Creación de una herramienta de software para predecir la interacción, actividad y función de fármacos.

A tool for predicting drugs functions, activities and interactions.

Hugo A. Jiménez Hernández

Máster en Bioinformática y Bioestadística
Universitat Oberta de Catalunya

Trabajo de Fin de Máster

31 de enero de 2018

Licencias alternativas (elegir alguna de las siguientes y sustituir la de la página anterior)

A) Creative Commons:

B) GNU Free Documentation License (GNU FDL)

C) Copyright

Título del trabajo:	Creación de una herramienta de software para predecir la interacción, actividad y función de fármacos.
Nombre del autor:	*Hugo Jiménez Hernández*
Nombre del consultor/a:	*Melchor Sánchez Martínez*
Nombre del PRA:	*Maria Jesus Marco Galindo*
Fecha de entrega (mm/aaaa):	01/2018
Titulación::	*Máster en Bioinformática y Bioestadística*
Área del Trabajo Final:	*Bioinformática Farmacéutica*
Idioma del trabajo:	*Castellano*
Palabras clave	*Bases de datos, predicción, similitud*

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

El desarrollo de fármacos para el tratamiento de enfermedades es un área muy compleja con un gran evolución. Hoy sabemos que los factores que desencadenan la patogénesis son múltiples (factores ambientales, fisiológicos, genéticos...).

Esto influye enormemente en el desarrollo de tratamientos farmacológicos y es por ello que en los últimos años hay una larga tendencia al estudio de tratamientos personalizados. Mediante disciplinas como la polifarmacología, la farmacogenómica, etc. Estos tratamientos personalizados generan cantidades ingentes de información que no es posible tratar de manera convencional. Es por ello que hay que recurrir a técnicas de Big Data y algoritmos de aprendizaje automático (*Machine Learning* o ML). El presente trabajo busca el desarrollo de una herramienta que realice una búsqueda por similaridad (virtual screening) de una molécula contra una bases de datos de compuestos preparada a partir de bases de datos externas, y en segunda instancia que pueda predecir los compuestos activos para un target dado sobre los compuestos de la base de datos mediante algoritmos de ML.

Abstract (in English, 250 words or less):

The drug discovery process for the treatment of diseases is wide and complex area in constant evolution. Today is known that multiple factors are responsible for triggering pathogenesis (environment, physiology, genetics...). This greatly influence the development of pharmacological treatments and hence, as of late, there have been a trend on personalized treatments by means of several disciplines like polypharmacology, pharmacogenomics, etc.

This customized treatments yields a large amount of data which cannot be processed in a conventional way. Therefore, it is necessary to use Big Data mining and Machine Learning techniques. The present work aims to develop a tool that does a similarity search (virtual screening) of a given molecule against an specially curated database built from other compounds databases and, in a second stage, try to predict by means of Machine Learning algorithms active compounds for a given target.

Índice

1 Introducción

1.1 Contexto y justificación del Trabajo

La elaboración de nuevos fármacos para el tratamiento de patologías es un proceso complejo y costoso, que consta de múltiples etapas: identificar 'dianas' terapéuticas, encontrar una sustancia química que sea capaz de provocar en dichas dianas el efecto deseado y por supuesto asegurarse que esta sustancia no supone un riesgo para la seguridad del paciente.

Por supuesto, una de las mayores dificultades es la gestión de la gran cantidad de información que se genera en el proceso que requiere de técnicas de Big Data y algoritmos de aprendizaje automático (*Machine Learning*).

El presente trabajo pretende desarrollar una herramienta que mediante técnicas de virtual screening pueda realizar una búsqueda por similitud de un compuesto con respecto a una serie de compuestos almacenados en una base de datos creada a tal efecto y poder emplear algoritmos de machine learning para crear un modelo que pueda predecir la relación entre una diana y un compuesto dado.

1.2 Objetivos generales

A continuación se detallan los objetivos generales del proyecto así como una serie de objetivos más específicos que permitan acotar el ámbito del trabajo.

- Diseñar e implementar métodos para la extracción de información de una series de bases de datos bioquímicas sobre compuestos y dianas para su posterior tratamiento y uso.

- Desarrollar e implementar métodos para la predicción de las dianas terapéuticas.

1.2.2 Objetivos Específicos

1. Seleccionar aquellos datos que se necesiten de la base de datos chembl[1] para el desarrollo del trabajo.

2. Desarrollar e implementar métodos para crear una base de datos local de compuestos a partir de la información de la base de datos de compuestos-dianas.

3. Desarrollar e implementar métodos de *virtual screening* basado en la similitud entre un compuesto suministrado y los compuestos almacenados en la base de datos local.

4. Desarrollar e implementar métodos para la predicción de las posibles dianas biológicas de un compuesto mediante algoritmos de *Machine Learning*.

5. Desarrollar una serie de pruebas para medir el rendimiento de los métodos implementados.

1.3 Enfoque y método seguido

1.3.2 Cálculo de similitud mediante Virtual Screening

En primer lugar, se ha planteado realizar una búsqueda de compuestos en base a su similitud empleando técnicas de virtual screening. Existen diversas técnicas de virtual screening, aunque se pueden agrupar en dos grandes categorías:

- *Structure-based:* Este método se basa en la aplicación de funciones que puntuan el acoplamiento de ligandos[2] candidatos a la proteina objetivo para estimar la probabilidad de que el ligando se acople a la proteina receptora con gran afinidad. Este método requiere un conocimiento de la estructura tridimensional del receptor y de sus centros activos.
- *Ligand-based*: Este método depende de la información sobre los ligandos que interactuan con el receptor de interés.

En este caso, se decide optar por la vertiente ligand-based pues suele ser lo habitual en el caso de trabajar con información de moleculas en 2D[1] [2]. En el

1 Base de datos de compuestos farmacológicos, pequeño compuesos y sus respectivas dianas.

2 Sustancia que forma un complejo con una biomolécula. Para más información vease ligando.

caso de tratar con información tridimensional, sería más óptimo trabajar con structure-based o directamente recurrir a algoritmos Machine Learning.

1.3.3 Clasificación de compuestos frente a un target con algoritmos de Machine Learning

En segunda instancia, se ha planteado mediante algoritmos de Machine Learning desarrollar métodos que permitan clasificar una molécula como activa o no con respecto a un determinado *target*.

Se conoce que las máquinas SVM generan muy buenos resultados, como indican [3] y [4]. No obstante, se planteó que si el tiempo lo permitía, analizarían diferentes soluciones usando no solo SVM, sino también otros algoritmos de Machine Learning como las ANN, *Random Forest* y *Naïve Bayes*.

1.3.4 Selección de soporte de almacenamiento

Tras sopesar diferentes opciones de bases datos SQL (MySQL, PostgreSQL, SQLite) y noSQL (couchDB, Cassandra, MongoDB), se ha optado por utilizar MongoDB como base de datos de almacenamiento local para la información de compuestos y dianas. Más adelante se mostrará una comparativa entre las BBDD relacionales y noSQL y las razones que llevan a decantarse por MongoDB.

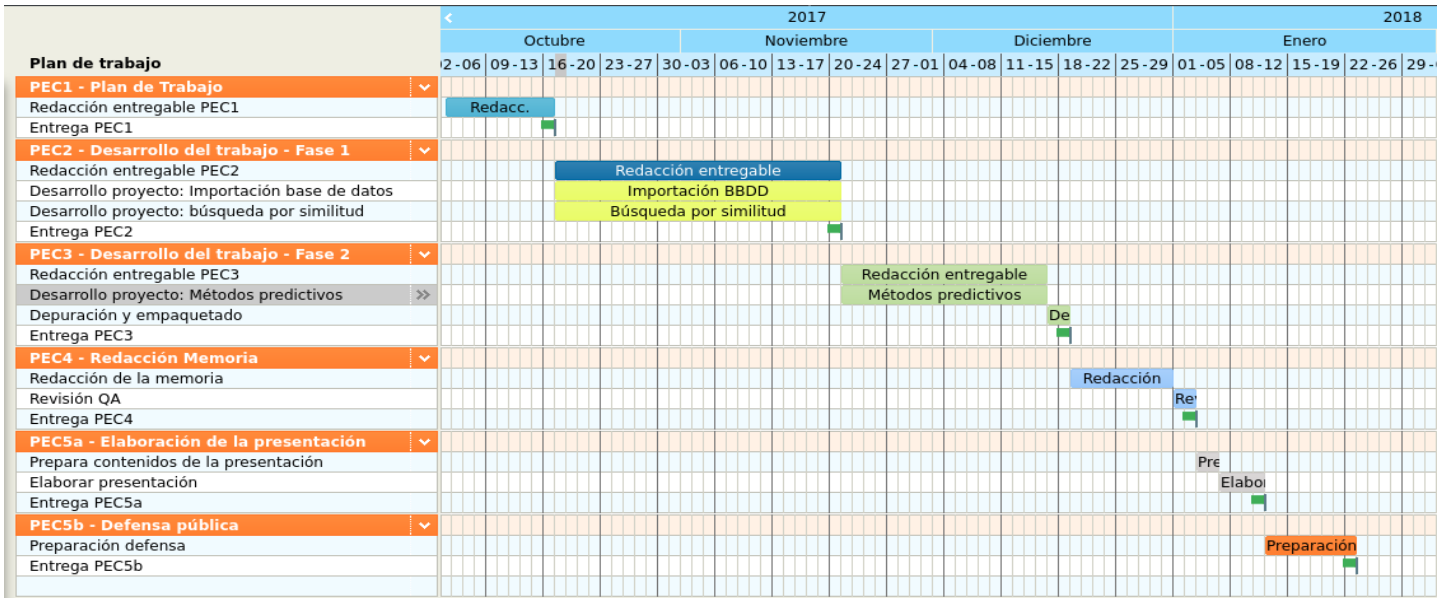
1.4 Metodología de desarrollo

Para el desarrollo de la aplicación, se ha optado por emplear el lenguaje de programación Python así como una metodología de desarrollo dirigida por pruebas (TDD[5]). De esta manera se pretende acelerar los tiempos de desarrollo a la par que dotar al código de robustez frente a nuevos cambios. Por otro lado, se pretende desarrollar una batería de pruebas para comprobar el rendimiento de los algoritmos de búsqueda y clasificación que se implementen durante el desarrollo de la aplicación, atendiendo a pruebas diagnósticas como la sensibilidad y la especificidad o los valores predictivos (positivos y negativos). Se ha optado por esta metodología, pues con ella se pretende reducir la cantidad de bugs generados y acelerar los tiempos de desarrollo y aportar robustez frente a los cambios que se vayan dando durante el desarrollo.

1.5 Requerimientos de software

Función	Software
Sistema operativo	Arch Linux
Base de datos Local	MongoDB
Librería para trabajar con MongoDB	PyMongo
Librería para manipular información de BBDD externas	Pandas
Librerería Métodos *Machine Learning*	SciKit Learn
Librería para el desarrollo de aplicación quimioinformáticas	RDKit
Libreria para pruebas unitarias	pytest

1.6 Planificación del Trabajo



1.6.2.1 Hitos

- Base de datos local con información importada y métodos de búsqueda por similitud – 19 de noviembre de 2017.

- Entrega PEC2 – 20 de noviembre de 2017.

- Métodos predictivos – 15 de diciembre de 2017.

- Depuración y empaquetado – 18 de diciembre de 2017.

- Entrega PEC3 – 18 de diciembre de 2017.

- Entrega PEC4 – 2 de enero de 2018.

- Elaboración presentación y entrega PEC5a – 10 de enero de 2018.

- Preparación defensa pública y entrega PEC5b – 11 de enero de 2018 a 22 de enero de 2018.

1.6.2.2 Análisis de riesgos

A continuación se detallan posibles incidencias que pueden surgir a lo largo de la ejecución del trabajo y las correspondientes medidas para prevenirlas o darles solución.

Averías de equipo: Es posible que durante el transcurso del trabajo el equipo de desarrollo sufra algun tipo de avería (Rotura de pantalla, subidas de tension, avería de disco duro, etc). Para ello se dispone de varios equipos de sustitución. Así mismo, tanto el código fuente como la memoria estarán respaldados mediante sistema de control de versiones (vease [15] o [16])

Retrasos en la planificación: Ya sea debido a limitaciones, cambios en las especificaciones o requerimientos de la aplicación. Como contramedida y de manera preventiva, siempre se asigna al menos un 40% más de tiempo de dedicación a las tareas como margen en caso de contratiempos. En caso de que sea insuficiente, se hará un reparto del tiempo usando parte del dedicado a tareas menos prioritarias.

Problemas de integración: Es posible que se den incompatibilidades o problemas de integración entre los distintos componentes de software. Para

ello, durante la etapa de diseño y en previsión a este tipo de eventualidad, se buscan alternativas tanto de componentes como de metodologías de integración. Por ejemplo, si una librería no se puede compilar se puede intentar extraer de ella aquellas funciones que se necesiten.

Objetivos definidos incorrectamente: Es posible que por desconocimiento, falta de comunicación o experiencia no se lleguen a detallar todos los aspectos relevantes para llevar a cabo el trabajo en la fase de definición de requisitos. Para evitar contratiempos, es importante mantener una comunicación fluida y constante con el director del TFM e informarle de los resultados parciales durante la ejecución del trabajo.

1.6.3 Resultados esperados

Los siguientes son los productos que se esperan obtener al finalizar el trabajo:

- Planificación del trabajo, con descripciones detalladas de cada etapa.
- Memoria del trabajo
- Aplicación instalable con las funcionalidades elaboradas durante la realización del trabajo
- Bases de datos local con la información relevante para el desarrollo del trabajo.
- Métodos de virtual screening para medir la similitud entre moléculas.
- Métodos de virtual profiling para la predicción de *targets* mediante algoritmos de machine learning.

1.6.4 Estructura del proyecto

El proyecto se divide en los siguientes capítulos:

1. Interfaz de BBDD: Preparación de los datos.

En este capítulo se aborda el proceso de creación de la base de datos local sobre la cual se realiza toda la operatoria del proyecto. Se discute ademas la selección de las bases de datos de compuestos-dianas así como la base de datos escogida para almacenar localmente los datos, incluyendo las razones por las que se ha escogido.

2. Virtual Sreening: busqueda por similitud

En este capítulo se expone el uso de las técnicas de *Virtual Screening*. Se incluye una comparativa de varios tipos de algoritmos de fingerprinting y se

expone que algoritmo es escogido para su uso en conjunción con el índice de similitud de tanimoto.

3. Clasificación en la actividad de compuestos con respecto a un target

En este capítulo se expone la creación de métodos para la clasificación de compuestos en base a la actividad o no sobre un determinado *target*.

4. Conclusiones

En este capítulo se exponen las conclusiones, se realiza un análisis retrospectivo de la planificación.

5. Analisis del Avance del proyecto

En este anexo se exponen los avances del proyecto, el grado de cumplimiento de los hitos y objetivos así como la justificación de cambios o realización de actividades no previstas y resolución de problemas.

6. Resultados de pruebas comparativas

En este anexo se exponen los resultados de las diferentes pruebas realizadas mediante el software desarrollado.

7. Bibliografía

En este anexo se compilan los enlaces referenciados en la memoría a artículos de blogs, papers, libros, etc.

2 Adaptación y Almacenamiento de los datos

2.1 Enfoque

Para trabajar con los datos contenidos en las bases de datos de compuestos es necesarios seleccionar y adaptar la información que sea necesaria.
En el presente trabajo se ha seleccionado la base de datos ChEMBL para como base de datos para realizar todas las pruebas. Por supuesto es posible trabajar con otras bases de datos, pero a efectos prácticos para el desarrollo del trabajo es más que suficiente.

En primer lugar, se ha seleccionado entre todas las tablas de la base de datos aquellas relevantes para el cálculo de similitud y la búsqueda por algoritmos de machine learning. Por un lado los SMILES para poder calcular sus correspondientes fingerprints y las diferentes características fisicoquímicas que fueran relevantes para la predicción de *targets* mediante los algoritmos de ML. En segundo lugar, una relación entre compuestos y *targets* para entrenamiento y la predicción de los algoritmos ML.

- ALOGP[3]: Valor calculado para la lipofilicidad de una molécula, es decir, la capacidad de dicha molécula de disolverse en una solución lipídica, expresada en formato logarítmico.
- HBA[4]: Número de aceptadores de enlaces de hidrógeno.
- HBD[5] Número de donantes de enlaces de hidrogeno.
- PSA[6]: Area de superficie para átomos polares.
- RTB[7]: Número de enlaces con rotación libre en una molécula.
- Canonical SMILES[8]: Identificador único en formato de texto para un compuesto determinado.

2.1.2 Local vs Remoto

La mayoria de bases de datos proporcionan APIs REST para acceder a la información de compuestos y ChEMBL no una excepción. No obstante, durante la fase de análisis del proyecto se opto por el almacenamiento local debido por un lado a la **disponibilidad**, es decir, tanto problemas de conexión

3 Véase coeficiente de reparto.
4 Véase *Hydrogen Bonding Aceptor*.
5 Véase *Hydrogen Bonding Donor*.
6 Véase *Polar Area Surface*.
7 Véase *Rotable Bonds*.
8 Véase *Canonical SMILES*.

como del propio servidor nos impedirían el acceso a la información. Por otro lado, **no existe una solución estandarizada** actual para el acceso mediante API REST en Python y el desarrollo de los métodos necesarios se escapa del objeto del presente trabajo. Además, el uso de la API REST **añadiría más latencia** al pipeline del trabajo. Por todo esto, se ha optado por el almacenamiento en local.

2.1.3 Sistema de almacenamiento

Tradicionalmente, para el almacenamiento de información se han empleado las llamadas bases de datos relacionales. Por otro lado, han surgido en los últimos años las denominadas bases de datos NoSQL, las cuales se presentan como una alternativa a las bases de datos SQL principalmente en base a los siguientes aspectos:

SQL	NOSQL
Schema Obligatorio	Sin Schema
Escalabilidad vertical	Escalabilidad Horizontal
Rendimiento dependiente de los recursos del servidor	Rendimiento basado en las medidas de consistencia

Teniendo en cuenta la información a almacenar y como ya se adelantó en el anterior capítulo, se ha escogido la bases de datos NoSQL MongoDB partiendo de las características arriba mencionadas y en base a lo siguiente:

- Si bien se plantea un diseño inicial de como se almacenaran los datos en local, es factible que esta información varíe, por lo tanto es interesante no verse limitados por el schema de la base de datos como es el caso de las bases de datos relacionales.

- Aunque en un principio se harán bastantes escrituras en bases de datos, no se presentarán problemas de inconsistencia por escrituras concurrentes ya que la inserción de la información será manual y solo se realizarán accesos de lectura con lo cual podemos prescindir del forzado de la consistencia, ganando así en rendimiento.

- La escritura y acceso de la información resulta más sencillo que realizando complicadas consulta en lenguaje SQL.

- Dentro del conjunto de bases de datos NoSQL, es la que mejor soporte tiene, incluyendo una interfaz para python con todo lo necesario para el proyecto.

2.1.4 Métodología de Importación

La base de datos ChEMBL no proporciona un dump para mongoDB. Por ello es necesario importar la información de los dumps de bases de datos más reciente que ChEMBL tenga publicados.

2.1.4.1 Preparando dumps de bases de datos

Para importar los datos a la base de datos MongoDB local, se ha seleccionado un dump para PostgreSQL de la versión más reciente a fecha de edición de este trabajo.

En primer lugar se han creado en la base de datos PostgreSQL unas vistas con la información necesaria para el trabajo. Esto se ha hecho para facilitar la importación y para acelerarla ya que de está manera se reduce la cantidad de información con la que el motor de Base de datos PostgreSQL tiene que trabajar. Se puede ver el código SQL correspondiente en en el anexo correspondiente.

2.1.4.2 Importando dumps preprocesados a MongoDB

Una vez creada las vistas en la base de datos PostgreSQL, se procede a crear los registros para insertar en la base de datos MongoDB.
Cada registro almacena la información que se puede necesitar durante el proyecto. Entre otras cosas, el registro contiene el ID de ChEMBL, el objeto MOL[9] creado mediante la librería RDKIT, así como varios tipos de fingerprints, las propiedades de la molécula en cuestión y los *targets* asociados a esa molécula que se emplearán en la segunda parte del trabajo con la predicción de *targets* mediante algoritmo de *Machine Learning.*

9 Objeto creado a partir de la representación SMILES de una molécula mediate la librería rdkit para realizar las diferentes operativas de cálculo de similitud.

3 Búsqueda por similitud

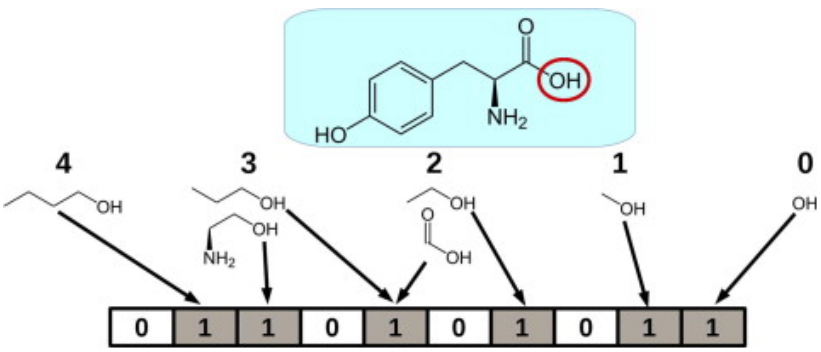
El proceso de desarrollo de fármacos pasa por identificar compuestos nuevos que interacción con una diana. En primera instancia, esto requeriría probar todas las posibles dianas disponibles en la base de datos, lo cual sería tremendamente costoso en tiempo y recursos.

Para ello y siguiendo la premisa de que los compuestos con estructuras similares deben tiene funciones similares, se efectúan búsquedas por similitud del nuevo compuesto.

3.1 Fingerprint molecular

Comparar una molécula contra toda una base de datos de moléculas es una tarea que puede resultar tremendamente costosa. Por ello es necesario buscar algún metodo que simplifique esta tarea. En general, para representar una molécula se suele emplear su representacion en SMILES o InChi. Dicha representación permite identificar la molécula de manera única[10]. No obstante, en términos computacionales, comparar dos cadenas de texto no es eficiente en absoluto.

Una firma o *fingerprint* molecular es una representación binaria de una molécula. El ser una cadena binaria permite reducir la complejidad computacional de la comparativa entre moléculas ya que permite aplicar directamente funciones como las de cálculo de distancia.



Illustración 1: Circular Fingerprinting

10 Se puede [illegible] comparten la misma ca[illegible]

Existen diversos algoritmos para el cálculo de los fingerprints de compuestos. Mediante la librería rdkit, se ha optado por emplear el algoritmo de fingerprinting de tipo circular Morgan.

3.1.2 Morgan Fingerprint

Este tipo de *fingerprint* denominado *Extended Connectivity Fingerprint* (ECFP) emplea el algoritmo de Morgan que como todos los fingerprints de tipo circular utiliza el conocimiento que tiene cada átomo de sus «vecinos» para construir la firma.

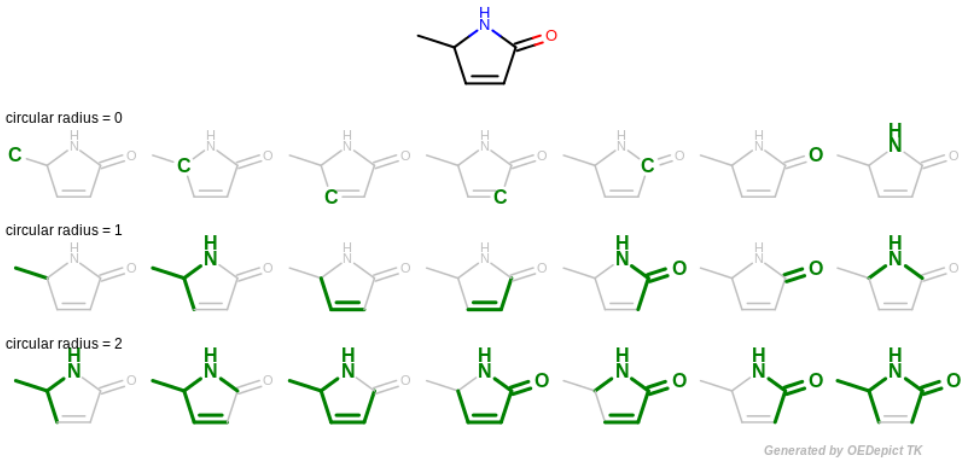


Ilustración 2: Fingerprint circular: según número de radios

Como se indica en [31], el mejor algoritmo de *fingerprinting* para virtual screening depende del tipo de *dataset.* Por ello y por que Morgan ofrece valores medios de similaridad.

3.2 Cálculo de la similitud

Medir la similitud de dos compuestos se convierte en esencia en un cálculo de distancia entre fingerprints. Al igual que existen muchos algoritmos para la obtención de los fingerprints, existen otros tantos para el cálculo de similitud. De todos ellos, el más empleado es el índice de Tanimoto y es el que se ha escogido en este trabajo.

$$S_{A,B} = \frac{\sum_{j=1}^{j=n} x_{jA} x_{jB}}{\sum_{j=1}^{j=n} \left(x_{jA}\right)^2 + \sum_{j=1}^{j=n} \left(x_{jB}\right)^2 - \sum_{j=1}^{j=n} x_{jA} x_{jB}}$$

Ilustración 3: Fórmula del índice de tanimoto

3.3 Workflow

Para realizar el cálculo de similitud, se han precalculado, mediante los métodos de la librería rdkit, los fingerprints de los compuestos de la base de datos local y almacenado en la misma. A su vez, se ha calculado el fingerprint para el compuesto a comparar.

A continuación se ha iterado por cada uno de los compuestos de la base de datos y se ha ejecutado el función de cálculo del índice de tanimoto. En este caso se seleccionado un umbral de similitud del 85% y 90%.

3.4 Pruebas de cálculo de similitud

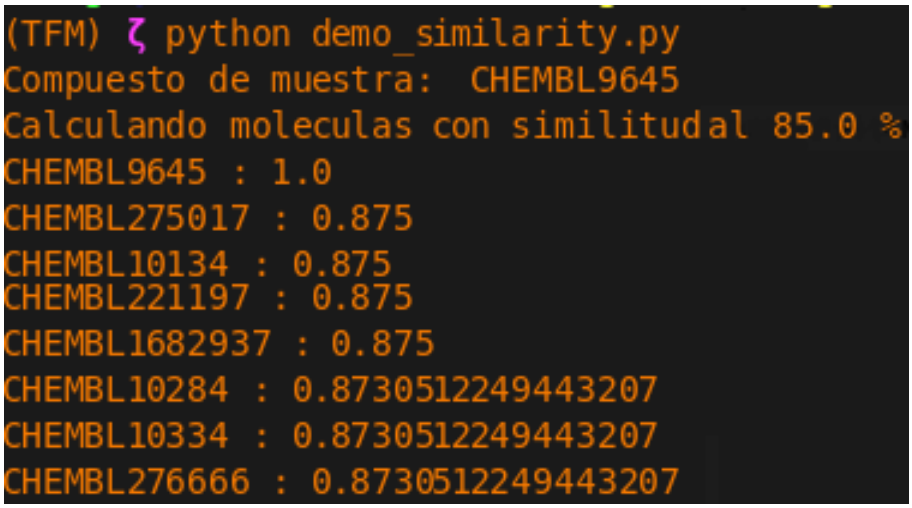
Para comprobar el funcionamiento de las funciones de cálculo de similitud, se han usado un par de compuestos incluidos dentro de la base de datos local. Se han realizado pruebas con varios compuestos de la base de datos. Por tanto se esperan obtenerse un listado de moléculas con al menos una con coincidencia perfecta (coincidencia de 1).

En el siguiente fragmento de código se puede ver como se extraen en primer lugar los compuestos. Se fija un límite inferior de similitud del 85%. A continuación se escoge una molecula de muestra (en este caso el compuesto CHEMBL419) y se extrae su fingerprint. Se itera por todos los compuestos de la base de datos, se extrae el fingerprint en cada caso y se compara con el fingerprint de la molecula de muestra mediante la función de cálculo de similitud de tanimoto. Finalmente se imprimen los resultados ordenados de mayor a menor coincidencia.

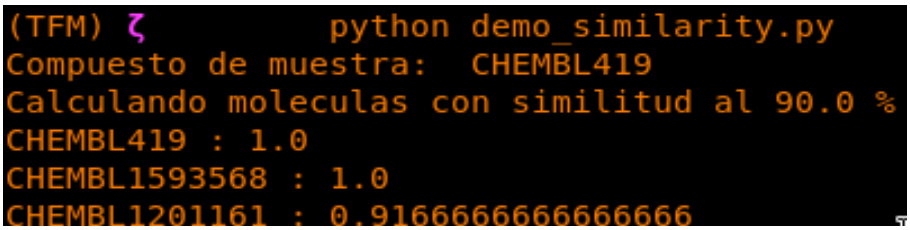
```
compounds = MongoClient('mongodb://127.0.0.1:27017/')['DrugDealer']["compounds"]
threshold = 0.85
similarity = ComputeSimilarity()
sample_molecule = compounds.find({'CHEMBL_ID':'CHEMBL9645'})[0]
print ("Compuesto de muestra: ", sample_molecule['CHEMBL_ID'])
sample_molecule_fingerprint = sample_molecule["FingerPrints"]["Morgan"]["data"]
results = {}
print("Calculando moleculas con similitud al", threshold * 100, "%")
for molecule in compounds.find():
    try:
        tanimoto = similarity.getTanimotoSimilarity(loads(sample_molecule_fingerprint),
                                                    loads(molecule["FingerPrints"]["Morgan"]["data"]))
        if tanimoto > threshold:
            results[molecule['CHEMBL_ID']] = tanimoto
    except:
        pass
for result  in sorted(results.items(), key=lambda x: x[1], reverse=True):
    print (result[0],":", result[1])
```

Extracto de código para el cálculo de la similitud

A continuación se muestran capturas con el listado de moleculas similares a una molecula escogida para similitudes superiores al 85 y al 90% respectivamente.



Calculo de similitud para el compuesto CHEMBL9645 al 85%



Calculo de similitud para el compuesto CHEMBL419 al 90%

4 Métodos predictivos

En la medida en que las bases de datos de compuestos se hacen más grandes es necesario elaborar estrategias de búsqueda que sean más eficientes. Para ello existe un enfoque denominado QSAR (Quantitative *Structure-Activity Relationship*) que en esencia relaciona la estructura molecular o tridimensional con su actividad biológica. En el presente trabajo se han empleado algunos de los más importantes algoritmos de ML para relacionar un compuesto dado con su target correspondiente.

Con el propósito de obtener el mejor resultado posible con los datos almacenados de compuestos, se han seleccionado los algoritmos de Naive Bayes, Redes Neuronales, Regresión Logística y Random Forest.

4.1 Metodología

4.1.2 Selección de los datos

La construcción de los datos de entrenamiento y testeo se ha llevado a cabo seleccionando los compuestos activos de un Target dado. Una vez obtenidos, se han seleccionando el mismo número de compuestos no activos para que actuasen como *decoys*[11]. Finalmente se ha creado un solo conjunto aleatorizado de compuestos activos y *decoys* y se ha divido en dos conjuntos al 80% y al 20% para las fases de entrenamiento y de testeo respectivamente.

4.1.3 Elaboración del modelo

Para elaborar los modelos de ML, se ha empleado la librería sci-kit learn. Cada modelo tiene sus casuísticas particulares pero en general el proceso aplicado es el siguiente:

- Prueba de ajuste y selección de los hiperparámetros mediante el método **RandomizedSearchCV**.
- Ajuste definitivo del modelo.
- Prueba predictiva del modelo con los datos de testeo en base a la puntuación proporcionado por el **estimador F1**[12]

Se han seleccionado tres *targets* para la elaboración de los modelos. La intención del presente trabajo es constrastar la capacidad predictiva entre los diferentes algoritmos. Por ello se han seleccionado targets que tengan asociados diferentes número de compuestos activos para ver como se

11 Moléculas que no son activas con respecto a una determinada diana biológica.
12 Véase *F1 score*.

comportan los clasificadores en casos en los que el conjunto de entrenamiento sea pequeño, mediano o grande.

Como medida adicional y para comprobar la eficacia de los clasificadores, se ha repetido la elaboración del modelo para cada algoritmo 10 veces y se ha hallado la media de los resultados para evitar sesgos en la distribución aleatoria del conjunto de datos de entrenamiento y testeo.

El siguiente código es un extracto que muestra el proceso de clasificación del modelo de Naïve-Bayes. En primer lugar se preparan los conjuntos de datos de entrenamiento y de prueba. A continuación se suministra al modelo estos para su entrenamiento y se realiza un prueba predictiva. Con los resultados de la predicción se emplean los métodos del paquete “metrics” para extraer la precisión y la puntuación del estimador F1.

```
print("Clasificador Naïve Bayes\n")
for i in range(0, 10):
    data_train, data_test, type_train, type_test = prepare_data(Target)
    predicted_NB = GaussNB_classification(data_train, data_test, type_train, type_test)
    results_accuracy_score.append(metrics.accuracy_score(type_test, predicted_NB))
    results_f1_score.append(metrics.f1_score(type_test, predicted_NB))


print("Precisión: ", np.mean(results_accuracy_score))
print("Puntuación F1: ", np.mean(results_f1_score))
```

Extracto de código para clasificación de compuestos mediante Naive-Bayes.

4.2 Modelos y resultados

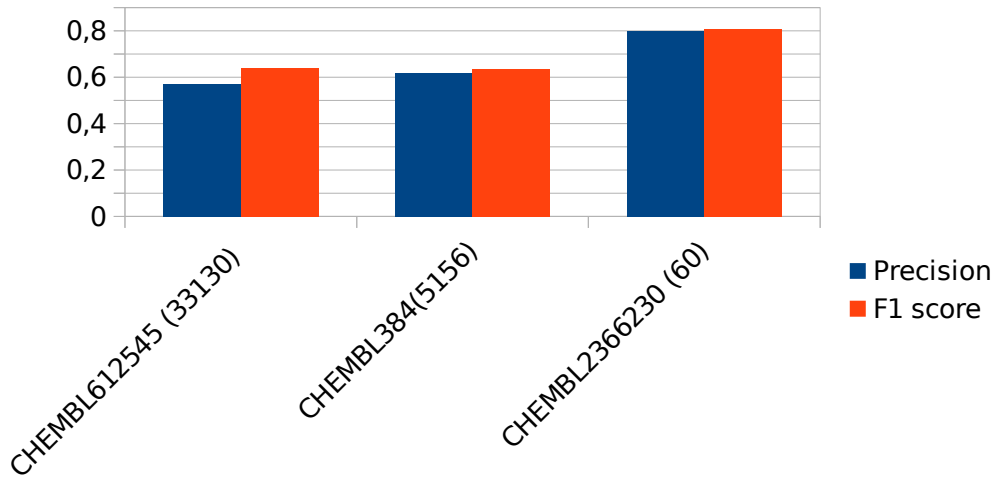
4.2.2 Naïve Bayes

Naive Bayes es una técnica para construir clasificadores probabilísticos que basan su funcionamiento en el teorema de Bayes. La característica principal de este tipo de clasificadores es que se asume que el valor de una característica es independiente al valor de cualquier otra característica.

Para crear este modelo se ha empleado el método GaussNB de la libreria sklearn. Debido a su funcionamiento no es necesario suministrar ningún hiperparámetro para el ajuste del modelo.

A continuación se puede comprobar los resultados para cada Target.

Target(num de compuestos)	Precision	F1 score
CHEMBL612545 (33130)	0,5706	0,6383
CHEMBL384(5156)	0,6189	0,6359
CHEMBL2366230 (60)	0,8	0,8092

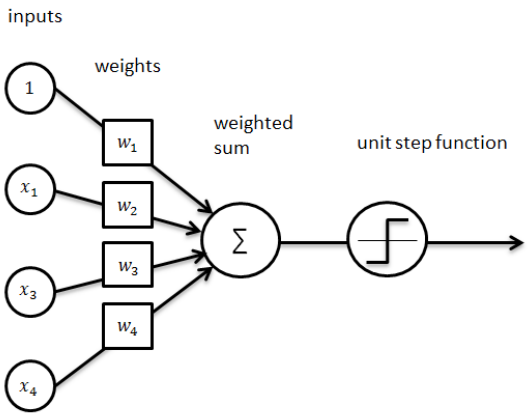


Target (número componenetes activos)

4.2.3 Redes Neuronales Artificiales

Las RNA son un conjunto de modelos de aprendizaje automático que basa su funcionamiento en el comportamiento de las neuronas de los seres vivos.

Siendo la unidad mínima el perceptrón (el equivalente a la neurona), este se compone de una combinación lineal de parametros de entrada asociados a un vector de pesos. Estas entradas son suministradas a una función denominada función de activación que decide si para una determinada combinación de parámetros entrada se suministra un valor u otro como salida.



Esquema de un perceptrón

Para elaborar el modelo se ha empleado el método MLPClassifier para crear el modelo de RNA. Para intentar obtener una precisión mayor en la clasificación,

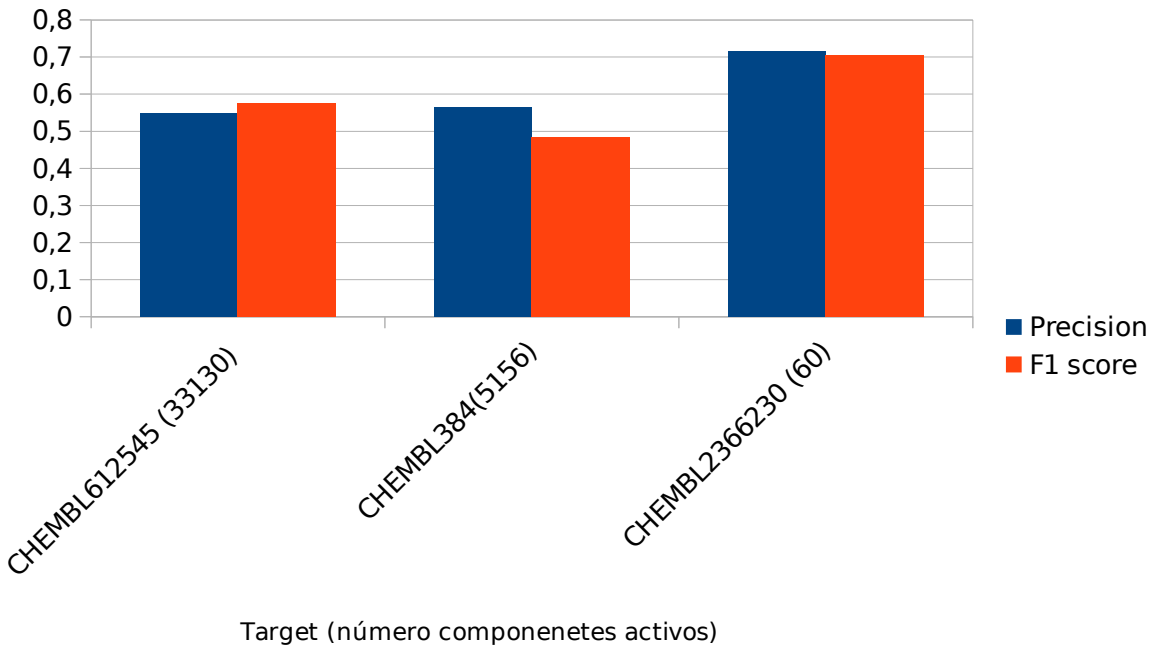
mediante el método RandomizedSearchCV se ha empleado la siguiente combinaciones de hiperparámetros para el ajuste del modelo[30]:

```
param_grid = {"alpha": [.1, .01, .001],
              "momentum": [.7, .9],
              "learning_rate_init": [.1, .01, .001]
}
```

A continuación se puede comprobar los resultados para cada Target.

Target(num de compuestos)	Precision	F1 score
CHEMBL612545 (33130)	0,5488	0,5754
CHEMBL384(5156)	0,5654	0,4841
CHEMBL2366230 (60)	0,7145	0,7041

Resultados Clasificación con Algoritmo de Redes Neuronales



4.2.4 Regresión Logística

La regresión logística es una técnica de estadística clásica que muchas veces se engloba dentro del ML. Es un tipo de regresión que está indicada especialmente para la predicción de variables categóricas (En el caso del presente trabajo, si un target es activo para un compuesto o no).

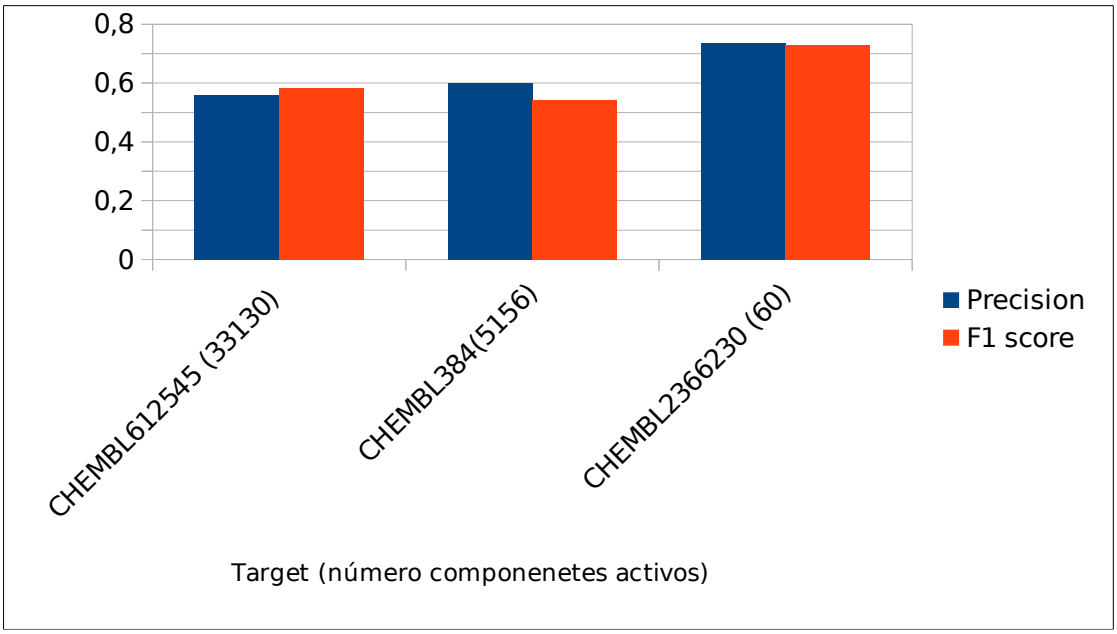
Para intentar obtener una precisión mayor en la clasificación, mediante el método RandomizedSearchCV se ha empleado la siguiente combinaciones de hiperparámetros para el ajuste del modelo:

```
param_grid = {'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
              'solver': ['newton-cg', 'lbfgs'],
              'max_iter': [100, 250, 1000, 4000]
             }
```

A continuación se puede comprobar los resultados para cada Target:

Target(num de compuestos)	Precision	F1 score
CHEMBL612545 (33130)	0,5607	0,5818
CHEMBL384(5156)	0,599	0,542
CHEMBL2366230 (60)	0,7375	0,7279

Resultados Clasificación mediante algoritmo de regresión logística



4.2.5 Random Forest

Random Forest es un clasificador que se engloba dentro del conjunto de clasificadores denominados *Bootstrap bagging* (o simplemente *bagging*) dentro de un grupo más amplio denominado métodos de ensamblado. El mecanismo El mecanismo de este clasificador es combinar una serie de árboles de decisión sobre subconjuntos de los datos de entrenamiento [29]

Este modelo es el que ofrece por norma general una precisión más alta, especialmente cuando el conjunto de datos es muy grande.

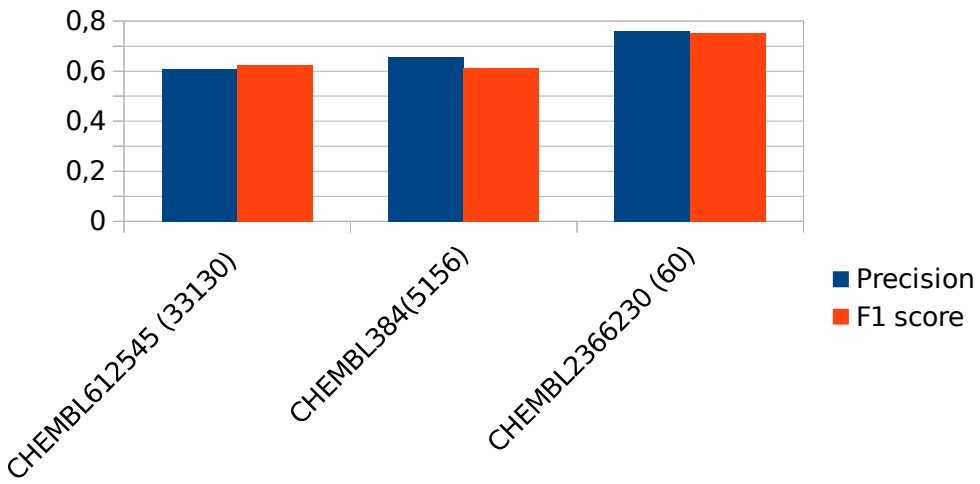
Como en los anteriores clasificadores se ha empleado el método RandomizedSearchCV para el ajuste del modelo con los siguientes hiperparámetros:

```
param_dist = {"max_depth": [None],
      "max_features": ['log2', None],
      "bootstrap": [True, False],
      "criterion": ["gini", "entropy"]}
```

Hiperparámetros para modelo de Random Forest

A continuación se puede observar los resultados del clasificador:

Target(num de compuestos)	Precision	F1-score
CHEMBL612545 (33130)	0,6094	0,6255
CHEMBL384(5156)	0,6544	0,6121
CHEMBL2366230 (60)	0,7593	0,7529



Target (número componentes activos)

4.3 Discusión de los resultados

Como ya se mencionó en el apartado de elaboración del modelo, se decidió emplear el estimador F1 para medir el rendimiento de los diferentes modelos.
El mejor clasificador es aquel que tenga una combinación de sensibilidad y valor predictivo positivo más alto. Esto expresado a través del estimador F1.

De los cuatro clasificadores empleados, Naïve Bayes presenta las puntuaciones de F1-score más alta para los tres targets.
Si se observan nuevamente los resultados, se puede comprobar que para un número medio y grande de compuestos, la potencia de clasificación de similar. Mientras que para un número bajo de compuestos se obtiene una potencia de clasificación mucho mayor. De aquí se puede concluir por un lado que a partir de un determinado número de compuestos, el rendimiento de naïve-bayes tiende a “estancarse”. Por otro lado, se observa que el rendimiento de Naïve-Bayes tiene un mejor rendimiento para conjuntos pequeños de datos como ya se indica en [32].

Target(num de compuestos)	Precision	F1 score
CHEMBL612545 (33130)	0,5706	0,6383
CHEMBL384 (5156)	0,6189	0,6359
CHEMBL2366230 (60)	0,8	0,8092

5 Análisis del avance del proyecto

A continuación se muestra un análisis de la consecución de los distintos hitos que conforman el trabajo respecto al plan fijado. Así mismo se listan las modificaciones que hayan sido necesarias con respecto a entregas pasadas de la memoria.

5.1 Progresión de los objetivos - PEC2

- Base de datos local con información importada y métodos de búsqueda por similitud.
  - Fecha prevista de finalización: 19 de noviembre de 2017.
  - Fecha de finalización: 2 de noviembre de 2017.
  - Comentarios: Aunque la importación y la gestión de la base de datos ha sido la parte más laboriosa del trabajo, se ha podido finalizar antes de la fecha prevista.

- Métodos predictivos

  - Fecha prevista de finalización: 15 de diciembre de 2017.
  - Fecha de finalización: 13 de noviembre de 2017.
  - Comentarios: Aún faltando por elaborar una prueba demostración de las funciones desarrollados, los métodos de predicción por *Machine Learning* se han terminado un més antes de la fecha prevista de finalización. Esto permitirá un mayor tiempo dedicado a corregir posibles errores que hayan pasado por alto y depurar el rendimiento de las distintas funciones.

- La entrega de las PEC va según lo establecido en el calendario.

5.2 Registro de cambios - PEC2

- Corrección de errores en fechas del listado de hitos.
- Se actualiza el índice para adaptarse a los contenidos solicitados en la PEC2.
- Correcciones menores.

5.3 Progresión de los objetivos - PEC3

- Depuración y empaquetado
  - Fecha prevista de finalización: 18 de diciembre de 2017

    - Fecha de finalización: 16 de diciembre de 2017
    - Comentarios: Debido al adelanto en la fecha de finalización del hito de Métodos predictivos se ha podido adelantar la fase de depuración y empaquetado del código.

- Métodos predictivos

    - Fecha prevista de finalización: 15 de diciembre de 2017.
    - Fecha de finalización: 10 de diciembre de 2017.
    - Comentarios: Aún faltando por elaborar una prueba demostración de las funciones desarrollados, los métodos de predicción por *Machine Learning* se han terminado un més antes de la fecha prevista de finalización. Esto permitirá un mayor tiempo dedicado a corregir posibles errores que hayan pasado por alto y depurar el rendimiento de las distintas funciones.

- La entrega de las PEC va según lo establecido en el calendario.

5.4 Registro de cambios - PEC3

- Cambio del Threshold al 85% y al 90% en el cálculo de similitud.
- Añadidos resultados de pruebas de similitud.
- Añadido Capítulo sobre los algoritmos de Machine Learning donde se da una explicación de cada uno y se exponen los resultados obtenidos.
- Atendidas las sugerencias del informe de Evaluación de la PEC2.
- Actualizada el informe de progresión de objetivos.

5.5 Registro de cambios - PEC4

- Se atienden cambios sugeridos por el director de proyecto
- Se cambia el glosario por notas a pie de páginas con hipervínculos para facilitar la lectura.
- Se agrega apartado de discusión de resultados a los métodos predictivos.

6 Conclusiones

Al termino del presente trabajo se ha podido desarrollar e implementar los métodos para crear la base de datos local de compuestos. Por una parte ha supuesto cierta dificultad extraer la información concreta de compuestos ya que la base de datos chembl contiene un esquema con numerosas tablas y no se tenía el conocimiento previo del mismo. Por otra parte el no tener un servidor a parte con la base de datos cargada, provocaba consumos de memoria principal elevados. No obstante, esto se pudo resolver limitando la memoria asignada a la cache del proceso de MongoDB.

Con respecto a los métodos de virtual screening, esto no supuso apenas carga de trabajo, puesto que la librería RDKit ya tenía implementados tanto los métodos para calcular los *fingerprints* como el algoritmo de cálculo de distancia de tanimoto.

Finalmente, durante el desarrollo de lo métodos de predicción, se probaron diferentes algoritmos de entre los cuales destaca por su rápidez y precisión Naïve Bayes. Como ya se hizo en [28], se realizaron algunas pruebas con el algoritmo SVM. Este algoritmo alcanzaba un porcentaje de precisión bastante alto, equiparable al de Naïve Bayes o Random Forest, lamentablemente para ello era necesario emplear los métodos de ajuste de los hiperpárametros como GridSearchCV lo cual podía llegar a tardar más de 8 o 10 horas para el orden 10k o 20k compuestos y con una ganancia de entorno a un 5% más que el Random Forest (Todo ello realizando una estandarización de los hiperparámetros siguiendo la misma tónica que dicho *paper*). Por ello se decidió excluir del banco de pruebas el algoritmo SVM.

En definitiva, se han combinado varias tecnologías y disciplinas, como la gestión de bases de datos SQL y NoSQL, se ha trabajado con *toolkit* de quimioinformática y además de emplear *frameworks* para el desarrollo de modelos ML. Siendo todas estás herramientas de código abierto lo que permite a la comunidad aprender y nutrirse de su desarrollo.

Se han cumplido todas los objetivos planteados, por lo que se puede afirmar que el proyecto se ha llevado a término de forma satisfactoria.

6.1 Líneas de Trabajo Futuras

El desarrollo del proyecto ha establecido una base sobre la que desarrollar futuros proyectos. A continuación se exponen algunos planteamientos:

6.1.2 Importación de otras bases de datos

La base de datos creada a partir del dump de ChEMBL ha sido más que suficiente como prueba de concepto. No obstante, es posible adaptar y generalizar los métodos de importación a otras bases de datos de compuestos, especialmente al haberse empleado la base de datos NoSQL MongoDB. Ya que esto permite insertar nuevos registros sin necesidad de seguir un esquema concreto.

6.1.3 Interfaz web

Es posible integrar el código desarrollado con alguno de los frameworks de desarrollo web escritos en python, como Django , Flask o Pyramid. Su instalación como servicio web facilitaría su uso ya que elimina la necesidad de un despliegue masivo y se facilita a su vez a redundancia a través de conjuntos replicados o clusters mediante el proceso de sharding de MongoDB.

Por otro lado, es posible desarrollar métodos para comunicarse con otras plataformas bioinformáticas mediante interfaces estilo REST (Las cuales son bastante comunes actualmente) o similar.

6.1.4 Otras líneas de trabajo

Las principales implementaciones del interprete de python emplean el denominado Global Interpreter Lock o GIL. GIL es empleado para asegurarse que la ejecución del bytecode por el intérprete se restringe a un solo nucleo. Ello imposibilita en primera instancia la ejecución paralela de código. La principal forma recomendada por los desarrolladores de python es emplear la concurrencia a nivel de proceso.

Otra alternativa sería emplear módulos en C/C++ que se encargan de las tareas más intensivas y donde es mucho más sencillo el desarrollo de programs multihilo/multiproceso.

7 Anexos

7.1 Código SQL de preparación de vistas para importación de datos a MongoDB

```
DROP TABLE IF EXISTS COMPOUNDS_TARGETS;
CREATE TABLE COMPOUNDS_TARGETS AS
SELECT MOLREGNO, COMPOUND, TARGET
FROM (
            SELECT MD.MOLREGNO, MD.CHEMBL_ID as COMPOUND, TD.CHEMBL_ID as TARGET, COUNT(1) OVER (PARTITION BY TD.CHEMBL_ID) AS cnt
          FROM (((MOLECULE_DICTIONARY MD
           INNER JOIN ACTIVITIES AC ON (MD.MOLREGNO = AC.MOLREGNO AND PCHEMBL_VALUE >= 5))
                          INNER JOIN ASSAYS ASS ON (AC.ASSAY_ID = ASS.ASSAY_ID    AND ASS.ASSAY_ORGANISM='Homo sapiens'))
          INNER JOIN TARGET_DICTIONARY AS TD ON (ASS.TID = TD.TID))
      ) as v
      WHERE cnt > 20
      GROUP BY MOLREGNO, COMPOUND, TARGET;
            ALTER TABLE COMPOUNDS_TARGETS ADD CONSTRAINT PK_COMPOUNDS_TARGETS PRIMARY KEY (COMPOUND, TARGET);
       CREATE INDEX IDX_CT_MOLREGNO ON public.compounds_targets USING btree (molregno ASC NULLS LAST) TABLESPACE pg_default;
        CREATE INDEX IDX_CT_MOLREGNO_COMPOUND ON public.compounds_targets USING btree (molregno ASC NULLS LAST, COMPOUND ASC NULLS LAST) TABLESPACE pg_default;

      DROP TABLE IF EXISTS FILTERED_COMPOUND_PROPERTIES;

      CREATE TABLE FILTERED_COMPOUND_PROPERTIES AS
      SELECT COMPOUND, alogp, hba, hbd, psa, rtb, canonical_smiles
            FROM (SELECT DISTINCT MOLREGNO, COMPOUND FROM COMPOUNDS_TARGETS )  CT
              inner join compound_structures on (CT.MOLREGNO = compound_structures.molregno)
                                  inner join compound_properties on (compound_structures.molregno = compound_properties.molregno)
      where
      compound_properties.alogp IS NOT NULL
      AND compound_properties.hba IS NOT NULL
      AND compound_properties.hbd IS NOT NULL
      AND compound_properties.psa IS NOT NULL
      AND compound_properties.rtb IS NOT NULL
      AND compound_structures.canonical_smiles IS NOT NULL;

      COMMIT;
```

7.2 Código fuente de Importación y preparación de la base de datos

```
from pymongo import MongoClient
from .utils import formatRecord
from psycopg2 import connect
from pathlib import Path
from os.path import join, dirname
from pandas import read_sql

class DBPopulator:

    def __init__(self):
        self.client = MongoClient()
        self.mongoDB = self.client['DrugDealer']
        #self.SQLiteDB = connect(str(Path(join(dirname(__file__),
        #    "../DBs/chembl_22_1.db")).resolve()))
             self.con  =  connect(dbname="chembl_23",  user="postgres",
password="postgres")
        self.chunksize = 1000
        self.preparing_tables_query = """
        DROP TABLE IF EXISTS COMPOUNDS_TARGETS;
        CREATE TABLE COMPOUNDS_TARGETS AS
        SELECT MOLREGNO, COMPOUND, TARGET
        FROM (
            SELECT MD.MOLREGNO, MD.CHEMBL_ID as COMPOUND, TD.CHEMBL_ID
as TARGET, COUNT(1) OVER (PARTITION BY TD.CHEMBL_ID) AS cnt
            FROM (((MOLECULE_DICTIONARY MD
            INNER JOIN ACTIVITIES AC ON (MD.MOLREGNO = AC.MOLREGNO AND
PCHEMBL_VALUE >= 5))
             INNER JOIN ASSAYS ASS ON (AC.ASSAY_ID = ASS.ASSAY_ID  AND
ASS.ASSAY_ORGANISM='Homo sapiens'))
            INNER JOIN TARGET_DICTIONARY AS TD ON (ASS.TID = TD.TID))
        ) as v
        WHERE cnt > 20
        GROUP BY MOLREGNO, COMPOUND, TARGET;
                    ALTER   TABLE   COMPOUNDS_TARGETS   ADD   CONSTRAINT
PK_COMPOUNDS_TARGETS PRIMARY KEY (COMPOUND, TARGET);
        CREATE INDEX IDX_CT_MOLREGNO ON public.compounds_targets USING
btree (molregno ASC NULLS LAST) TABLESPACE pg_default;
                        CREATE   INDEX   IDX_CT_MOLREGNO_COMPOUND   ON
public.compounds_targets  USING  btree  (molregno  ASC  NULLS  LAST,
COMPOUND ASC NULLS LAST) TABLESPACE pg_default;

        DROP TABLE IF EXISTS FILTERED_COMPOUND_PROPERTIES;

        CREATE TABLE FILTERED_COMPOUND_PROPERTIES AS
        SELECT COMPOUND, alogp, hba, hbd, psa, rtb, canonical_smiles
                      FROM (SELECT DISTINCT MOLREGNO, COMPOUND FROM
COMPOUNDS_TARGETS )  CT
                     inner join compound_structures on (CT.MOLREGNO =
compound_structures.molregno)
                                  inner  join  compound_properties  on
(compound_structures.molregno = compound_properties.molregno)
        where
        compound_properties.alogp IS NOT NULL
        AND compound_properties.hba IS NOT NULL
        AND compound_properties.hbd IS NOT NULL
        AND compound_properties.psa IS NOT NULL
        AND compound_properties.rtb IS NOT NULL
```

```
        AND compound_structures.canonical_smiles IS NOT NULL;

        COMMIT;
        """


         self.targets_query = """ SELECT TARGET FROM COMPOUNDS_TARGETS
WHERE COMPOUND = '%s' """

                self.compound_properties_query  =  """  SELECT  *  FROM
FILTERED_COMPOUND_PROPERTIES """

    def targets_from_compound(self, compound_id):
             return  (read_sql(sql=self.targets_query  %  (compound_id),
con=self.con)["target"].values.tolist())

    def prepareTables(self):
        self.cur = self.con.cursor()
        self.cur.execute(self.preparing_tables_query)

    def populateDatabase(self):
        documents = []

        i = 0
        for results in read_sql(sql=self.compound_properties_query,
                                con=self.con,
                                chunksize=self.chunksize):
            try:
                for _, row in results.iterrows():
                                                              targets   =
self.targets_from_compound(row['compound'])
                    record = formatRecord(row, targets)
                    documents.append(record)

                self.mongoDB['compounds'].insert_many(documents)
                del documents
                documents = []
                print("Conjunto"+ str(i) + "volcado en mongoDB" )
                i += 1
            except:
                pass

        if documents:
            self.mongoDB['compounds'].insert_many(documents)
```

7.2.2 Tests funcionales preparación de la base de datos

```
from sqlite3 import connect
from pathlib import Path
from pandas import read_sql
from drugdealer.utils import formatRecord
from drugdealer.DBPopulator import DBPopulator
from os.path import join, dirname
from pickle import load


def test_formatRecord():
    pathToDB = str(Path(join(dirname(__file__),
                             "../DBs/chembl_22_1.db")).resolve())
    pathToSampleMongoRecord = str(Path(join(dirname(__file__),
                                            "mock/sampleMongoRecord.bson"))
                                  .resolve())

    con = connect(pathToDB)

    compound_query = \
        """ SELECT DISTINCT chembl_id, entity_type, alogp,
                            hba, hbd, psa, rtb, canonical_smiles
            FROM chembl_id_lookup INNER JOIN compound_structures
            ON (entity_type='COMPOUND'
            AND entity_id = compound_structures.molregno)
            INNER JOIN compound_properties
            ON ( compound_structures.molregno = compound_properties.molregno
            AND compound_properties.alogp IS NOT NULL
            AND compound_properties.hba IS NOT NULL
            AND compound_properties.hbd IS NOT NULL
            AND compound_properties.psa IS NOT NULL
            AND compound_properties.rtb IS NOT NULL) LIMIT 1
        """

    targets_query = \
        """ SELECT distinct MD.CHEMBL_ID AS COMPOUND, TD.CHEMBL_ID as TARGET
            FROM ((((COMPOUND_STRUCTURES CS
                        INNER JOIN MOLECULE_DICTIONARY MD ON (CS.MOLREGNO =
MD.MOLREGNO))
            INNER JOIN ACTIVITIES AC ON (MD.MOLREGNO = AC.MOLREGNO))
            INNER JOIN ASSAYS ASS ON (AC.ASSAY_ID = ASS.ASSAY_ID))
                  INNER JOIN TARGET_DICTIONARY TD ON (ASS.TID = TD.TID)) WHERE
MD.CHEMBL_ID = \"%s\"
        """

    compound_query_result = read_sql(sql=compound_query, con=con)
    chembl_id = compound_query_result['chembl_id'][0]
              targets = read_sql(sql=targets_query % (chembl_id), con=con)
["TARGET"].values.tolist()

    sampleMongoRecordFile = open(pathToSampleMongoRecord, "rb")
    mongoRecord = load(sampleMongoRecordFile)
    sampleMongoRecordFile.close()


          assert formatRecord(compound_query_result.iloc[0,:].to_dict(), targets) ==
mongoRecord
```

```
def test_targets_from_compound():
    db_populator = DBPopulator()
    compound_id = "CHEMBL1"
    pathToSampleTargetsList = str(Path(join(dirname(__file__),
                                      "mock/sampleTargetsList"))
                                .resolve())

    SampleTargetsListFile = open(pathToSampleTargetsList, "rb")
    targets_list = load(SampleTargetsListFile)
    SampleTargetsListFile.close()

    assert db_populator.targets_from_compound(compound_id) == targets_list
```

7.3 Demo Cálculo de similitud

```
# -*- coding: utf-8 -*-
from fingerprinter import FingerPrinter
from computesimilarity import ComputeSimilarity
from pymongo import MongoClient
from pickle import loads

def demo_calculo_similitud():

    db = MongoClient()['DrugDealer']["compounds"]
    threshold = 0.80
    similarity = ComputeSimilarity()
    sample_molecule = db.find().limit(2444)[12]
    print ("Compuesto de muestra: ",sample_molecule["CHEMBL_ID"])
        sample_molecule_fingerprint  =  sample_molecule["FingerPrints"]["AtomPairs"]
["data"]
    for molecule in db.find():
        try:
                                                                                tanimoto        =
similarity.getTanimotoSimilarity(loads(sample_molecule_fingerprint),
loads(molecule["FingerPrints"]["AtomPairs"]["data"]))
            if tanimoto > threshold:
                print ('%s : %s' % (tanimoto, molecule['CHEMBL_ID']))
        except:
            pass

"""Main module."""
if __name__ == '__main__':
    demo_calculo_similitud()
```

7.4 Cálculo para la similitud

```
from rdkit.DataStructs import TanimotoSimilarity

class ComputeSimilarity:

    def __init__(self):
        pass

    def getTanimotoSimilarity(self, fingerprint1, fingerprint2):
        return TanimotoSimilarity(fingerprint1, fingerprint2)
```

7.4.2 Tests funcionales funciones fingerprinting

```
from drugdealer.fingerprinter import FingerPrinter
from rdkit.Chem import MolFromSmiles
from pathlib import Path
from os.path import join, dirname
import pickle


class TestFingerprinter:

    def test_getTopologicalFingerPrint(self):
        fingerprinter = FingerPrinter()
        sampleSMILE = "Cc1cc(ccc1C(=O)c2ccccc2Cl)N3N=CC(=O)NC3=O"

        topologicalFile = open(str(Path(
                              join(
                                 dirname(__file__),
                                 "mock/TopologicalFingerPrint.pickle"))
                              .resolve()), "rb")
        topologicalFingerPrint = pickle.load(file=topologicalFile)
        topologicalFile.close()

        fingerprint = fingerprinter.getTopologicalFingerPrint(
            mol=MolFromSmiles(sampleSMILE))
        assert fingerprint == topologicalFingerPrint

    def test_getMorganFingerPrint(self):
        fingerprinter = FingerPrinter()
        sampleSMILE = "Cc1cc(ccc1C(=O)c2ccccc2Cl)N3N=CC(=O)NC3=O"

        morganFile = open(str(Path(
                           join(
                              dirname(__file__),
                              "mock/MorganFingerPrint.pickle"))
                           .resolve()), "rb")
```

```
    morganFingerPrint = pickle.load(file=morganFile)
    morganFile.close()

    fingerprint = fingerprinter.getMorganFingerPrint(
        mol=MolFromSmiles(sampleSMILE))
    assert fingerprint == morganFingerPrint

def test_getAtomPairsFingerPrint(self):
    fingerprinter = FingerPrinter()
    sampleSMILE = "Cc1cc(ccc1C(=O)c2ccccc2Cl)N3N=CC(=O)NC3=O"

    AtomPairsFile = open(str(Path(
                        join(
                            dirname(__file__),
                            "mock/AtomPairsFingerPrint.pickle"))
                        .resolve()), "rb")
    atomPairsFingerPrint = pickle.load(file=AtomPairsFile)
    AtomPairsFile.close()

    fingerprint = fingerprinter.getAtomPairsFingerPrint(
        mol=MolFromSmiles(sampleSMILE))
    assert fingerprint == atomPairsFingerPrint
```

7.5 Tests funcionales funciones cálculo de similitud

```
import pickle
from drugdealer.computesimilarity import ComputeSimilarity
#sys.path.insert(1, os.path.join(sys.path[0], '..'))
from pathlib import Path
from os.path import join, dirname


class TestSimilarity:


    def test_getTanimotoSimilarity(self):
        self.ComputeSimilarity = ComputeSimilarity()
        self.sampleSMILE1 = "Cc1cc(ccc1C(=O)c2ccccc2Cl)N3N=CC(=O)NC3=O"
         self.sampleSMILE2 = "BP(=O)(OP(=O)(O)OP(=O)(O)OC[C@H]1O[C@H]([C@H]
(O)[C@@H]1O)N2C=CC(=O)NC2=O)OP(=O)(O)OP(=O)(O)OC[C@H]3O[C@H]([C@H]
(O)
[C@@H]3O)N4C=CC(=O)NC4=O.CCCCN(CCCC)CCCC.CCCCN(CCCC)CCCC.CCCCN(C
CCC)CCCC.CCCCN(CCCC)CCCC.CCCCN(CCCC)CCCC"
        fingerprint1File = open(str(Path(join(dirname(__file__),
                          "mock/MorganFingerPrint1.pickle"))
                          .resolve()), "rb")
        fingerprint2File = open(str(Path(join(dirname(__file__),
                         "mock/MorganFingerPrint2.pickle"))
                         .resolve()), "rb")


        self.fingerprint1 = pickle.load(file=fingerprint1File)
        self.fingerprint2 = pickle.load(file=fingerprint2File)
        fingerprint1File.close()
        fingerprint2File.close()
        self.tanimotoSimilarity =  0.0601092
                                                                    similarity          =
self.ComputeSimilarity.getTanimotoSimilarity(self.fingerprint1,self.fingerprint2)
        assert abs(similarity - self.tanimotoSimilarity) < 0.0001
```

7.6 Codigo de predicción de Targets mediante Machine Learning

```
from sklearn import metrics
from sklearn.preprocessing import Normalizer
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from operator import itemgetter

from sklearn import preprocessing
from sklearn.pipeline import Pipeline
import numpy as np

from scipy.stats import randint as sp_randint
from time import time

from pymongo import MongoClient
mongoDB = MongoClient()["DrugDealer"]

def prepare_data(Target):

    cursor_actives = mongoDB["compounds"].find({"Targets": Target})
    actives_count = cursor_actives.count()
            cursor_decoys = mongoDB["compounds"].find({"Targets": {"$nin":
[Target]}}).limit(actives_count*1)

    dataset = []

    for active in cursor_actives:
        item = [np.int(1), list(active["Properties"].values())]
        dataset.append(item)


    for decoy in cursor_decoys:
        item = [np.int(0), list(decoy["Properties"].values())]
        dataset.append(item)


    np.random.shuffle(dataset)

    type_test  = [elem[0] for elem in dataset[0:np.int(len(dataset) * 0.2)]]
    data_test  = [elem[1] for elem in dataset[0:np.int(len(dataset) * 0.2)]]
    type_train = [elem[0] for elem in dataset[np.int(len(dataset) * 0.2):]]
    data_train = [elem[1] for elem in dataset[np.int(len(dataset) * 0.2):]]
```

```
    return data_train, data_test, type_train, type_test


def model_fit(model, data, target):
    return model.fit(data, target)

def model_predict(fitted_model, test_data):
   return fitted_model.predict(test_data)

def print_prediction_results(original, predicted):
    print(metrics.classification_report(original, predicted))
    print(metrics.confusion_matrix(original, predicted))

def random_forest_classification_RSCV(data_train, data_test, type_train,
type_test):
    #Random forest
    param_dist = {"max_depth": [None],
          "max_features": ['log2', None],
          "bootstrap": [True, False],
          "criterion": ["gini", "entropy"]}

    n_iter_search = 5
                                                    random_search =
RandomizedSearchCV(RandomForestClassifier(n_estimators=100),
param_distributions=param_dist,
                                n_iter=n_iter_search, n_jobs=5)

    start = time()
    random_search.fit(data_test, type_test)

    #print("RandomizedSearchCV took %.2f seconds for %d candidates"
    #      " parameter settings." % ((time() - start), n_iter_search))


                                                        fitted_model =
model_fit(RandomForestClassifier(**random_search.best_params_,
n_estimators=888), data_train, type_train)
    predicted = model_predict(fitted_model, data_test)

    return predicted

def random_forest_classification_GSCV(data_train, data_test, type_train,
type_test):
    #Random forest
    param_dist = {"max_depth": [None],
                "max_features": ['log2', None],
                "bootstrap": [True, False],
                "criterion": ["gini", "entropy"]}
```

```
    RF_pipeline = Pipeline([('clf', RandomForestClassifier(n_estimators=10))])

    param_grid = [{'clf__n_estimators': [10,20],
                   'clf__max_features':[3,4,5],
                   'clf__bootstrap': [True, False],
                   'clf__criterion': ["gini", "entropy"]
                  }]

                            grid_search = GridSearchCV(estimator=RF_pipeline,
param_grid=param_grid,
                             scoring='f1_micro', n_jobs=5, cv=3)

    grid_search_fitted = grid_search.fit(data_train, type_train)



    best_params = dict()
    for key, value in grid_search_fitted.best_params_.items():
        best_params[key[5:]] = value

    RF_model = RandomForestClassifier(**best_params)

    fitted_model = model_fit(RF_model, data_train, type_train)

    predicted = model_predict(fitted_model, data_test)

    return predicted

def ANN_classification(data_train, data_test, type_train, type_test):

    param_grid = {"alpha": [.1, .01, .001],
                  "momentum": [.7, .9],
                  "learning_rate_init": [.1, .01, .001]
                 }

                                    mlp= MLPClassifier(hidden_layer_sizes=(50,),
max_iter=10000,batch_size=600,
                                                                    algorithm='sgd',
random_state=1,activation="logistic",learning_rate="constant")
            random_search = RandomizedSearchCV(mlp, param_distributions=
param_grid,n_iter = 8)
    fitted_model = model_fit(random_search, data_train, type_train)
    predicted = model_predict(fitted_model, data_test)

    return predicted

def GaussNB_classification(data_train, data_test, type_train, type_test):
```

```
    fitted_model = model_fit(GaussianNB(), data_train, type_train)
    predicted = model_predict(fitted_model, data_test)

    return predicted

def logistic_regression_classification(data_train, data_test, type_train, type_test):

    param_grid = {'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000],
                  'solver': ['newton-cg', 'lbfgs'],
                  'max_iter': [100, 250, 1000, 4000]
                 }
                  random_search = RandomizedSearchCV(LogisticRegression(), param_distributions=param_grid, n_jobs=5)

    #start = time()
    fitted_model = model_fit(random_search, data_train, type_train)

    #print("RandomizedSearchCV took %.2f seconds"
    #      " parameter settings." % ((time() - start)))


                                                        fitted_model = model_fit(LogisticRegression(**random_search.best_params_, n_jobs=5), data_train, type_train)
    predicted = model_predict(fitted_model, data_test)
    return predicted

def test(Target):
    print("Target a probar: ", Target, "\n")

    results_f1_score = []
    results_accuracy_score = []

    cursor_actives = mongoDB["compounds"].find({"Targets": Target})
    actives_count = cursor_actives.count()

    print("Número de componentes activos: ", actives_count)

    print("Clasificador Naïve Bayes\n")
    for i in range(0, 10):
        data_train, data_test, type_train, type_test = prepare_data(Target)
         predicted_NB = GaussNB_classification(data_train, data_test, type_train, type_test)
                  results_accuracy_score.append(metrics.accuracy_score(type_test, predicted_NB))
        results_f1_score.append(metrics.f1_score(type_test, predicted_NB))
```

```
    print("Precisión: ", np.mean(results_accuracy_score))
    print("Puntuación F1: ", np.mean(results_f1_score))

    print("\nClasificador Redes Neuronales Artificiales\n")

    for i in range(0, 10):
        data_train, data_test, type_train, type_test = prepare_data(Target)
            predicted_ANN = ANN_classification(data_train, data_test, type_train,
type_test)
                results_accuracy_score.append(metrics.accuracy_score(type_test,
predicted_ANN))
        results_f1_score.append(metrics.f1_score(type_test, predicted_ANN))


    print("Precisión: ", np.mean(results_accuracy_score))
    print("Puntuación F1: ", np.mean(results_f1_score))

    print("\nClasificador Regresión Logística\n")

    for i in range(0, 10):
        data_train, data_test, type_train, type_test = prepare_data(Target)
            predicted_LR = logistic_regression_classification(data_train, data_test,
type_train, type_test)
                results_accuracy_score.append(metrics.accuracy_score(type_test,
predicted_LR))
        results_f1_score.append(metrics.f1_score(type_test, predicted_LR))


    print("Precisión: ", np.mean(results_accuracy_score))
    print("Puntuación F1: ", np.mean(results_f1_score))


    print("\nClasificador Random Forest\n")

    for i in range(0, 10):
        data_train, data_test, type_train, type_test = prepare_data(Target)
            predicted_RF_RSCV = random_forest_classification_RSCV(data_train,
data_test, type_train, type_test)
                results_accuracy_score.append(metrics.accuracy_score(type_test,
predicted_RF_RSCV))
                                results_f1_score.append(metrics.f1_score(type_test,
predicted_RF_RSCV))
        print(i,"/10")

    print("Precisión: ", np.mean(results_accuracy_score))
    print("Puntuación F1: ", np.mean(results_f1_score))
```

7.7 Demo Machine Learning

```
import ML

def demo_machine_learning():

    ML.test("CHEMBL612545")
    ML.test("CHEMBL384")

    ML.test("CHEMBL2366230")


if __name__ == '__main__':
  demo_machine_learning()
```

8 Bibliografía

[1] “Slide 1.” https://www.ebi.ac.uk/sites/ebi.ac.uk/files/content.ebi.ac.uk/materials/2013/

[2] Wikipedia, “Virtual screening – Wikipedia, the free encyclopedia.” http://en.wikipedia.org/w/index.php?title=Virtual%20screening&oldid=764822281

[3] K. Heikamp and J. Bajorath, “Support vector machines for drug discovery.,” Expert Opin Drug Discov, vol. 9, pp. 93–104, Jan. 2014.

[4] V. V. Zernov, K. V. Balakin, A. A. Ivaschenko, N. P. Savchuk, and I. V. Pletnev, “Drug discovery using support vector machines. the case studies of drug-likeness, agrochemical-likeness, and enzyme inhibition predictions,” Journal of Chemical Information and Computer Sciences, vol. 43, no. 6, pp. 2048–2056, 2003. PMID: 14632457.

[5] “Redes neuronales.” http://avellano.usal.es/~lalonso/RNA/

[6] “Random forest – wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Random_forest

[7] “Clasificador bayesiano ingenuo – wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/Clasificador_bayesiano_ingenuo

[8] “Apache couchdb.” http://couchdb.apache.org/

[9] “Apache cassandra.” http://cassandra.apache.org/

[10] “Reinventando la gestión de datos — mongodb.” https://www.mongodb.com/es

[11] Wikipedia, “Test-driven development — Wikipedia, the free encyclopedia.” http://en.wikipedia.org/w/index.php?title=Test-driven%20development&oldid=768863944

[12] “Arch linux.” https://www.archlinux.org/

[13] “Texmaker (free cross-platform latex editor).” http://www.xm1math.net/texmaker/

[14] “Tex live - tex users group.” https://www.tug.org/texlive/

[15] “Pymongo documentation” https://api.mongodb.com/python/current/

[16] “Python data analysis library — pandas: Python data analysis library.” http://pandas.pydata.org/

[17] “Rdkit.” http://www.rdkit.org/

[18] “scikit-learn: machine learning in python — scikit-learn 0.18.1 documentation.” http://scikit-learn.org/stable/

[19] “Unittest — unit testing framework — python 3.x documentation.” https://docs.python.org/3/library/unittest.html

[20] “The world’s leading software development platform, github.” https://github.com/

[21] “Bitbucket — the git solution for professional teams.” https://bitbucket.org/

[22] “Structure-activity relationship - wikipedia.”
https://en.wikipedia.org/wiki/Structure%E2%80%93activity_relationship

[23] “Np-hard - wikipedia, la enciclopedia libre.” https://es.wikipedia.org/wiki/NP-hard

[24] “Transferencia de estado representacional - wikipedia, la enciclopedia libre.”
https://es.wikipedia.org/wiki/Transferencia_de_Estado_Representacional

[25] P. Willett, “Similarity-based virtual screening using 2D fingerprints,” Drug Discov. Today, vol. 11, pp. 1046–1053, Dec 2006.

[26] D. Bajusz, A. Racz, and K. Heberger, “Why is Tanimoto index an appropriate choice
for fingerprint-based similarity calculations?,” J Cheminform, vol. 7, p. 20, 2015.

[27] “Errores detipo I y de tipo II – wikipedia, la enciclopedia libre.”
https://es.wikipedia.org/wiki/Errores_de_tipo_I_y_de_tipo_II

[28] Memoria TFM Raul González Balea

[29] https://quantdare.com/random-forest-vs-simple-tree/

[30] https://github.com/akash13singh/DataMining/blob/master/nnet.py

[31] http://www.rdkit.org/UGM/2012/Landrum_RDKit_UGM.Fingerprints.Final.pptx.pdf

[32] https://www.quora.com/Will-Naive-Bayes-give-a-good-result-if-the-training-set-is-much-smaller-than-the-no-of-features-in-a-machine-learning-classifier