiento muy redundante. En algunos casos (número de clusters y de frameworks, y así como del porcentaje de frameworks nuevos) también se da una interacción significativa entre la diversidad intensiva y el condicionamiento.

- Se ha modificado el SSVAE con éxito para generar productos naturales. La diversidad y novedad del output incondicionado es menor que con moléculas “drug-like”, lo cual es esperable dado la mayor complejidad de estas moléculas. La novedad aumenta al condicionar la generación de moléculas, lo cual es muy interesante ya que abre la puerta a diseñar moléculas con las ventajas de los productos naturales (e.g. su polaridad y riqueza de estereocentros) pero sin las desventajas de los mismos (alto peso molecular y complejidad).
- Se ha modificado el SSVAE para condicionar la generación de moléculas a más de una propiedad. También se ha probado con propiedades no consideradas por Kang y Cho (concretamente, TPSA, MR y LASA), obteniendo resultados similares en cuanto a corrección, diversidad y novedad, y se ha condicionando simultáneamente a un valor diana de TPSA y MR resultando en la generación de moléculas doble-condicionadas con éxito.
- Finalmente se ha utilizado este SSVAE modificado para múltiples propiedades al objeto de generar análogos condicionados; en particular, se ha generado un análogo del primer centroide del conjunto de entrenamiento de Kang y Chao con polaridad aumentada (logP menor que el menor de los logP del correspondiente cluster).

En definitiva, el SSVAE se presenta como una herramienta con alto potencial para hacer diseño molecular *in silico* mediante QSAR inverso. Mediante la modulación del conjunto de entrenamiento y de las propiedades condicionantes existe una gran variedad de posibles aplicaciones. En particular, las modificaciones implementadas aquí han permitido generar productos naturales y análogos condicionados. Un área de interés para futuras investigaciones es hacer pruebas de estrés del modelo en cuanto a su diversidad y novedad, vía generación de conjuntos output del orden de millones o decenas de millones de moléculas.

4. Glosario

ELU	Exponential Linear Unit
GAN	Generative Adversarial Network
GRU	Gated Recurrent Unit
LASA	Labute Approximate Surface Area
LSTM	Long Short-Term Memory
MR	Molar Refractivity
MWt	Molecular Weight
ReLU	Rectifier Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SMILES	Simplified Molecular Input Line Entry Specification
SSVAE	Semi-Supervised Variational Autoencoder
TPSA	Topological Polar Surface Area
VAE	Variational Autoencoder
WLN	Wiswesser Line Notation

5. Bibliografía

1. Kuhn, B., Guba, W., Hert, J., Banner, D., Bissantz, C., Ceccarelli, S., Haap, W., Körner, M., Kuglstatter, A., Lerner, C. and Mattei, P. A Real-World Perspective on Molecular Design: Miniperspective. *Journal of medicinal chemistry*, **2016**, *59*, 4087-4102.
2. *De novo molecular design*; Schneider, G., Ed. Wiley-VCH Verlag: Weinheim, Germany, 2003
3. Reymond, J.-L.; Ruddigkeit, L.; Blum, L.; van Deursen, R. The enumeration of chemical space. *Wiley Interdisc. Rev. Comp. Mol. Sci*. **2012**, *2*, 717−733.
4. Virshup, A.M., Contreras-García, J., Wipf, P., Yang, W. and Beratan, D.N. Stochastic voyages into uncharted chemical space produce a representative library of all possible drug-like compounds. *J. Am. Chem. Soc.* **2013**, *135*, 7296-7303.
5. Polishchuk, P. G.; Madzhidov, T. I.; Varnek, A. Estimation of the size of drug-like chemical space based on GDB-17 data. *J. Comput.-Aided Mol. Des*. **2013**, *27*, 675−679.
6. Macarron, R., Banks, M.N., Bojanic, D., Burns, D.J., Cirovic, D.A., Garyantes, T., Green, D.V., Hertzberg, R.P., Janzen, W.P., Paslay, J.W. and Schopfer, U. Impact of high-throughput screening in biomedical research. *Nat. Rev. Drug discovery*, **2011**, *10*, 188-195.
7. Cherkasov, A., Muratov, E.N., Fourches, D., Varnek, A., Baskin, I.I., Cronin, M., Dearden, J., Gramatica, P., Martin, Y.C., Todeschini, R. and Consonni, V. QSAR modeling: where have you been? Where are you going to? *J. Med. Chem.*, **2014**, *57*, 4977-5010.
8. Wong, W.W. and Burkowski, F.J. A constructive approach for discovering new drug leads: Using a kernel methodology for the inverse-QSAR problem. *J. Cheminf.*, **2009**, *1*, 4.
9. Miyao, T.; Arakawa, M.; Funatsu, K. Exhaustive structure generation for inverse-QSPR/QSAR. *Mol. Inf*. **2010**, *29*, 111−125.
10. Miyao, T.; Kaneko, H.; Funatsu, K. Inverse QSPR/QSAR Analysis for chemical structure generation (from y to x). *J. Chem. Inf. Model*. **2016**, *56*, 286−299.
11. Lusci, A., Pollastri, G. and Baldi, P. Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules. *J. Chem. Inf. Mod.*, **2013**, 53, 1563-1575.
12. Mayr, A., Klambauer, G., Unterthiner, T. and Hochreiter, S. DeepTox: toxicity prediction using deep learning. *Front. Environ. Sci.*, **2016**, *3*, 80.
13. Xu, Y., Dai, Z., Chen, F., Gao, S., Pei, J. and Lai, L. Deep learning for drug-induced liver injury. *J. Chem. Inf. Mod.*, **2015**, *55*, 2085-2093.
14. Zhang, L., Tan, J., Han, D. and Zhu, H. From machine learning to deep learning: progress in machine intelligence for rational drug discovery. *Drug Discovery Today*, **2017**, *22*, 1680-1685.
15. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; The MIT Press: Massachusetts, 2016.
16. LeCun, Y., Bengio, Y. Hinton, G. Deep learning. *Nature*, **2015**, *521*, 436-444.
17. Géron, A. *Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent Systems*; O'Reilly Media, Inc.: Boston, 2017.
18. Glorot, X. and Bengio, Y., 2010, March. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*; 2010; pp 249-256.
19. He, K., Zhang, X., Ren, S. and Sun, J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *Proceedings of the IEEE International conference on computer vision;* 2015*;* pp 1026-1034.

20. Hahnloser, R.; Sarpeshkar, R.; Mahowald, M. A.; Douglas, R. J.; Seung, H. S. Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*. **2000**,*405*, 947–951.
21. Maas, A.L.; Hannun, A.Y.; Ng, A.Y. Rectifier nonlinearities improve neural network acoustic models. *Proceedings of the 30th International Conference on Machine Learning.* **2013**, *30*, 3.
22. Clevert, D.A.; Unterthiner, T.; Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv.org* **2015**, No. arXiv:1511.07289.
23. Kingma, D.P.; Ba, J. Adam: A method for stochastic optimization. *arXiv.org* **2014**, No. arXiv:1412.6980.
24. Duchi, J., Hazan, E. and Singer, Y., 2011. Adaptive subgradient methods for online learning and stochastic optimization. *J. Machine Learning Res.*, **2011**, *12*, 2121-2159.
25. Tieleman, Tijmen and Hinton, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012
26. Ioffe, S.; Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv.org* **2015**, No. arXiv:1502.03167.
27. Hochreiter, S.; Schmidhuber, J. Long short-term memory. *Neural comp*, **1997**, *9*,1735-1780.
28. Zaremba, W.; Sutskever, I.; Vinyals, O. Recurrent neural network regularization. *arXiv.org* **2014**, No. arXiv:1409.2329.
29. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv.org* **2014**, No. arXiv:1406.1078.
30. Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. *Advances in Neural Information Processing Systems* **2014**, 2672−2680.
31. Kingma, D. P.; Welling, M. Auto-encoding Variational Bayes. *arXiv.org* **2014**, No. arXiv:1312.6114v10.
32. Sanchez-Lengeling, B.; Aspuru-Guzik, A. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*. **2018**, 361:360-365.
33. Weininger, D. SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules. *J. Chem. Inf. Model*. **1988**, *28*, 31−36.
34. Segler, M. H. S.; Kogej, T.; Tyrchan, C.; Waller, M. P. Generating Focused Molecule Libraries for Drug Discovery with Recurrent Neural Networks. *ACS Cent. Sci.* **2018**, *4*, 120-131.
35. Olivecrona, M.; Blaschke, T.; Engkvist, O.; Chen, H. Molecular De-Novo Design through Deep Reinforcement Learning. *J. Cheminf*. **2017**, *9*, 1-14.
36. Popova, M.; Isayev, O.; Tropsha, A. Deep Reinforcement Learning for De-Novo Drug Design. *arXiv.org* **2017**, No. arXiv:1711.10907.
37. Gómez-Bombarelli, R.; Wei, J. N.; Duvenaud, D.; Hernández-Lobato, J. M.; Sánchez-Lengeling, B.; Sheberla, D.; Aguilera- Iparraguirre, J.; Hirzel, T. D.; Adams, R. P.; Aspuru-Guzik, A. Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules. *ACS Cent. Sci*. **2018**, *4*, 268-276.
38. Kusner, M. J.; Paige, B.; Hernández-Lobato, J. M. Grammar Variational Autoencoder. *arXiv.org* **2017**, No. arXiv:1703.01925.
39. Dai, H.; Tian, Y.; Dai, B.; Skiena, S.; Song, L. Syntax-Directed Variational Autoencoder for Structured Data. *arXiv.org* **2018**, No. arXiv:1802.08786.
40. Guimaraes, G. L.; Sanchez-Lengeling, B.; Farias, P. L. C.; Aspuru-Guzik, A. Objective-Reinforced Generative Adversarial Network (ORGAN) for Sequence Generation Models. arXiv.org **2017**, No. arXiv:1705.10843.

41. Putin, E.; Asadulaev, A.; Vanhaelen, Q.; Ivanenkov, Y.; Aladinskaya, A.V.; Aliper, A.; Zhavoronkov, A. Adversarial Threshold Neural Computer for Molecular De Novo Design. *Mol. Pharmaceutics*, **2018**, *15*, 4386–4397.
42. Makhzani, A.; Shlens, J.; Jaitly, N.; Goodfellow, I.; Frey, B. Adversarial autoencoders. *arXiv* **2015**, No. arXiv:1511.05644.
43. Kadurin, A.; Nikolenko, S.; Khrabrov, K.; Aliper, A.; Zhavoronkov, A. druGAN: an advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico. *Mol. Pharmaceutics*, **2017**, *14*, 3098-104.
44. Simonovsky, M.; Komodakis, N. GraphVAE: Towards Generation of Small Graphs Using Variational Autoencoders. *arXiv.org* **2018**, No. arXiv:1802.03480.
45. Jin, W.; Barzilay, R.; Jaakkola, T. Junction Tree Variational Autoencoder for Molecular Graph Generation. *arXiv.org* **2018**, No. arXiv:1802.04364.
46. De Cao, N.; Kipf, T. MolGAN: An Implicit Generative Model for Small Molecular Graphs. *arXiv.org* **2018**, No. arXiv:1805.11973.
47. Samanta, B.; De, A.; Ganguly, N.; Gomez-Rodriguez, M. Designing Random Graph Models Using Variational Autoencoders with Applications to Chemical Design. *arXiv.org* **2018**, No. arXiv:1802.05283.
48. Kang, S.; Cho, K. Conditional Molecular Design with Deep Generative Models. *J. Chem. Inf. Mod.* **2018** (Article ASAP) **DOI:** 10.1021/acs.jcim.8b00263
49. Kingma, D. P.; Mohamed, S.; Rezende, D. J.; Welling, M. Semi-Supervised Learning with Deep Generative Models. *Proceedings of the 28th Annual Conference on Neural Information Processing Systems*; **2014**; pp 3581−3589.
50. Irwin, J. J.; Shoichet, B. K. ZINC-a free database of commercially available compounds for virtual screening. *J. Chem. Inf. Comput. Sci.* **2005**, *45*, 177-182.
51. Sterling, T.; Irwin, J. J. Zinc 15-Ligand Discovery for Everyone. *J. Chem. Inf. Model*. **2015**, *55*, 2324−2337.
52. Bickerton, G. R.; Paolini, G. V.; Besnard, J.; Muresan, S.; Hopkins, A. L. Quantifying the Chemical Beauty of Drugs. *Nat. Chem.* **2012**, *4*, 90−98.
53. Wildman, S. A.; Crippen, G. M. Prediction of Physicochemical Parameters by Atomic Contributions. *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 868−873.
54. Bickerton, G. R.; Paolini, G. V.; Besnard, J.; Muresan, S.; Hopkins, A. L. Quantifying the Chemical Beauty of Drugs. *Nat. Chem.* **2012**, *4*, 90−98.
55. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; Zheng, X. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation*; 2016; pp 265−283.
56. Ertl, P.; Rohde, B.; Selzer, P. Fast calculation of molecular polar surface area as a sum of fragment-based contributions and its application to the prediction of drug transport properties. *J. Med. Chem*. **2000**, *43*, 3714-7.
57. Labute, P. A widely applicable set of descriptors. *J. Mol. Graph. Model.* **2000**, *18*, 464-477.
58. Dalke A. The FPS fingerprint format and chemfp toolkit. *J. Cheminf.* **2013**, *5*, 36.
59. Landrum, G. RDKit: Open-Source Cheminformatics. http://www.rdkit.org (accessed September 1st, 2018).
60. TIBCO Spotfire, version 7.6; TIBCO Software Inc.: Boston, 2016.
61. Bemis, G.W.; Murcko, M.A. The properties of known drugs. 1. Molecular frameworks. *J. Med. Chem.*, **1996**, *39*, 2887-2893.

62. Taylor, R. Simulation analysis of experimental design strategies for screening random compounds as potential new drugs and agrochemicals. *J. Chem. Inf. Comput. Sci.*, **1995**, *35*, 59-67.
63. Butina, D. Unsupervised data base clustering based on daylight's fingerprint and Tanimoto similarity: a fast and automated way to cluster small and large data sets. *J. Chem. Inf. Comput. Sci.* **1999**, *39*, 747-50.
64. Maggiora, G.M. Introduction to molecular similarity and chemical space. *Foodinformatics: Applications of Chemical Information to Food Chemistry*. Martínez-Mayorga, K.; Medina-Franco, J.L., Eds. Springer: New York, 2014; pp 1-82.
65. Koch, M.; Schuffenhauer, A.; Scheck, M.; Wetzel, S.; Casaulta, M.; Odermatt, A.; Ertl, P.; Waldmann, H. Charting biologically relevant chemical space: a structural classification of natural products (SCONP). *Proc. Natl. Acad. Sci. U.S.A.* **2005**, *102*, 17272-17277.
66. Ertl, P.; Roggo, S.; Schuffenhauer, A. Natural product-likeness score and its application for prioritization of compound libraries. *J. Chem. Inf. Mod.*, **2008**, *48*, 68-74.
67. Leach, A.R.; Gillet, V.J. *An Introduction to Chemoinformatics*; Springer: Dordrecht, The Netherlands, 2007.
68. *Chemoinformatics for Drug Discovery*; Bajorath, J., Ed.; Wiley: New Jersey, 2013.
69. Wiswesser, W. J. *A Line-Formula Chemical Notation*; Crowell: New York, 1954.
70. Johnson, M.A.; Maggiora, G.M. *Concepts and Applications of Molecular Similarity*; Wiley-Interscience: New Jersey, 1990.
71. Lipinski, C.A.; Lombardo, F.; Dominy, B.W.; Feeney, P.J., 1997. Experimental and computational approaches to estimate solubility and permeability in drug discovery and development settings. *Adv. Drug Deliv. Rev*, **1997**, *23*, 3-25.
72. Hann, M. M. Molecular Obesity, Potency and Other Addictions in Drug Discovery. *MedChemComm* **2011**, *2*, 349−355.
73. Teague, S.J.; Davis, A.M.; Leeson, P.D.; Oprea, T. The design of leadlike combinatorial libraries. *Angewandte Chem. Int. Ed.*, **1999**, *38*, 3743-3748.

6. Anexos

Anexo I. Bases de Quimioinformática

La Quimioinformática es la disciplina que estudia la aplicación de métodos computacionales para tratar problemas químicos, con especial énfasis en la manipulación y procesado de información química. Excelentes textos de Quimioinformática pueden encontrarse en Leach & Gillet, 2007[68] y Bajorath, 2013.[69]

Representaciones de la estructura molecular.

Una gran parte de la Quimioinformática se enfoca en el desarrollo de representaciones de la estructura molecular para almacenar información química en bases de datos, y también para utilizarla en modelos *in silico* de distintos tipos. Podemos representar o codificar diferentes aspectos de la estructura molecular, resultando en distintas representaciones: por ejemplo, la posición de todos sus átomos en 3 dimensiones. En este caso, podemos generar una matriz de coordenadas cartesianas junto con una columna que etiquete el elemento y número de cada átomo. Ejemplo de este tipo de representación son el formato de ficheros XYZ para estructuras moleculares 3D, y con algunas columnas más de conectividad y otras informaciones, los formatos SDF y MDL.
Una representación alternativa consiste fijarse en la topología de la molécula y entonces asimilarla a un grafo no dirigido con vértices y aristas: los vértices serían los átomos, y las aristas los enlaces que conectan los átomos. Tanto los vértices como las aristas serían de distintos tipos: en el caso de los átomos, corresponderían a elementos con distintos estados de hibridación (e.g. carbono con hibridación sp^3, sp^2 y sp), y en el caso de los enlaces, a distintos tipos de enlace (e.g. sencillo, doble, triple, aromático).
Esta última es la representación que usualmente utiliza un químico cuando trabaja con moléculas y reacciones químicas, en ausencia de información tridimensional. Es por ello que desde hace bastante tiempo la Quimioinformática se ha preocupado de derivar codificaciones compactas de dichos grafos. Entre ellas, destacan las *notaciones de línea*, que consisten en códigos en forma de una línea de caracteres que mediante una serie de reglas permiten reconstruir reversiblemente el grafo molecular. Por ejemplo, ya en 1954 fue derivada la notación de línea Wiswesser (Wiswesser Line Notation o WLN),[69] que utilizaba una muy compleja combinación de reglas para su generación y decodificación.
Posteriormente apareció la notación SMILES[33] (de Simplified Molecular Input Line Entry Specification). Esta notación ha devenido en el estándar en Quimioinformática para la mayoría de las aplicaciones en las que se trabaja con grafos moleculares, dada su simplicidad y elegancia, y se implementa en forma de una secuencia de caracteres que corresponden a átomos, dependiendo el carácter de un átomo de su elemento y estado

de enlace, junto con otros caracteres que denotan ramificaciones y apertura y clausura de anillos. Por ejemplo, el código SMILES de la siguiente molécula:

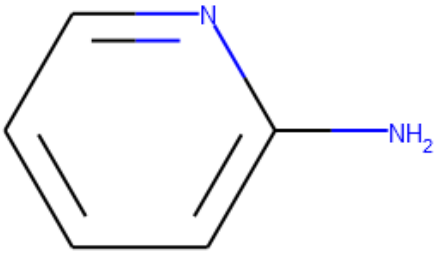


Figura 30. **Grafo molecular de una molécula ejemplo**

es c1ccnc(N)c1: los átomos son representados por su símbolo elemental, en minúscula si son aromáticos, y en mayúscula si no. Las ramificaciones se incluyen dentro de paréntesis, y el inicio y final de un anillo por el mismo número entero tras el símbolo de los átomos correspondientes.

Similitud molecular

El concepto de similitud molecular es de gran importancia, dado que moléculas similares tienden a tener similares propiedades y/o actividades,[70] permitiendo la existencia de modelos de QSAR que funcionen. Existen muchas medidas posibles de similitud molecular, dependiendo de la representación molecular que se utilice y de cómo se contabilicen las distintas características comunes y no comunes de las dos moléculas. En general, esperaríamos para una medida razonable de similitud un valor entre 0 (total disimilitud) y 1 (total identidad). Si nos ceñimos a grafos moleculares como los que hemos visto más arriba, podríamos calcular el máximo subgrafo común entre las dos moléculas utilizando teoría de grafos, seguido de la aplicación de una similitud de tipo Tanimoto (también conocido como coeficiente de Jaccard):

$$S_{AB} = \frac{c}{a + b - c}$$

donde *c* sería el número de átomos del máximo subgrafo común, *a* el número de átomos de A y *b* el número de átomos de B.

Este tipo de similitud, aunque muy fundamentada teóricamente, tiene el inconveniente de que es muy lenta de calcular, pues requiere de teoría de grafos. Un método aproximado, con resultados bastante satisfactorios, y muy rápido de calcular, se basa en la utilización de *fingerprints* moleculares, es decir, codificaciones del grafo molecular en forma de vector de 0s y 1s.

Existen múltiples tipos de fingerprint, pero los que utilizaremos en este TFM son los de tipo de “camino molecular” (molecular path), inicialmente

presentes en la toolkit quimioinformática Daylight, y posteriormente implementada en muchas otras toolkits como RDKit, utilizada en ese TFM. Básicamente consiste en determinar todos los caminos moleculares únicos (secuencias de átomos conectados por enlaces) presentes en una molécula desde un átomo hasta un número máximo de átomos, normalmente 7. Cada uno de ellos sirve como semilla para generar un número pseudo-aleatorio cuyo output es un conjunto de bits (normalmente 4 ó 5). Cada uno de estos conjuntos de bits se adicionan (OR lógico) al fingerprint. De este modo, moléculas similares tendrán caminos moleculares similares, que a su vez resultarán en conjuntos de 1s en posiciones similares en los correspondientes fingerprints. Para el cálculo de *c* en la similitud de Tanimoto simplemente hay que determinar las posiciones de los fingerprints en las que los 1s coinciden, y *a* y *b* serán el número de 1s de A y B, respectivamente, lo cual es muchísimo más rápido que calcular el subgrafo común máximo de dos moléculas.

La utilización de fingerprints se ha extendido y perfeccionado, y actualmente existen toolkits como chemfp que utilizan el formato FPS de ficheros de fingerprints,[58] así como una serie de opciones computacionales que permiten hacer cálculos de similitudes de grandes colecciones de moléculas en tiempos muy cortos. En este TFM hemos utilizado chemfp para el cálculo de similitudes de Tanimoto (por ejemplo, para hacer los cálculos de novedad química y para el clustering de moléculas, ver abajo).

Clustering molecular

El clustering molecular nos permite encontrar patrones estructurales en colecciones de moléculas y resumir la información química presente en las mismas. El clustering molecular está altamente relacionado con la similitud molecular, ya que los algoritmos de clustering se basan en la utilización de una similitud (o distancia, habitualmente 1 - similitud) entre las instancias que se desea clusterizar.

De acuerdo a lo comentado en el apartado de *Similitud Molecular*, los algoritmos de clustering de grafos moleculares (los más habituales) utilizan, por mayor rapidez y ahorro de memoria, matrices de similitud obtenidas mediante fingerprints y distancias de Tanimoto. Uno de los más utilizados y rápidos es el denominado de Taylor-Butina[62,63] o de "esferas duras" (esquematizado en la **Fig. 31**).

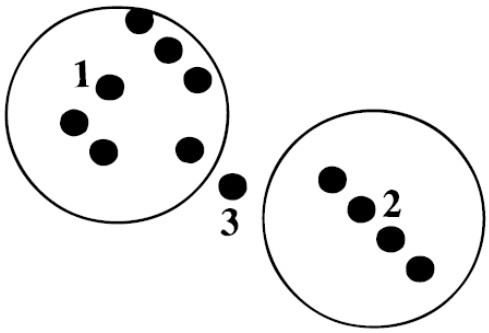


Figura 31. Esquema del funcionamiento del algoritmo de Taylor-Butina. Tomado de (63).

En este algoritmo, se define una similitud umbral *T*, normalmente 0.8 ó 0.85, por encima de la cual dos moléculas se consideran similares. Para cada molécula se determina el número de moléculas con similitud igual o mayor a *T.* El primer cluster se forma con la molécula con mayor número de moléculas similares o vecinas, que será el centroide del cluster, más dichas vecinas. Las moléculas de ese cluster se dejan de considerar y se repite la operación con el resto de la colección, y así sucesivamente hasta que lo único que quede sea moléculas sin vecinas ("singletones"). En la **Fig. 31**, la molécula 1 tiene el mayor número de vecinas (6), por lo que se queda como centroide del cluster que forma ella con las otras vecinas, que será el primer cluster. A continuación, la molécula 2 tiene el mayor número de vecinas, por lo que forma el segundo cluster. La molécula 3 no tiene vecinas, y formará el tercer cluster (un singleton).

Este algoritmo tiene como principal ventaja su rapidez, ya que solo requiere el cálculo de la matriz de similitud (cuello de botella de la mayoría de los algoritmos de clustering) una vez, y no necesita almacenar las similitudes, sino una simple tabla de vecinos. También tiene como ventaja que es determinista, ya que la creación de un nuevo cluster vendrá determinada por el centroide con mayor número de vecinos, y en caso de empate por el índice del fingerprint (ver código de la función **`taylor_butina_cluster`** en el **Anexo II**), resultando un proceso totalmente reproducible. Como inconveniente está el que no obtenemos una jerarquía o dendrograma de relaciones a múltiples similitudes, y que tiene tendencia a generar "singletones falsos", es decir, moléculas que se quedan fuera de clusters a pesar de que son muy similares a otras moléculas, simplemente por el hecho de que su vecina o vecinas han sido asignadas a otros clusters previamente (en la **Fig. 31** sería la molécula 3).

Frameworks moleculares

Bemis & Murcko,[61] al analizar la base de datos del Comprehensive Medicinal Chemistry, idearon un esquema de resumen de las estructuras moleculares en cuatro unidades:

1. Sistemas de anillos: ciclos y ciclos que comparten una arista (un enlace). Por ejemplo, el benceno, el antraceno y el naftaleno todos son un sistema de anillos sencillo.
2. Atomos linker: átomos que forman parte de un camino que conecta a dos sistemas de anillos.
3. Atomos de cadena lateral: átomos que no forman parte ni de un sistema de anillos ni de un linker.
4. Frameworks: conjunto de sistemas de anillos y linkers de una molécula, sin cadenas laterales.

De esta forma pudieron encontrar regularidades en la base de datos analizada, ya que por ejemplo la mitad de la misma estaba representada por únicamente los 32 frameworks más frecuentes.

Los frameworks así definidos son ampliamente utilizados en el análisis y resumen de conjuntos de estructuras químicas, y nos dan una visión alternativa de la diversidad química de una colección de moléculas y la novedad de la misma respecto de otra. Están implementados en multitud de toolkits de Quimioinformática, y nosotros los obtendremos utilizando la función **`GetScaffoldForMol`** de RDKit.

Moléculas "drug-like" vs "lead-like"

El descubrimiento de fármacos sigue una serie de etapas, cada una caracterizada por compuestos con distintas propiedades. En primer lugar, cuando se obtiene un compuesto activo en una determinada diana o ensayo biológico se habla de un *hit*. Este hit se optimiza inicialmente en cuanto a sus propiedades de potencia, especificidad, toxicidad, absorción y metabolismo en ensayos *in vitro*, resultando en un *lead*. A continuación se prueba en ensayos in vivo con animales, midiendo su farmacocinética, toxicidad, y si hay ensayo disponible, su eficacia; sucesivas modificaciones para optimizar estas propiedades en un *candidato* para ser probado en humanos en ensayos clínicos. Finalmente, si los resultados son positivos tendríamos una "droga".

Lipinski et al.[71] realizaron un análisis de las drogas en el mercado y propusieron la llamada "regla de cinco" para definir un compuesto como "drug-like":

- Peso molecular entre 150 y 500 Da
- logP <= 5
- Número de enlaces rotables <= 7
- PSA < 150
- Número de donores de enlaces de hidrógeno <= 5
- Número de aceptores de enlaces de hidrógeno <= 10

Inicialmente las colecciones de screening contenían este tipo de compuestos mayoritariamente. Posteriormente se observó que a medida que se progresa en el proceso de optimización química, se tendía a aumentar el tamaño y el logP de las moléculas, por lo que se vio que era mejor partir en las colecciones de screening de moléculas "lead-like". Teague et al definieron, en referencia a la regla de cinco, una regla similar para compuestos "lead-like":

- Peso molecular entre 250 y 350
- logP <= 3.5
- Número de enlaces rotables <= 7

Anexo II. Módulo myfuncs.py

```
import chemfp
from chemfp import search
import pandas as pd
import time
import sys
import subprocess as sp
from rdkit import Chem
import numpy as np
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from scipy.stats import kde
import scipy.spatial.distance as ssd
import matplotlib.pyplot as plt
import pylab
import random
from rdkit.Chem import Draw
from rdkit.Chem.Scaffolds import MurckoScaffold as ms
from rdkit.Chem.Draw import IPythonConsole
from rdkit.Chem import PandasTools as pt
from rdkit.Chem import Descriptors
from mpl_toolkits.axes_grid.inset_locator import inset_axes


### The results of the Taylor-Butina clustering (A. Dalke)
class ClusterResults(object):
    def __init__(self, true_singletons, false_singletons, clusters):
        self.true_singletons = true_singletons
        self.false_singletons = false_singletons
        self.clusters = clusters



### The clustering implementation (taken from A. Dalke)
def taylor_butina_cluster(similarity_table):
    # Sort the results so that fingerprints with more hits come
    # first. This is more likely to be a cluster centroid. Break ties
    # arbitrarily by the fingerprint id; since fingerprints are
    # ordered by the number of bits this likely makes larger
    # structures appear first.:

    # Reorder so the centroid with the most hits comes first.  (That's why I d
o
    # a reverse search.)  Ignore the arbitrariness of breaking ties by
    # fingerprint index

    centroid_table = sorted(((len(indices), i, indices)
                                    for (i,indices) in enumerate(similarity_table
.iter_indices())),
                               reverse=True)

    # Apply the leader algorithm to determine the cluster centroids
    # and the singletons:

    # Determine the true/false singletons and the clusters
    true_singletons = []
    false_singletons = []
    clusters = []

    seen = set()
    for (size, fp_idx, members) in centroid_table:
        if fp_idx in seen:
            # Can't use a centroid which is already assigned
            continue
        seen.add(fp_idx)
```

```
62.
63.         # Figure out which ones haven't yet been assigned
64.         unassigned = set(members) - seen
65.
66.         if not unassigned:
67.             false_singletons.append(fp_idx)
68.             continue
69.
70.         # this is a new cluster
71.         clusters.append((fp_idx, unassigned))
72.         seen.update(unassigned)
73.
74.     # Return the results:
75.     return ClusterResults(true_singletons, false_singletons, clusters)
76.
77.
78.
79. ### Calculate distance matrix for hierarchical clustering (A. Dalke)
80. def distance_matrix(arena):
81.     n = len(arena)
82.
83.     # Start off a similarity matrix with 1.0s along the diagonal
84.     similarities = np.identity(n, "d")
85.
86.     ## Compute the full similarity matrix.
87.     # The implementation computes the upper-triangle then copies
88.     # the upper-triangle into lower-triangle. It does not include
89.     # terms for the diagonal.
90.     results = search.threshold_tanimoto_search_symmetric(arena, threshold=0.0)

91.
92.     # Copy the results into the NumPy array.
93.     for row_index, row in enumerate(results.iter_indices_and_scores()):
94.         for target_index, target_score in row:
95.             similarities[row_index, target_index] = target_score
96.
97.     # Return the distance matrix using the similarity matrix
98.     return 1.0 - similarities
99.
100.
101.
102.### Create a smis list from a smiles file (just smiles)
103.def smif2smis(name):
104.    smidf = pd.read_csv(name, delim_whitespace = True, names = ['smiles'], hea
    der = None)
105.    return list(smidf['smiles'])
106.
107.
108.
109.### Find the correct smiles in a list of smiles
110.def corrsmis(smis):
111.    n = len(smis)
112.    corr_smi_yn = [x != None for x in [Chem.MolFromSmiles(s) for s in smis]]
113.    ncorr = sum(corr_smi_yn)
114.    smis = [smi for i, smi in enumerate(smis) if corr_smi_yn[i] == True]
115.    wrongsmis = [smi for i, smi in enumerate(smis) if corr_smi_yn[i] == False]

116.    return ncorr, n, smis, wrongsmis
117.
118.
119.
120.### Create a dataframe of smiles, id from smiles list
121.def smis2smidf(smis):
122.    return pd.DataFrame({'smiles': smis, 'id': ['s' + str(x) for x in range(1,
     len(smis)+1)]}, columns = ['smiles','id'])
123.
```

```
124.
125.
126.### Create a dataframe of smiles, id from smiles file
127.def smisf2smidf(smisf, noid = True, random = False, seed = 1234):
128.
129.    if noid:
130.        smidf = pd.read_csv(smisf, delim_whitespace = True, names = ['smiles']
    , header = None)
131.    else:
132.        smidf = pd.read_csv(smisf, delim_whitespace = True, names = ['smiles',
    'id'], header = None)
133.
134.    if random == True:
135.    smidf = smidf.sample(frac=1, random_state = seed)
136.    return smidf
137.
138.
139.
140.### Create arena from smiles df
141.def smidf2arena(smidf, reorder = True):
142.
143.    # Write df of smiles, id
144.    smidf.to_csv('smidf.smi', header = False, sep = ' ', index = False)
145.
146.    # Generate fps file
147.    sp.call(['rdkit2fps', './smidf.smi', '-o', 'smidf.fps'])
148.
149.    ## Load the FPs into an arena
150.    try:
151.        arena = chemfp.load_fingerprints('./smidf.fps', reorder = reorder)
152.    except IOError as err:
153.        sys.stderr.write("Cannot open fingerprint file: %s" % (err,))
154.        raise SystemExit(2)
155.
156.    # Remove files
157.    sp.call(['rm', './smidf.smi', './smidf.fps'])
158.
159.    # Return arena
160.    return arena
161.
162.
163.
164.### Create an fps file from a smiles df
165.def smidf2fps(smidf, name):
166.    # Write df of smiles, id
167.    smidf.to_csv('smidf.smi', header = False, sep = ' ', index = False)
168.
169.    # Generate fps file
170.    sp.call(['rdkit2fps', './smidf.smi', '-o', name + ".fps"])
171.
172.    # Remove files
173.    sp.call(['rm', './smidf.smi'])
174.
175.
176.
177.### Remove fps
178.def remfps(name):
179.    sp.call(['rm', './' + name + '.fps'])
180.
181.
182.
183.### Cluster from smiles df
184.def clusmidf(smidf, th = 0.8, method = 'butina', arena = None):
185.
186.    if method != 'butina' and method != 'cl':
187.        print('Please select butina or cl')
```

```
188.         return None
189.
190.     # Init time counter
191.     start = time.time()
192.
193.     # Get the arena
194.     if arena is None:
195.         arena = smidf2arena(smidf)
196.
197.     # Do the clustering
198.     if method == 'butina':
199.         # Generate the similarity table
200.         similarity_table = search.threshold_tanimoto_search_symmetric(arena, t
    hreshold = th)
201.
202.         # Cluster the data
203.         clus_res = taylor_butina_cluster(similarity_table)
204.
205.         # Output
206.         out = []
207.         # We need to re-
    sort the clusters as the creation of them does not generate a monotonously decr
    easing list
208.         cs_sorted = sorted([(len(c[1]), c[1], c[0]) for c in clus_res.clusters
    ], reverse = True)
209.         for i in range(len(cs_sorted)):
210.             cl = []
211.             c = cs_sorted[i]
212.             cl.append(arena.ids[c[2]]) # Retrieve the arenaid of the centroid
    and add to the cluster
213.             cl.extend([arena.ids[x] for x in c[1]]) # Retrieve the arenaid of
    the neighbors and add to cluster
214.             out.append(cl)
215.         for i in range(len(clus_res.false_singletons)):
216.             cl = [arena.ids[clus_res.false_singletons[i]]]
217.             out.append(cl)
218.         for i in range(len(clus_res.true_singletons)):
219.             cl = [arena.ids[clus_res.true_singletons[i]]]
220.             out.append(cl)
221.
222.     elif method == 'cl':
223.         # Generate the condensed distance table
224.         distances  = ssd.squareform(distance_matrix(arena))
225.
226.         # Cluster the data
227.         clus_res = fcluster(linkage(distances, method='complete'), th, 'distan
    ce')
228.
229.         # Ouptut
230.         aids = arena.ids
231.         out = []
232.         for i in np.unique(clus_res):
233.             cl = [aids[i] for i in list(np.where(clus_res == i)[0])]
234.             out.append(cl)
235.         out = [x[2] for x in sorted([(len(x), i, x) for (i, x) in enumerate(ou
    t)], reverse = True)]
236.
237.
238.     # End time count and report
239.     end = time.time()
240.     elapsed_time = end - start
241.     print('Clustering time: ' + time.strftime("%H:%M:%S", time.gmtime(elapsed_
    time)))
242.
243.     # Return cluster results
244.     return out
```

```
245.
246.
247.
248.### Draw a set of molecules from smiles list
249.def paintmols(smis, molsPerRow = 5, subImgSize=(150,150)):
250.    ms = [Chem.MolFromSmiles(s) for s in smis]
251.    return Draw.MolsToGridImage(ms,molsPerRow=molsPerRow,subImgSize=subImgSize
    )
252.
253.
254.
255.### Generate framework for a SMILES, handling for errors
256.def framecheck(s):
257.    try:
258.        return Chem.MolToSmiles(ms.GetScaffoldForMol(Chem.MolFromSmiles(s)))
259.    except:
260.        pass
261.
262.
263.
264.### Generate generic framework for a SMILES, handling for errors
265.def gframecheck(s):
266.    try:
267.        return Chem.MolToSmiles(ms.MakeScaffoldGeneric(Chem.MolFromSmiles(s)))

268.    except:
269.        pass
270.
271.
272.
273.### Diversity analysis
274.def divan(smidf, summ = False, OnlyBu = False, arena = None):
275.
276.    start = time.time()
277.
278.    # Cluster by butina and cl
279.    clr_bu = clusmidf(smidf, arena = arena)
280.    if(not OnlyBu):
281.        clr_cl = clusmidf(smidf, method = 'cl', th = 0.55, arena = arena)
282.
283.    # Count the number of cluster in each method
284.    ncl_bu = len(clr_bu)
285.    if(not OnlyBu):
286.        ncl_cl = len(clr_cl)
287.
288.    # Count Murko frameworks
289.    fras = list(set([framecheck(s) for s in smidf.smiles]))
290.    nfra = len(fras)
291.
292.    # Count generic Murko frameworks
293.    frasg = list(set([gframecheck(s) for s in smidf.smiles]))
294.    nfrag = len(frasg)
295.
296.    end = time.time()
297.    eltime = end - start
298.    print('Diversity analysis time: ' + time.strftime("%H:%M:%S", time.gmtime(
    eltime)))
299.
300.    if(summ):
301.        if(OnlyBu):
302.            return ncl_bu, nfra, nfrag
303.    else:
304.            return ncl_bu, ncl_cl, nfra, nfrag
305.    else:
306.        if(OnlyBu):
307.            return clr_bu, fras, frasg
```

```
    else:
        return clr_bu, clr_cl, fras, frasg


### Novelty analysis
def novan(smidfq, smidft, th = 0.7, arq = None, art = None):

    start = time.time()

    # Get the arenas
    if arq is None:
        arq = smidf2arena(smidfq)
    if arq is None:
        art = smidf2arena(smidft)

    end = time.time()
    eltime = end - start
    print('Arenas creation time: ' + time.strftime("%H:%M:%S", time.gmtime(elt
ime)))

    # Find hits
    results = chemfp.search.threshold_tanimoto_search_arena(arq, art, threshol
d=th)

    # Generate list with new guys (no neighbors in target arena) and calculate
 its length
    news = []
    for query_id, query_hits in zip(arq.ids, results):
        if len(query_hits) == 0:
            news.append(query_id)


    # Generate list of frameworks for query and target
    fraq = [framecheck(s) for s in smidfq.smiles]
    fraq = list(np.unique(fraq))
    frat = [framecheck(s) for s in smidft.smiles]
    frat = list(np.unique(frat))

    newfraqs = [f for f in fraq if f not in frat]

    # Generate list of generic frameworks for query and target
    gfraq = [gframecheck(s) for s in smidfq.smiles]
    gfraq = list(np.unique(gfraq))
    gfrat = [gframecheck(s) for s in smidft.smiles]
    gfrat = list(np.unique(gfrat))

    newgfraqs = [f for f in gfraq if f not in gfrat]

    end = time.time()
    eltime = end - start
    print('Novelty analysis time: ' + time.strftime("%H:%M:%S", time.gmtime(el
time)))

    return news, fraq, newfraqs, gfraq, newgfraqs


### Plot clusters
def plotclus(d, xlab, ylab, xloglab, yloglab):

    ax1 = plt.axes()  # standard axes
    ax2 = plt.axes([0.45, 0.45, 0.4, 0.4])
    ax1.scatter(d.iloc[:,0], d.iloc[:,1], marker = '.', linewidth = 0)
    ax1.set_xlabel(xlab)
    ax1.set_ylabel(ylab)
```

```
    ax2.set_xscale("log")
    ax2.set_yscale("log")
    ax2.set_xlabel(xloglab)
    ax2.set_ylabel(yloglab)
    ax2.scatter(d.iloc[:,0], d.iloc[:,1], marker = '.', linewidth = 0)


### Plot list of clusters
def plotmulticlus(cls, sizex, sizey):

    ncl = len(cls)

    fig, ax = plt.subplots(ncl, 2, figsize=[sizex, sizey], squeeze = False)
    plt.subplots_adjust(hspace = 0.3, wspace = 0.3)

    ni = 0
    for i in range(ncl):
        cl = cls[i]

        # Left column: cluster distribution
        a0 = ax[i][0]
        d = pd.DataFrame({'clid':range(1, len(cl)+1), 'n':map(len, cl)})
        a0.scatter(d.iloc[:,0], d.iloc[:,1], marker = '.', linewidth = 0)
        a0.set_xlabel("Cluster ID")
        a0.set_ylabel("# Elements")
        ia0 = inset_axes(a0, width=1.3, height=0.9)
        ia0.set_xscale("log")
        ia0.set_yscale("log")
        ia0.set_xlabel("log Cluster ID")
        ia0.set_ylabel("log #Elements")
        ia0.scatter(d.iloc[:,0], d.iloc[:,1], marker = '.', linewidth = 0)

        # Right column: neighbor distribution
        a1 = ax[i][1]
        d2 = pd.DataFrame({"clsize":d.n.groupby(d.n).unique(), "n":d.n.groupby
(d.n).sum()})
        a1.scatter(d2.iloc[:,0], d2.iloc[:,1], marker = '.', linewidth = 0)
        a1.set_xlabel("Cluster Size")
        a1.set_ylabel("# Elements")
        ia1 = inset_axes(a1, width=1.3, height=0.9)
        ia1.set_xscale("log")
        ia1.set_yscale("log")
        ia1.set_xlabel("log Cluster Size")
        ia1.set_ylabel("log # Elements")
        ia1.scatter(d.iloc[:,0], d.iloc[:,1], marker = '.', linewidth = 0)
    return


### Paint property histogram
def painthis(smidf, prop):

    pt.AddMoleculeColumnToFrame(smidf,"smiles")
    smidf['pr'] = smidf['ROMol'].map('Descriptors.' + prop)
    del smidf["ROMol"]
    ax = smidf['pr'].hist(bins = 50)
    ax.set_xlabel(prop)


### Paint a bunch of histograms
def paintmultihist(prs, xlab, nrow, ncol, xtxt, ytxt, sizex, sizey, legx, legy
, leg):

    mes = map(np.mean, prs)
```

```
    sds = map(np.std, prs)
    fig, ax = plt.subplots(nrow, ncol, figsize=[sizex, sizey], squeeze = False
)
    plt.subplots_adjust(hspace = 0.3, wspace = 0.3)

    ni = 0

    for row in range(nrow):
        for col in range(ncol):
            if ni < len(prs):
                ax[row][col].hist(prs[ni], bins = 40)
                ax[row][col].set_xlabel(xlab)
                ax[row][col].text(xtxt,ytxt, "Mean=" + str(round(mes[ni],2)))

                ax[row][col].text(xtxt,ytxt-
40, "SD=" + str(round(sds[ni],2)))
                ax[row][col].text(legx, legy, leg[ni])
                ni = ni+1

    plt.show()



### Whole diversity and novelty analysis for iterations
def wholean(it, name_train = "train", name_pref = "unc", th = 0.7):

    nit = len(it)
    df = pd.DataFrame(np.nan, index = range(1, nit+1),\
                        columns =\
                       ["# train","%corr inp","# un train","# clus inp","# fram
inp","# gen fram inp",\
                       "# out","%corr out","# un out","# clus out","# fram out",
"# gen fram out",\
                       "% new str","% new fram","% new gen fram"])

    cls = [] # List with lists of clusters

    for i in range(len(it)):

        # Find corrects and unique in input and fill ntrain, nuntrain and pcor
r inp
        smis = smif2smis('./' + name_train + str(it[i]) + '.smi')
        ncorr, n, smis, wrongsmis = corrsmis(smis)
        smis = list(set(smis))
        nuntrain = len(smis)
        smidft = smis2smidf(smis)
        del smis
        df["# train"].iloc[i] = n
        df["%corr inp"].iloc[i] = round(ncorr/float(n)*100,2)
        df["# un train"].iloc[i] = nuntrain

        # Find corrects and unique in output
        smis = smif2smis('./' + name_pref +str(it[i]) + '.smi')
        ncorr, n, smis, wrongsmis = corrsmis(smis)
        smis = list(set(smis))
        nunout = len(smis)
        smidfq = smis2smidf(smis)
        del smis
        df["# out"].iloc[i] = n
        df["%corr out"].iloc[i] = round(ncorr/float(n)*100,2)
        df["# un out"].iloc[i] = nunout

        # Generate arenas
        art = smidf2arena(smidft)
        arq = smidf2arena(smidfq)

```

```
494.         # Diversity analysis of input and fill nclus inp, nfram inp, ngenfram
    inp
495.         clb, fs, fg = divan(smidft, OnlyBu = True, arena = art)
496.         df["# clus inp"].iloc[i] = len(clb)
497.         df["# fram inp"].iloc[i] = len(fs)
498.         df["# gen fram inp"].iloc[i] = len(fg)
499.         cls.append(clb)
500.
501.         # Diversity analysis of output and fill nclus out, nfram out, ngenfram
     out
502.         clb, fs, fg = divan(smidfq, OnlyBu = True, arena = arq)
503.         df["# clus out"].iloc[i] = len(clb)
504.         df["# fram out"].iloc[i] = len(fs)
505.         df["# gen fram out"].iloc[i] = len(fg)
506.
507.         # Novelty analysis
508.         news, fraq, newfraqs, gfraq, newgfraqs = novan(smidfq, smidft, th = th
    , arq = arq, art = art)
509.         df["% new str"].iloc[i] = round(100*len(news)/float(smidfq.shape[0]),2
    )
510.         df["% new fram"].iloc[i] = round(100*len(newfraqs)/float(len(fraq)),2)

511.         df["% new gen fram"].iloc[i] = round(100*len(newgfraqs)/float(len(gfra
    q)),2)
512.
513.     # Return dataframe with output
514.     return df, cls
515.
516.
517.### Whole diversity and novelty analysis for iterations, returning in the outp
    ut also the clusters of the outputs
518.def wholean2(it, name_train = "train", name_pref = "unc", th = 0.7):
519.
520.     nit = len(it)
521.     df = pd.DataFrame(np.nan, index = range(1, nit+1),\
522.                       columns =\
523.                      ["# train","%corr inp","# un train","# clus inp","# fram
    inp","# gen fram inp",\
524.                       "# out","%corr out","# un out","# clus out","# fram out",
    "# gen fram out",\
525.                       "% new str","% new fram","% new gen fram"])
526.
527.     clsi = [] # List with lists of clusters input
528.     clso = [] # List with lists of clusters output
529.
530.
531.     for i in range(len(it)):
532.
533.         # Find corrects and unique in input and fill ntrain, nuntrain and pcor
    r inp
534.         smis = smif2smis('./' + name_train + str(it[i]) + '.smi')
535.         ncorr, n, smis, wrongsmis = corrsmis(smis)
536.         smis = list(set(smis))
537.         nuntrain = len(smis)
538.         smidft = smis2smidf(smis)
539.         del smis
540.         df["# train"].iloc[i] = n
541.         df["%corr inp"].iloc[i] = round(ncorr/float(n)*100,2)
542.         df["# un train"].iloc[i] = nuntrain
543.
544.         # Find corrects and unique in output
545.         smis = smif2smis('./' + name_pref +str(it[i]) + '.smi')
546.         ncorr, n, smis, wrongsmis = corrsmis(smis)
547.         smis = list(set(smis))
548.         nunout = len(smis)
549.         smidfq = smis2smidf(smis)
```

```
        del smis
        df["# out"].iloc[i] = n
        df["%corr out"].iloc[i] = round(ncorr/float(n)*100,2)
        df["# un out"].iloc[i] = nunout

        # Generate arenas
        art = smidf2arena(smidft)
        arq = smidf2arena(smidfq)

        # Diversity analysis of input and fill nclus inp, nfram inp, ngenfram inp
        clb, fs, fg = divan(smidft, OnlyBu = True, arena = art)
        df["# clus inp"].iloc[i] = len(clb)
        df["# fram inp"].iloc[i] = len(fs)
        df["# gen fram inp"].iloc[i] = len(fg)
        clsi.append(clb)

        # Diversity analysis of output and fill nclus out, nfram out, ngenfram out
        clb, fs, fg = divan(smidfq, OnlyBu = True, arena = arq)
        df["# clus out"].iloc[i] = len(clb)
        df["# fram out"].iloc[i] = len(fs)
        df["# gen fram out"].iloc[i] = len(fg)
        clso.append(clb)

        # Novelty analysis
        news, fraq, newfraqs, gfraq, newgfraqs = novan(smidfq, smidft, th = th, arq = arq, art = art)
        df["% new str"].iloc[i] = round(100*len(news)/float(smidfq.shape[0]),2)
        df["% new fram"].iloc[i] = round(100*len(newfraqs)/float(len(fraq)),2)
        df["% new gen fram"].iloc[i] = round(100*len(newgfraqs)/float(len(gfraq)),2)

    # Return dataframe with output
    return df, clsi, clso



### Diversity sampler 0
def divsamp0(ar, th = 0.7, nlimit = 300000, seed = 1234):
    start = time.time()

    np.random.seed(seed)
    rnin = np.arange(len(ar))
    np.random.shuffle(rnin)

    neig = set()
    sel = []

    for i in range(len(ar)):
        if(len(sel) < nlimit):
            if i == 0:
                sel.append(rnin[i])
                fp = ar[rnin[i]][1]
                res = chemfp.search.threshold_tanimoto_search_fp(fp, ar, threshold= th)
                neig.update(res.get_indices())
            else:
                if rnin[i] not in neig:
                    sel.append(rnin[i])
                    fp = ar[rnin[i]][1]
                    res = chemfp.search.threshold_tanimoto_search_fp(fp, ar, threshold=th)
```

```
                    neig.update(res.get_indices())
            print "i=" + str(i) + "; nsel=" + str(len(sel)) + "; nneig=" + str(len(neig)) + "\r",

    end = time.time()
    eltime = end - start

    print "i=" + str(i) + "; nsel=" + str(len(sel)) + "; nneig=" + str(len(neig)) + "; Sampling time: " + time.strftime("%H:%M:%S", time.gmtime(eltime))

    return sel


# Paint a bidimensional scatter/contour-density plot
def bidiplot(data, xlab, ylab, alpha = 1, s = 2, nbins = 20, d = False):

    if d is False:
        ax = plt.axes()
        ax.scatter(data[:,0],data[:,1], alpha = alpha, s = 2)
        ax.set_xlabel(xlab)
        ax.set_ylabel(ylab)
    else:
        ax = plt.axes()
        ax.set_xlabel(xlab)
        ax.set_ylabel(ylab)
        x, y = data.T
        nbins = 20
        k = kde.gaussian_kde(data.T)
        xi, yi = np.mgrid[x.min():x.max():nbins*1j, y.min():y.max():nbins*1j]

        zi = k(np.vstack([xi.flatten(), yi.flatten()]))
        plt.pcolormesh(xi, yi, zi.reshape(xi.shape), shading='gouraud', cmap=plt.cm.afmhot_r)
        plt.contour(xi, yi, zi.reshape(xi.shape))
        plt.colorbar()
        plt.show()
```