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

Mikel Martínez Goikoetxea
Máster en Bioinformática y Bioestadística
Bioinformática farmacéutica

Melchor Sanchez Martínez
María Jesús Marco Galindo

24/05/2017

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Creación de una herramienta de software para predecir la interacción, actividad y función de fármacos*
Nombre del autor:	*Mikel Martínez Goikoetxea*
Nombre del consultor/a:	*Melchor Sanchez Martínez*
Nombre del PRA:	*María Jesús Marco Galindo*
Fecha de entrega (mm/aaaa):	05/2017
Titulación::	*Máster de Bioinformática y Bioestadística*
Área del Trabajo Final:	*Bioinformática farmacéutica*
Idioma del trabajo:	*Castellano*
Palabras clave	*Quimioinformática, similaridad química, machine learning*

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

Gracias al desarrollo de librerías de *software* de cálculo científico así como al abaratamiento de la potencia computacional, hoy en día es relativamente sencillo para el científico motivado aprovechar el gran poder que representan técnicas computacionales avanzadas como son las búsquedas por similitud química o el *machine learning*.

Este proyecto pretende desarrollar herramientas de *software* que puedan ser útiles en la fase de *drug discovery* en el contexto del desarrollo de fármacos. Para alcanzar este fin, se han seguido dos aproximaciones distintas:

En la primera, se ha tratado de crear una herramienta que permita predecir el efecto y la diana molecular de un fármaco en base a su similitud química con otros fármacos, cuyo efecto y diana son conocidos. Esta similitud química ha sido determinada mediante el cálculo de la distancia de Tanimoto.

En la segunda parte, se ha tratado de crear una herramienta que permita predecir si un fármaco se une o no a una diana molecular concreta a partir de la creación de un modelo utilizando un algoritmo de *machine learning* de tipo *support vector machine*.

Cada una de las aproximaciones ha resultado en dos programas independientes, que ejecutan todos los pasos necesarios para obtener dichas predicciones. En este documento se discuten los métodos empleados y hasta qué punto son exitosos.

Abstract (in English, 250 words or less):

Thanks to the development of software libraries for scientific calculations, as well as to the decrease in computational power cost, nowadays it is easy for the motivated scientist to make use of the great power that advanced computational techniques such as chemical similarity searches and machine learning algorithms represent.

In this project, software tools that could be useful in the drug discovery step (in the context of drug development) are developed. To achieve this, two approaches have been taken.

In the first one, a tool has been developed that allows the prediction of the effect and molecular target of a drug, by means of its chemical similarity to another drug of known effect and/or target.

In the second approach, a tool has been developed that allows to predict whether a drug interacts or not with a specific molecular target, by means of the creation of model with a machine learning algorithm, the support vector machine.

Each of the approaches have resulted in two independent programs, that run all the necessary steps required to perform the said predictions. In this document, the used methods are discussed, as well as their success.

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

La industria farmacéutica es seguramente uno de los campos científicos que más repercusión tiene en la vida humana, tanto a nivel de calidad de vida como en términos de actividad económica. Es de sobra conocido que el desarrollo de nuevos fármacos es un proceso laborioso, largo, y económicamente exigente, además de que presenta un rendimiento considerablemente bajo (sólo una fracción de los fármacos potenciales llegan a alcanzar el nivel de desarrollo de ser patentables).

Una parte de esto no es evitable, ya que tiene que ver con los análisis de seguridad y viabilidad que deben realizarse, de manera forzosa, en varias etapas y con un número considerable de pacientes. No obstante, hay varios puntos críticos del proceso que pueden ser optimizados mediante herramientas bioinformáticas. En concreto, en este proyecto se trata de crear herramientas de software que puedan ser utilizadas en una de las fases más tempranas del desarrollo de nuevos fármacos, la fase de *drug discovery*.

Durante esta fase, se determina qué compuestos químicos tienen cierto potencial de ser investigados más en profundidad con el objetivo de obtener un fármaco funcional. La herramienta más utilizada en este paso es el *screening* de alto rendimiento, con el que se comprueba si una librería de compuestos interaccionan y/o se unen a una diana molecular concreta. En este proyecto se pretende desarrollar herramientas que permitan realizar este proceso de forma completamente computacional. Como paso previo a fin de eliminar compuestos candidatos, o como una prueba adicional, estas herramientas podrían ser útiles en el contexto del descubrimiento de nuevos fármacos.

Gracias al desarrollo de librerías de cálculo científico así como al abaratamiento de la potencia computacional, hoy en día es relativamente sencillo para el científico motivado aprovechar el gran poder que representan técnicas computacionales avanzadas como son las búsquedas por similitud química o el *machine learning*. Este proyecto se ha dividido en dos partes: en la primera, se ha tratado de crear una herramienta que permita predecir el efecto y la diana molecular de un fármaco en base a su similitud química con otros fármacos, cuyo efecto y diana son conocidos; en la segunda parte, se ha tratado de crear una herramienta que permita predecir si un fármaco se une o no a una diana molecular concreta a partir de la creación de un modelo utilizando un algoritmo de *machine learning*.

1.2 Objetivos del Trabajo

En este proyecto, destacan dos objetivos generales, que corresponden con las dos partes del proyecto resumidas en el apartado 1.1. Estos objetivos generales son

1) Implementar una herramienta de *software* capaz de predecir, a partir de la estructura química de una molécula problema, sus dianas y actividades potenciales. La herramienta estará basada en la búsqueda por similitud química, ya que buscará moléculas para las cuales se conoce información sobre efecto y/o diana molecular que sean químicamente similares a la molécula problema.

2) Implementar una herramienta de *software* capaz de predecir, a partir de la estructura química de una molécula problema, si esta interacciona o no con una diana molecular específica. La herramienta estará basada en *machine learning*, por lo que antes de realizar predicciones, debe entrenarse un modelo (que será específico de esa diana molecular).

Por otra parte, cada objetivo general puede ser desgranado en varios objetivos específicos, los cuales a su vez serán divididos en tareas básicas en un apartado próximo.

Objetivos específicos del objetivo general 1:

1. Crear un set de datos representativo de fármacos, que incluya sus actividades y moléculas diana. Este objetivo incluye elegir una base de datos de compuestos.

2. Implementar un programa de búsqueda por similaridad de fármacos (utilizando la librería Rdkit de Python), que permita predecir la actividad y diana de un fármaco en base a moléculas conocidas parecidas

3. Obtener una valoración del predictor

Objetivos específicos del objetivo general 2:

1. Crear un set de datos representativo de fármacos y *decoys* para un *target* específico. El algoritmo de *machine learning* que se va a utilizar será de aprendizaje supervisado, es decir, que el *input* del algoritmo serán ejemplos de fármacos que se unen al *target* (ejemplos positivos) y fármacos que no se unen (ejemplos negativos).

2. Implementar un programa de *machine learning* que a partir del set de datos anterior cree un modelo capaz de predecir si un fármaco nuevo interaccionará con la diana específica con la que se ha creado el modelo

3. Obtener una valoración del predictor

4. Utilizar el programa creado con varias dianas distintas

1.3 Enfoque y método seguido

En cuanto a la parte del proyecto relacionada con la búsqueda por similitud, comenzar destacando que esta aproximación se basa completamente en la premisa 'una estructura similar sugiere una acción similar', que tan conocida es en el ámbito de la química farmacéutica [1]. Por este principio, se pretende poder predecir los efectos y dianas moleculares de un fármaco a través de la búsqueda de fármacos similares cuyos efectos y dianas sean conocidos. Para realizar esta tarea, nos basaremos en las rutinas quimioinformáticas implementadas en Rdkit, un modulo del lenguaje de programación Python [2].

Respecto a la parte de *machine learning*, también se hará uso de Rdkit para la obtención de descriptores moleculares, si bien será igualmente importante el uso de *machine learning*. Este será facilitado por el módulo scikit-learn de Python [3].

1.4 Planificación del Trabajo

Los recursos necesarios para llevar a cabo este proyecto son mínimos, ya que es completamente computacional y no se requiere de capacidad de procesamiento particularmente potente. No obstante, se hará inventario de los programas de *software* que serán utilizados y para qué fin.

- Sistema Operativo tipo Linux (Manjaro): Desarrollo del proyecto
  - Interprete de Python
  - Jupyter *notebook*: Entorno de desarrollo del proyecto, provee de funciones de autocompletado y ejecución de fragmentos de código.
  - Paquetes de Python:
    - Pandas: Leer y escribir archivos de datos, estructura de tipo *dataframe* similar a la que tiene R de forma nativa
    - Numpy: Estructura de datos de tipo matriz
    - Rdkit: Funciones de quimioinformática
    - Scikit-learn: Funciones de *machine learning*
    - Matplotlib: Visualización de datos, gráficos
  - Sqlite: Bases de datos
  - Gimp: Creación de gráficos
  - Gantt: Diagramas de Gantt y organización

- Sistema Operativo Windows 7: Desarrollo de la memoria y la presentación
    - Microsoft word 2013:
    - Microsoft powerpoint 2013:

En la planificación que sigue se desarrollan los objetivos generales, descomponiéndolos en tareas más simples.

Tareas generales:

- Estudiar y aprender herramientas y fundamentos necesarios para el desarrollo del proyecto. Esto incluye:
    - Manejo de Python para datos científicos: Principalmente, cargar datos, hacer gráficos, utilizar un entorno de desarrollo interactivo. Este apartado incluye las librerías Numpy, Matplotlib, Pandas, y el Jupyter notebook (entorno de desarrollo, antes llamado Ipython notebook).
    - Manejo de Python para tareas específicas: Búsqueda por similaridad, *machine learning*. Este apartado incluye las librería Rdkit (quimioinformática) y Scikit-learn (*machine learning*).
    - Bases de datos de fármacos: Estudiar cuales hay disponibles, sus diferencias, formatos más habituales, etc
    - Teoría de búsqueda por similaridad: Qué medidas de distancia existen, cual se debería utilizar, etc.
    - (Teoría de *machine learning*): Este apartado sería necesario de no ser porque se ha cursado una asignatura homónima, y por lo tanto se considera que se conocen los fundamentos básicos de cómo aplicar este tipo de algoritmos.

Tareas relacionadas con el objetivo general 1

- Elección de base de datos: Se han considerado las principales bases de datos con información relevante sobre compuestos químicos/dianas/efectos, tales como BioGrid, STITCH, Matador, DrugBank y ChEMBL. En la selección de qué base de datos utilizada, se han valorardo características como sencillez de uso, lo completas que son, en qué formato están disponibles, y de qué formas pueden descargarse.

- Obtención de base de datos: Mediante *scripts* se han descargado los datos relevantes de la base de datos de elección, en formato *db*.

- Procesamiento de base de datos: A fin de obtener archivos de texto legibles, deben extraerse las tablas del archivo de extensión *db* obtenido en la tarea anterior.

- Construir set de datos que relacione fármacos con *targets* y efectos.

- Implementar búsqueda por similitud química para predecir efecto y *target* de compuestos problema.

- Valorar predictor: Para valorar este predictor, se ha utilizado la aproximación *leave-one-out*, que consiste en coger compuestos del set de datos, e intentar predecir su efecto y diana quitando dicho compuesto del set de datos.

Tareas relacionadas con el objetivo general 2

- Ampliar el set de datos obtenido en el objetivo general 1 con los *decoys* específicos de cada *target* del cual se vaya a hacer un modelo de predicción de interacción.

- Construir set de datos con fármacos que interaccionan con la diana elegida, así como con *decoys.*

- Implementar un script en Python que mediante *machine learning* produzca un modelo capaz de predecir si una diana interacciona o no con un compuesto a partir de su estructura química.

- Valorar predictor con los estimadores y gráficos adecuados: Para esto, se han realizado predicciones con compuestos no utilizados en la creación del modelo, y ya que se conocen sus dianas reales, se ha valorado lo buena que es la predicción. Los métricos utilizados son matrices de confusión y curvas ROC.

- Optimización de parámetros: En el caso de esta aproximación mediante *machine learning*, este paso es una parte integral de la implementación del programa, y no un paso de 'mejora' posterior.

En la planificación inicial se definieron una serie de hitos que han marcado el progreso del proyecto

- Crear el set de datos (para predicción basada en búsqueda de similaridad).

- Primera predicción de diana y efecto de compuesto no presente en set de datos (método de similaridad).

- Valoración de capacidad predictiva de búsqueda por similaridad.

- Crear segundo set de datos (para predicción basada en *machine learning*).

- Primera predicción con compuesto no presente en set de datos (método *machine learning*).

- Valoración de capacidad predictiva de aproximación con *machine learning.*

Al comienzo del trabajo se proyectó el progreso esperado en un diagrama de Gantt. Este plan ha sido corregido y/o ajustado de acuerdo con los objetivos que se han ido cumpliendo en cada una de las PEC, como se muestra a continuación.

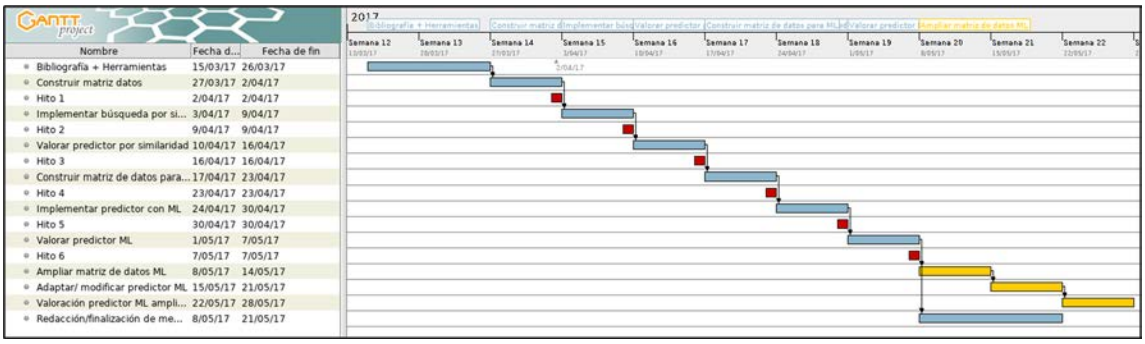


Ilustración 1: Diagrama de Gantt en la PEC1

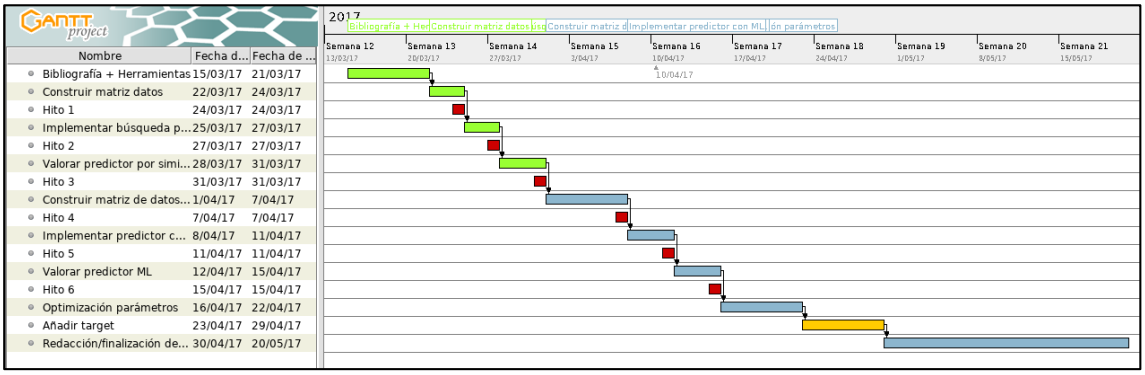


Ilustración 2: Diagrama de Gantt en la PEC2

Ilustración 3: Diagrama de Gantt en la PEC3

En general se ha seguido la temporalización propuesta inicialmente. Cabe destacar que la primera temporalización fue diseñada de manera bastante 'flexible', ya que había varios elementos del proyecto que era difícil predecir cuanto tiempo iban a llevar. Es por esto que se diseñó un plan realizable en el plazo indicado, si bien se contaba con poder adelantar trabajo respecto a ese calendario, e introducir las tareas consideradas inicialmente 'optativas' en el proyecto.

Por otra parte, los objetivos y tareas específicas del proyecto han sido adaptadas en la medida en la que se ha ido conociendo y estudiando mejor las herramientas con las que se ha realizado el trabajo. Los cambios de planteamiento ligados a limitaciones de las aproximaciones utilizadas serán comentados con más detalles en próximas secciones.

1.5 Breve sumario de productos obtenidos

Concluido el proyecto, esta es la lista de productos que se han obtenido, clasificados por a qué parte del proyecto pertenecen, así como una breve descripción de cada uno

Respecto de la primera parte del proyecto, que consiste en la predicción de diana y efecto mediante búsqueda por similaridad:

- Pipeline con *scripts* de bash y sqlite3 para descargar y procesar la base de datos ChEMBL a fin de obtener el set de datos 'útil'
- Tabla de compuestos '*FULL_TABLE_COMPOUNDS.csv'* obtenida mediante el *pipeline* anterior
- Programa 1_similarity_search.html (para visualizar ejemplo)
- Programa 1_similarity_search.ipynb (para probar ejemplo)
- Programa 2_similarity_search_evaluation.html (para visualizar ejemplo)
- Programa 2_similarity_search_evaluation.ipynb (para probar ejemplo)

Respecto a la segunda parte del proyecto, que consiste en predicción de interacción de compuestos con una diana mediante *machine learning*:

- *Pipeline* con scripts de bash para descargar *decoys* desde la base de datos DUDE para algunas dianas
- Programa 1_machine_learning_target_prediction.html (para visualizar ejemplo)
- Programa 1_machine_learning_target_prediction.ipynb (para probar ejemplo)

Respecto al proyecto en su conjunto:

- Memoria del trabajo (este documento)
- Presentación del proyecto (en video)

1.6 Breve descripción de los otros capítulos de la memoria

El capítulo uno ‘Introducción’, pone en contexto y define a grandes rasgos este proyecto de fin de máster, comentando sobre la planificación y los resultados obtenidos.

El capítulo dos ‘Aproximación mediante búsqueda por similitud química’ explica el programa desarrollado para este objetivo, así como un comentario sobre los resultados y valoración del método. El capítulo comienza con una explicación teórica del fundamento del programa, moviéndose progresivamente hacia cuestiones más prácticas.

El capítulo tres ‘Aproximación mediante *machine learning*’, de manera análoga al capítulo anterior, explica el programa desarrollado para esa parte del proyecto. El capítulo comienza con una explicación teórica sobre el algoritmo específico empleado, así como el tratamiento de los datos de partida. Por último, se incluyen apartados sobre los resultados del programa y de su valoración.

El capítulo cuatro ‘Conclusiones’, hace una reflexión crítica sobre el trabajo, qué objetivos se han cumplido y las lecciones aprendidas, y finaliza con un comentario sobre el trabajo que queda pendiente o que sería interesante continuar.

El capítulo cinco ‘Glosario’ define varios términos de interés que se utilizan de forma asidua en el documento.

El capítulo 6 ‘Bibliografía’ detalla las fuentes de información utilizadas en la elaboración de este documento

El capítulo 7 ‘Anexos’ está principalmente dedicado a mostrar el código de los programas más relevantes del proyecto, comentado.

2. Aproximación mediante búsqueda por similitud química

2.1 Similitud química

La similitud o similaridad química es un concepto central en la quimioinformática. La naturaleza química de los distintos compuestos que existen no es ilimitada, y sin embargo, es evidentemente muy amplia. De hecho, tanto es así, que no existe una forma única y clara de 'comparar' dos moléculas. Es relativamente sencillo determinar si dos compuestos son en realidad la misma especie química o no, pero cuando el objetivo es determinar de forma cuantitativa su parecido, la tarea crece en varios órdenes de dificultad.

La justificación para utilizar las búsquedas por similitud química reside en el conocido principio que establece que moléculas estructuralmente similares tienden a tener propiedades (fisicoquímicas) similares [1]. Esto se traduce en que conociendo la actividad biológica de un compuesto determinado, es razonable esperar que compuestos estructuralmente similares exhiban una actividad similar. Debido a esto a menudo se utilizan las búsquedas por similitud química como la versión *in silico* del *screening* de alto rendimiento, y se incluyen en la familia de técnicas actualmente conocidas como *virtual screening. S*u nombre se refiere al hecho de que es un *screening* con base completamente computacional.

Es por esto que el concepto de similaridad química ha sido ampliamente explorado por químicos y quimioinformáticos [4]. A continuación se ofrece un resumen orientativo de las distintas estrategias que han sido desarrolladas para tratar de comparar de forma cuantitativa distintas estructuras químicas.

Los *fingerprints* (literalmente 'huellas dactilares') de compuestos químicos son una generalización del bien conocido concepto forense. De forma análoga a las huellas dactilares humanas, los *fingerpritns* de compuestos químicos son suficientemente únicos como para distinguir cada compuesto químico, pero no bastan para determinar exactamente cómo es dicho compuesto químico. Hay varios tipos de *fingerprints* químicos, si bien en este caso nos centraremos en los *fingerprints* 2D.

Los *fingerprints* 2D son vectores binarios (es decir, pueden contener unos y ceros), en el que cada bit indica la presencia (1) o ausencia (0) de un fragmento subestructural en la molécula (la teoría de cómo detectar estas estructuras está basada en redes topológicas matemáticas). A partir de estos *fingerprints* pueden calcularse diversas medidas de distancia o similitud entre ellos (son conceptos opuestos, por lo que tienen una relación inversa; cuanto mayor es uno, más disminuye el otro). Si se están comparando los *fingerprints* de las moléculas A y B, sea *a* el número de bits "1" en A, *b* el número de bits "1" en B, y *c* el número de bits "1" tanto en A como B, pueden definirse diversas medidas de distancia o similitud.

Nombre	Fórmula para variables binarias
Tanimoto (coeficiente de Jaccard)	$S_{AB} = \frac{c}{a+b-c}$, rango: 0 a 1
Coeficiente de Dice (índice de Hodgkin)	$S_{AB} = \frac{2c}{a+b}$, rango: 0 a 1
Similaridad del Coseno (índice de Carbó)	$S_{AB} = \frac{c}{\sqrt{ab}}$, rango: 0 a 1
Distancia euclidea	$D_{AB} = \sqrt{a+b-2c}$, rango: 0 a N
Distancia de Hamming (Manhattan)	$D_{AB} = a+b-2c$, rango: 0 a N
Distancia de Soergel	$D_{AB} = \frac{a+b-2c}{a+b-c}$, rango: 0 a 1

Tabla 1: Medidas de distancia y similitud [4]

Notar que los tres primeros coeficientes son de similaridad, por lo que serán altos si las moléculas A y B son similares, e irán descendiendo conforme más diferentes sean. Al contrario, los tres siguientes índices de distancia serán más altos conforme más alejadas estén las dos moléculas. Además, hay que tener en cuenta que dos de los índices, la distancia euclidea y la distancia de Hamming no tienen un rango acotado de 0 a 1, por lo que difícilmente serán comparables con otras medidas de distancia que si tienen un rango acotado fijo.

Por otra parte, debido a la naturaleza de las fórmulas de la tabla anterior, cada índice suele tender a considerar más importantes unos u otros aspectos de las moléculas. Por ejemplo, los coeficientes de Tanimoto, de Dice y del coseno dependen directamente del número de bits en común. Por contraste, las distancias de Hamming y euclidea consideran la común ausencia de características como criterio de similaridad. Una consecuencia importante de esta diferencia es que al tratar con moléculas pequeñas, a menudo obtendremos valores de similaridad bajos (altos coeficientes de distancia) utilizando el coeficiente de Tanimoto (y parecidos) debido sencillamente al hecho de que moléculas pequeñas tendrán pocos bits de tipo “1” (porque al ser moléculas pequeñas contendrán pocas subestructuras químicas).

No es poco el esfuerzo que ha sido invertido en determinar qué métricos son los mejores y a caracterizar sus fortalezas y fallos, y precisamente uno de los índices preferidos para búsquedas basadas en similitud química es el de Tanimoto [4] [5]. Si bien es cierto que los índices parecidos al de Tanimoto pueden ‘inflar’ las diferencias entre moléculas cuando estas son pequeñas, el motivo es que este índice incluye un término de normalización por tamaño en el denominador, y esto previene un sesgo hacia reducir las diferencias entre moléculas grandes (que, al contrario que en el problema anteriormente explicado, al tener más bits tipo “1” pueden aparentar ser más parecidas). No obstante, entre la mayoría de los expertos en el campo hay consenso en que lo

ideal es combinar varios índices para tratar de superar las limitaciones que cada uno sufre si son utilizados de manera individual. Porque es un índice ampliamente utilizado por su robustez, y porque para esta aplicación no se requiere demasiada sofisticación, el índice de similaridad que se utilizará en este proyecto será el de Tanimoto.

2.2 Base de datos ChEMBL

ChEMBL es mucho más que una base de datos. En su versión 22_1, la plataforma mantenida por el Instituto Europeo de Bioinformática (EBI) junto con el Laboratorio Europeo de Biología Molecular (EMBL) incluye más de 11500 dianas moleculares, registros de más de 1.7 millones de compuestos con más de 14.5 millones de actividades asociadas, y enlaces a más de 67000 publicaciones con evidencias sobre fármacos bioactivos. Además, la plataforma contiene herramientas para buscar fármacos (permite dibujar la estructura que se desea buscar), buscar fármacos por subestructura, por diana, por indicaciones y mucho más [6].

En nuestro caso, no obstante, el principal interés de la plataforma son sus datos. ChEMBL contiene un registro considerablemente amplio de fármacos, y gran parte del esfuerzo en su desarrollo se dirige a facilitar la descarga y uso de estos datos. Desde la propia página (https://www.ebi.ac.uk/chembl/downloads), es posible descargar la base de datos al completo en diferentes formatos. Por mera preferencia personal, se ha optado por descargar la base de datos en versión sqlite3 (extensión .db).

La base de datos de ChEMBL está organizada en distintas tablas, y cada tabla guarda un registro distinto. Distintas tablas están relacionadas por términos comunes y códigos de acceso (como por ejemplo, número de registro de molécula). El siguiente diagrama, presente en el propio servidor de ChEMBL a modo de orientación, da una idea de la extensión y tipo de información que contiene esta base de datos. Nótese que es una imagen muy grande diseñada para ser visualizada haciendo *zoom.* Las tablas se encuentran en ‘cajas’ con color de fondo distinto cada una, reflejando distintas áreas temáticas (información del compuesto, información experimental, información de la diana…).

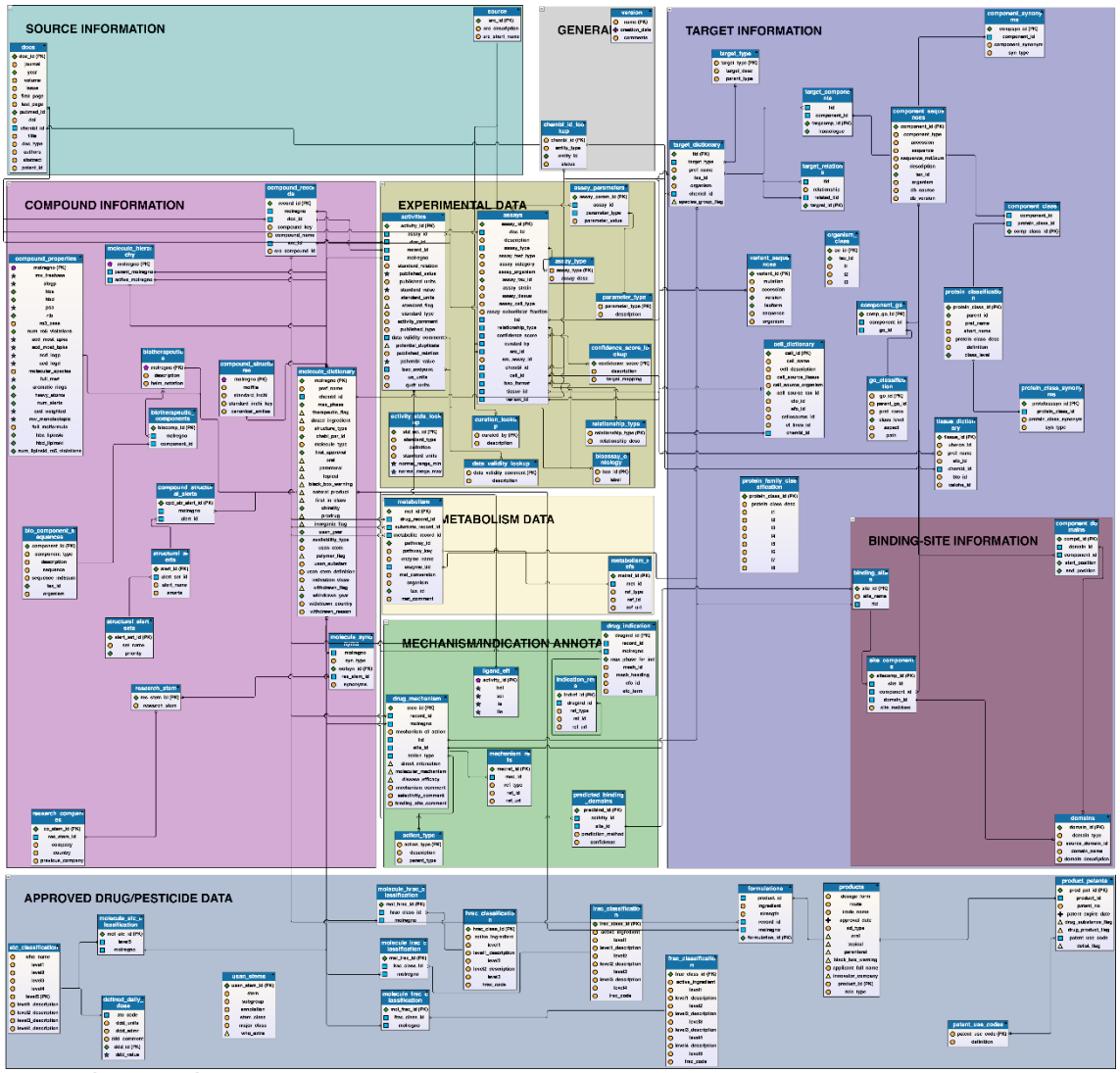


Ilustración 4: Patrón de tablas de ChEMBL

Las tablas que se utilizan, junto con qué campos de datos específicos, son las siguientes

- compound_structures
  - molregno: número de identificación de compuesto
  - canonical_smiles: código SMILES del compuesto
- drug_indication
  - molregno
  - mesh_heading:
- drug_mechanism
  - molregno
  - mechanism_of_action
  - tid
- molecule_dictionary
  - molregno
  - pref_name
- target_dictionary
  - tid
  - pref_name

Con estas tablas, el programa 1_similarity_search ensambla el set de datos que se utilizará, y lo vuelca en el archivo FULL_TABLE_COMPOUNDS.csv. El proceso de cargar las tablas y cruzarlas es un paso que requiere relativamente mucha memoria (si bien se ha ahorrado mucha cargando solo las columnas de interés en lugar de las tablas al completo), y exportando el *dataframe* resultante a un archivo permite agilizar usos posteriores de estos datos.

El objetivo inicial era crear un set de datos con todos los compuestos de ChEMBL que bien tuvieran información sobre su *target* y/o tuvieran información asociada sobre sus efectos o mecanismos. No obstante, sucede que hay compuestos que tienen mecanismo y/o indicación médica (efecto) asociado, pero no están asociados a ningún código SMILES. Se escribió un email al servicio técnico de ChEMBL, pero no ha sido respondido en dos meses. El siguiente gráfico ilustra los distintos tipos de registros que están contenidos en el set de datos: registros con indicaciones médicas, mecanismo, y código SMILES, registros con indicaciones médicas y código SMILES, y registros con mecanismo de acción asociado y código SMILES. La cuantía total de compuestos que cumple cualquiera de estas condiciones asciende a 3455 (recuadro rojo).

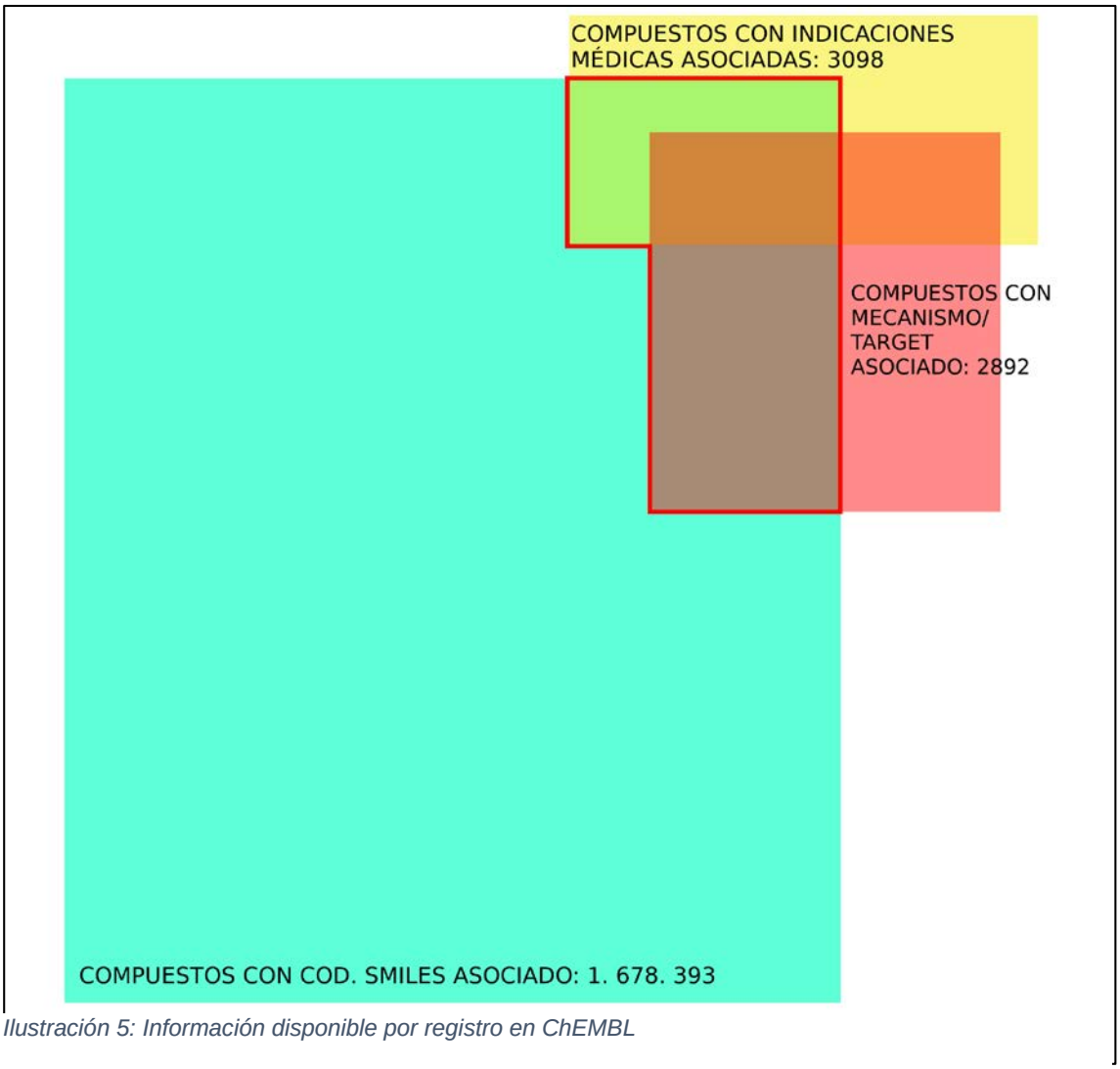


Ilustración 5: Información disponible por registro en ChEMBL

2.3 Resultados:

A este documento se adjunta el programa 1_similarity_search, con el código comentado, por lo que no se explicará su funcionamiento en detalle en esta sección. Se incluye en formato html, para observarlo con ejemplos de cómo funciona, y en formato ipynb (jupyter *notebook*) a fin de poder probarlo. El programa se ha dividido en varias secciones que se deben ejecutar de forma secuencial.

Antes de poder ejecutar el programa, debe utilizarse el *pipeline* que se adjunta para descargar, descomprimir, y extraer las tablas de ChEMBL en formato *csv*. El *script* 0_main.sh se encarga de ejecutar una serie de scripts que descargan y descomprimen la base de datos de ChEMBL, y el *script* 1_extract_tables.sh ejecuta una serie de comandos en sql con el programa sqlite3 para exportar las tablas a archivos de extensión csv.

En la primera sección, el programa carga las tablas que previamente deben haber sido obtenidas mediante el *pipeline*. Mediante la carga de columnas específicas, se ahorra mucho espacio en memoria. Esto es particularmente importante con el archivo al cual se ha volcado la tabla *compound_structures*, la cual es muy extensa.

La segunda sección, opcional, realiza una cuenta de los datos, poniendo de relieve que hay registros con indicaciones de mecanismo o de uso sin estructura química (código SMILES asociado).

En la tercera sección, el programa cruza las distintas tablas cargadas previamente, construye el set de datos que se va a utilizar con los campos necesarios, y exporta el set de datos al archivo FULL_COMPOUNDS_TABLE.csv, a fin de que este proceso de construcción del *dataset* pueda ser omitido en ejecuciones posteriores del programa. En la siguiente figura se observa el aspecto de la tabla creada, con los tres primeros compuestos.

	molregno	pref_name	canonical_smiles
0	115	NICOTINE	CN1CCC[C@H]1c2cccnc2
1	146	OFLOXACIN	CC1COc2c(N3CCN(C)CC3)c(F)cc4C(=O)C(=CN1c24)C(=O)O
2	147	NALIDIXIC ACID	CCN1C=C(C(=O)O)C(=O)c2ccc(C)nc12

mesh_heading	mechanism_of_action	tid	target
TOBACCO USE DISORDER;SMOKING CESSATION;MEMORY ...	Neuronal acetylcholine receptor; alpha4/beta2 ...	104283	Neuronal acetylcholine receptor; alpha4/beta2
HEARING LOSS;OTITIS MEDIA	Bacterial DNA gyrase inhibitor	105577	Bacterial DNA gyrase
NaN	Bacterial DNA gyrase inhibitor	105577	Bacterial DNA gyrase

Ilustración 6: Tres registros de la tabla FULL_TABLE_COMPOUNDS

En la cuarta sección, se toma un código SMILES de una molécula problema (en el ejemplo que se adjunta se ha utilizado la Simvastatina, un compuesto que inhibe la síntesis de colesterol endógeno), y se busca en el set datos los registros de compuestos que superen un umbral de similaridad (por defecto, es de 0.7). Para ello, se calcula el índice de similaridad de Tanimoto de todos los compuestos de la tabla respecto a la molécula de *input*. El cálculo de la distancia es sorprendentemente rápido, siendo los procesos más lentos instanciar la clase 'molecule' de Rdkit y calcular sus *fingerprints*. A término de la ejecución de esta sección, se obtiene una tabla con los resultados, esto es, los compuestos químicamente más parecidos a la molécula de *input*. Estos pueden observarse en la siguiente figura.

	similarity	molregno	pref_name	canonical_smiles	mesh_heading
0	0.976834	9495	LOVASTATIN	CC[C@H](C)C(=O)O[C@H]1C[C@@H](C)C=C2C=C[C@H](C...	PULMONARY DISEASE, CHRONIC OBSTRUCTIVE;HIV INF...
1	0.871988	30602	PRAVASTATIN SODIUM	[Na+].CC[C@H](C)C(=O)O[C@H]1C[C@H](O)C=C2C=C[C...	HIV INFECTIONS;ACUTE CORONARY SYNDROME;ANGINA ...
2	1.000000	138562	SIMVASTATIN	CCC(C)(C)C(=O)O[C@H]1C[C@@H](C)C=C2C=C[C@H](C)...	ALZHEIMER DISEASE;PULMONARY DISEASE, CHRONIC O...

	mesh_heading	mechanism_of_action	tid	target
PULMONARY DISEASE, CHRONIC OBSTRUCTIVE;HIV INF...	HMG-CoA reductase inhibitor	24	HMG-CoA reductase
HIV INFECTIONS;ACUTE CORONARY SYNDROME;ANGINA ...	HMG-CoA reductase inhibitor	24	HMG-CoA reductase
ALZHEIMER DISEASE;PULMONARY DISEASE, CHRONIC O...	HMG-CoA reductase inhibitor	24	HMG-CoA reductase

Ilustración 7: Las tres moleculas más similares a la Simvastatina

Como se puede apreciar, la cercanía en términos de similaridad química bien puede funcionar como una forma de predecir propiedades de compuestos de interés terapeútico, tales como efecto, indicación médica o diana molecular. Exceptuando el tercer compuesto que tiene una similaridad de 1 (esto significa

que es el mismo compuesto), la información asociada a los otros sirve como predicción de que la simvastatina se utiliza en pacientes con dolencias cardiovasculares, y que su diana molecular es la HMG-CoA reductasa. Esta información es correcta, por lo que se puede concluir que la predicción es buena.

En este punto, uno podría refutar que utilizar una especie de fármaco tan abundante como son las estatinas (hay una enorme familia de fármacos como la simvastatina; en la figura anterior se observan la pravastatina y la lovastatina) no es representativo de lo útil que es el método con familias de fármacos menos abundantes o incluso no representada en este *set* de datos. Por este motivo, se ha creado otro programa para realizar tests automatizados del método.

2.4 Valoración del predictor:

Con el ejemplo de la sección anterior queda claro que el método funciona. Es intuitivo pensar que el 'cuello de botella' limitante de este método de predicción consiste en lo completa que sea la base de datos utilizada. Para evaluar cómo funciona la que se ha construido en este proyecto, se ha creado el programa 2_similarity_search_evaluation. Este, como el anterior, se incluye tanto en formato html (para observar el ejemplo) como en ipynb (para poder ejecutar y probar el programa).

Básicamente, 2_similarity_search toma la tabla creada en el programa anterior, extrae 10 compuestos, y para cada uno de ellos 'predice' su acción buscando compuestos similares en la tabla completa, pero sin esos compuestos. Este método de valoración es conocido como *leave-one-out*, que de forma genérica se refiere a tomar uno de los ejemplos que se ha utilizado para entrenar un modelo, o como en este caso, hacer una tabla de compuestos de búsqueda, eliminarlo de dicha tabla, y tratar de predecir la clasificación o información de dicho ejemplo. A continuación, el programa presenta, para cada compuesto, sus efectos y dianas 'reales' así como los predichos (es decir, de compuestos que se han determinado como similares según el índice de similaridad de Tanimoto. En el ejemplo adjuntado en el apéndice se observa, entre otros, el siguiente resultado, que es parte de la salida del programa.

```
#####################################################################

ACTUAL Medical indications:
        KERATOSIS- ACTINIC
        MUSCULAR DISEASES
        OSTEOARTHRITIS
        PAIN
        DYSMENORRHEA
        OSTEOARTHRITIS- KNEE
        SKIN NEOPLASMS
        ARTHRITIS- RHEUMATOID
        MULTIPLE SCLEROSIS
ACTUAL Mechanism:
        Cyclooxygenase inhibitor
//////////^\\\\\
Molecule registry number:  123693
```

```
Medical indications:
        PANCREATITIS
        PAIN
        INFERTILITY
        MYALGIA
        OSTEOARTHRITIS
        PULMONARY EMBOLISM
        MACULAR EDEMA
        WOUNDS AND INJURIES
        FEVER
        EYE DISEASES
        RHEUMATIC DISEASES
        ARTHRALGIA
        SPONDYLITIS- ANKYLOSING
        KERATOSIS- ACTINIC
        ARTHRITIS- RHEUMATOID
        URINARY TRACT INFECTIONS
        MIGRAINE DISORDERS
        RESPIRATORY TRACT INFECTIONS
        GLAUCOMA
        OSTEOARTHRITIS- KNEE
        DYSMENORRHEA
        CARCINOMA- BASAL CELL
        DEPRESSIVE DISORDER
        DIABETIC RETINOPATHY
Mechanism
        Cyclooxygenase inhibitor

#####################################################################
```

En este ejemplo se observan las indicaciones y mecanismo de un compuesto, y debajo los que han sido 'predichos' por estar asociados a compuestos químicamente similares. En este caso la predicción parece ser bastante buena, ya que casi todas las indicaciones y el mecanismo de acción coinciden.

En el siguiente ejemplo, se aprecia algo curioso. En algunos casos, el compuesto estaba asociado a cierta información, pero le faltaba o mecanismo o indicación médica, y la predicción (en este caso es siempre *dentro* del set de datos de la tabla) completa la información. Esto se ilustra a continuación.

```
#####################################################################

ACTUAL Medical indications:
        No information available
ACTUAL Mechanism:
        Bacterial dihydropteroate synthase inhibitor
//////////^\\\\\
Molecule registry number:  2130
Medical indications:
        INFECTION
Mechanism
        Bacterial dihydropteroate synthase inhibitor

#####################################################################
```

En el ejemplo de arriba, se observa que en el compuesto de consulta no había indicación médica. Mediante similaridad química, se ha predicho correctamente cuál es su *target*, así como que la indicación médica es infección ('INFECTION').

Finalmente, se ha probado el programa con 200 compuestos aleatorios, mediante la aproximación *leave-one-out*, que como se indicaba anteriormente consiste en eliminar un compuestos de la tabla construida y predecir sus efectos y dianas moleculares. El criterio que determina que una predicción es correcta ha sido que al menos la indicación médica o el mecanismo molecular sean comunes entre la predicción y los datos conocidos del compuesto. En este escenario, el 37 % de los compuestos han sido correctamente predichos.

La idea de hacer este programa era poder hacer muchas pruebas seguidas de predicciones para poder determinar un índice o porcentaje de aciertos. No obstante, existe el problema de cómo valorar cuando es un *acierto*. Si determinamos que un *acierto* es que todos los campos coincidan, el índice de aciertos será muy bajo, pero cualquier otro criterio será en mayor o menor medida, arbitrario. Por ejemplo, se podría analizar desde un punto de vista bioquímico como de precisas son las predicciones, o si aunque sean diferentes, están relacionadas en el plano biológico.

Por otra parte, está el hecho de que al no crearse un modelo propiamente dicho, la capacidad predictiva está firmemente sujeta a lo completo que sea el set de datos que se ha construido. Muchos de los compuestos carecen de la información completa, y esto hace que sea complicado determinar si la predicción es correcta o no.

Para incrementar más la dificultad de evaluar este método de predicción, aun y si se dispusiera de una base de datos con absolutamente todos los compuestos químicos que existen y su acción, el hecho es que el espacio químico no está uniformemente distribuido, y siempre habrá tipos de fármacos que se predecirán con más facilidad (por ser más abundantes) que otros (que puede que sean los únicos representantes de fármacos de ese tipo químico, y por lo tanto es virtualmente imposible predecirlos correctamente con este método).

3. Aproximación mediante *machine learning*

El término *machine learning,* que puede ser traducido como ‘aprendizaje automático’, engloba toda una serie de algoritmos y técnicas que permiten hacer programas que ‘aprendan’ de ejemplos y posteriormente realicen tareas de predicción o clasificación. Este concepto que existe en la intersección entre las matemáticas, la estadística, las ciencias de la computación y la inteligencia artificial, tiene aplicaciones tan variadas e interesantes como son la visión de ordenadores (programas capaces de analizar e interpretar imágenes), filtrado de correo basura, o predicción de valor de mercados en bolsa.

Por supuesto, su aplicación en los campos de la biología y la bioinformática es interesante, y gracias a multitud de librerías de cálculo científico que han sido desarrolladas con un usuario no experto en mente, es fácil utilizar este tipo de herramientas.

Existen multitud de algoritmos de *machine learning*. Cada uno funciona de manera distinta y están controlados por distintos parámetros. Por ello, cada algoritmo tiene ventajas e inconvenientes, y eso perfila para qué tareas son más adecuados. En esta parte del proyecto, se empleará el algoritmo conocido como *Support Vector Machine*, que se explicará a continuación con un ejemplo.

3.1 Support Vector Machines

Las máquinas de soporte vectorial o *Support Vector* Machines son un algoritmo de *machine learning* clasificado como de aprendizaje supervisado. El aprendizaje supervisado consiste en que el algoritmo construye una función de clasificación en base a ejemplos con categorías específicas. Una vez se construye esta función (o como se suele decir, se *entrena* el modelo), el algoritmo debería ser capaz de predecir la categoría de nuevos ejemplos no categorizados. En oposición, el aprendizaje no supervisado lidia con la tarea de encontrar estructura en datos no categorizados. Este último tipo de algoritmo puede ser muy útil, por ejemplo, si se tienen datos sobre el estilo de vida de consumidores de un determinado producto, y se quiere determinar si existen grupos de consumidores a fin de hacer ofertas que les interesen.

Para explicar su funcionamiento, lo haremos con un ejemplo. Digamos que se quiere predecir si un artículo científico va a publicarse o no. Las características o *features* con las que se representará cada ejemplo de artículo serán longitud (número de palabras) y reputación de los autores (suma del número de las publicaciones anteriores de los autores). Los datos que se utilizarán con el algoritmo serán ejemplos positivos (es decir, artículos que han sido publicados) y ejemplos negativos (artículos que han sido rechazados), que serán codificados como clase 1 y 0 respectivamente. Estos ejemplos pueden visualizarse en la siguiente figura

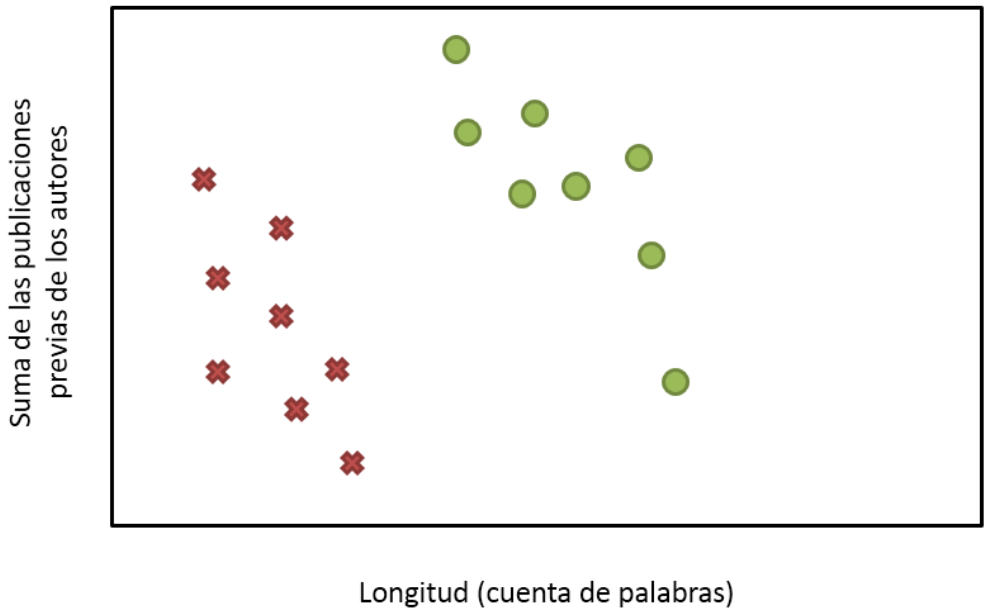


Ilustración 8: Ejemplos positivos (verde) y negativos (rojo)

Con estos datos, el algoritmo tratará de encontrar la línea que separa de forma más óptima (con mayor margen) ambos grupos de ejemplos. Para esto, el algoritmo ejecuta una serie de operaciones algebraicas con los ejemplos que se encuentran en la interfaz de ambos. Este proceso se encuentra representado de forma esquemática en la siguiente figura.

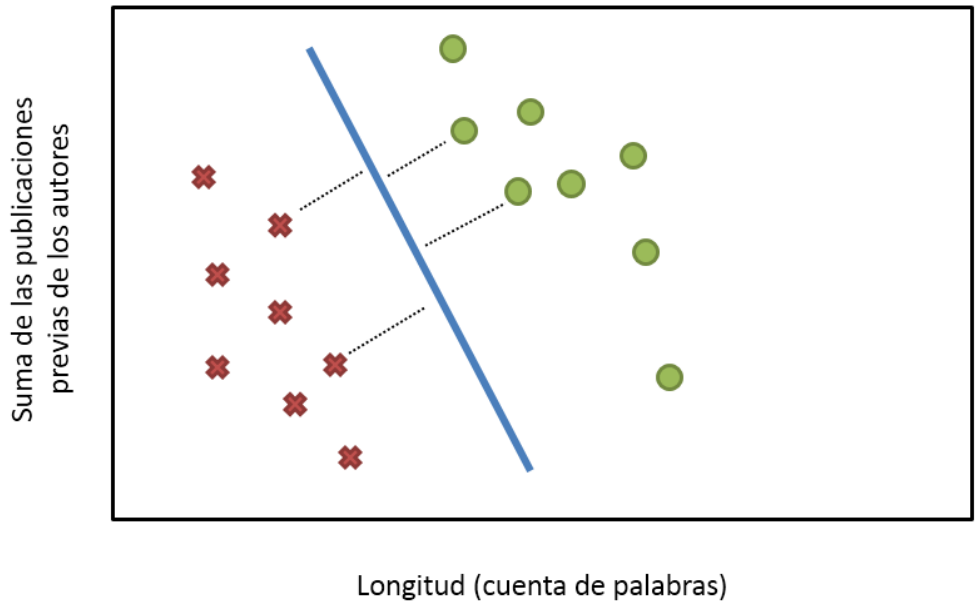


Ilustración 9: Creación del modelo SVM

Las líneas discontinuas son los llamados ‘vectores de apoyo’, con los que se construye la línea que separa ambos grupos, y de ahí el nombre del algoritmo. Una vez se ha entrenado el modelo (es decir, la línea que separa ambos grupos ha sido calculada), se pueden introducir nuevos casos de artículos para los cuales se quiera predecir si van a ser publicados o no, y el algoritmo devolverá la predicción basándose en qué lado de la línea están.

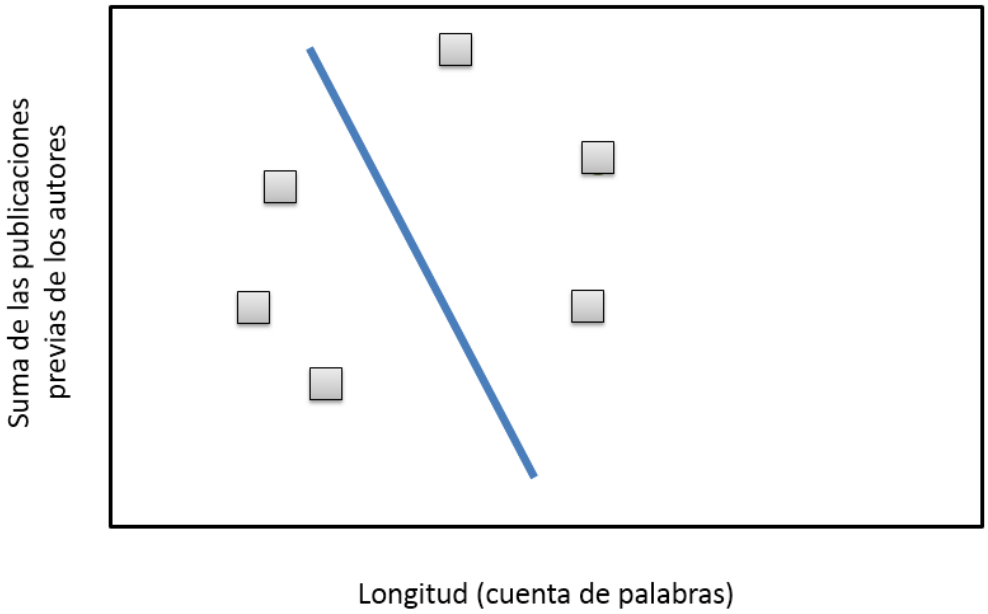


Ilustración 10: Se introducen nuevos casos en el modelo

En la figura anterior se observa cómo se podría visualizar el clasificador (línea azul), junto con los casos de artículos sin clase asociada (cuadrados grises). En la figura siguiente, se observa cómo el clasificador predice la clase de cada caso en base a en qué lado de la línea se encuentran.

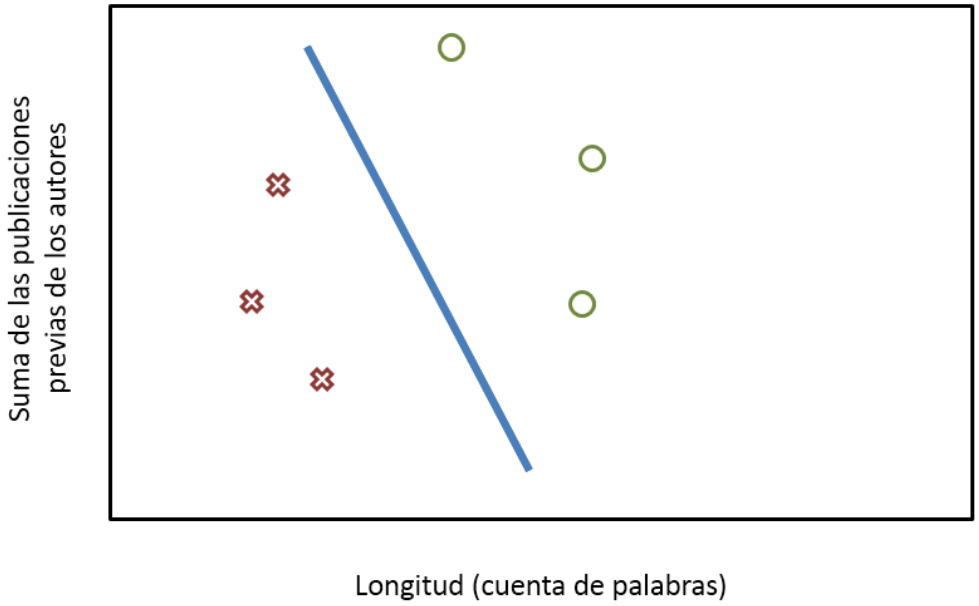


Ilustración 11: Predicción del modelo SVM

Podríamos tratar de hacer un clasificador mejor incluyendo una tercera característica, como por ejemplo, el índice de impacto de la revista en la cual se quiere publicar el artículo. En este escenario, el espacio en el que existirían los ejemplos sería un espacio tridimensional, en la que los ejes X, Y, y Z representarían longitud, prestigio de los autores, e índice de impacto de la revista respectivamente. Es intuitivo imaginarse que artículos que superen un mínimo de palabras, cuyos autores tengan cierta reputación y haya sido enviado a una revista no demasiado importante, serán publicados. En este caso, las dos clases, artículos publicados y artículos rechazados, estarán separados no por una línea, sino por un plano bidimensional, que existe en un espacio tridimensional (cada caso tiene tres características asociadas).

Aunque no es posible representarlo gráficamente, podríamos seguir añadiendo características a los ejemplos (dimensiones espaciales al modelo), hasta llegar a tener ejemplos representados en un espacio N dimensional, con ambos grupos separados por un plano "N-1" dimensional. Este plano se generaliza con el nombre de 'hiperplano', que no es más que un plano de cualquier dimensión. La línea del ejemplo inicial, por ejemplo, sería un hiperplano de dimensión 1. De esta forma, y siempre que ambos grupos sean correctamente separados por un hiperplano, el clasificador podrá predecir de forma satisfactoria la clase de nuevos casos.

Los *support vector machine* son algoritmos muy versátiles no solo por la alta dimensionalidad de datos con la que pueden trabajar de forma eficiente, sino por una serie de hiperparámetros que permiten adaptar el algoritmo a muchas tareas distintas [3]. Entre estos hiperparámetros, el más importante probablemente sea el *kernel*.

El *kernel* es una característica utilizada en muchos algoritmos de *machine learning*, entre ellos el *support vector machine*. Explicado de manera simplificada, permite convertir datos no separables por un hiperplano, en datos que son separables fácilmente por un hiperplano. Uno de los ejemplos más conocidos de este procedimiento es el siguiente. Pongamos que se observa una montaña desde las alturas, y se toman coordenadas en X e Y, y se determina si la temperatura es alta o baja. Estos datos se representan en la figura que sigue.

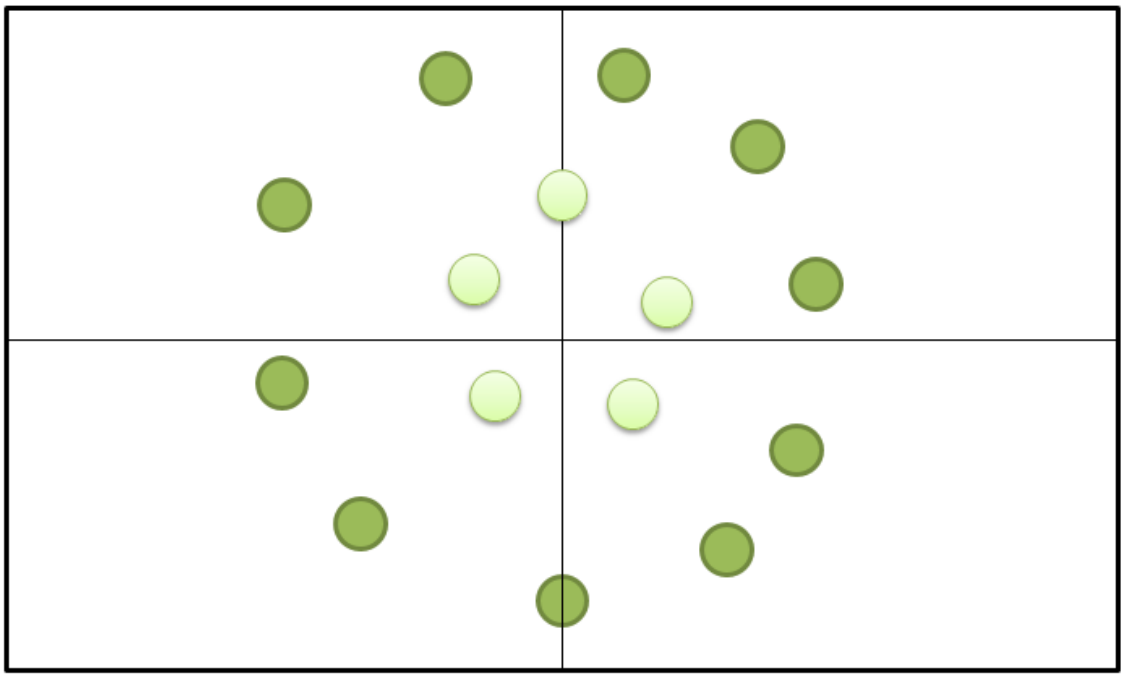


Ilustración 12: Visualización de datos sin transformar

Como cabría de esperar, el pico de la montaña es el punto más frío (simbolizado con un verde mucho más claro) que la falda de la montaña (puntos verdes). A priori uno podría pensar que no es posible separar estos datos con un hiperplano, pero de hecho, es posible aplicar una sencilla transformación de los datos para este propósito. Definiremos una nueva dimensión Z, cuyo valor será $X^2 + Y^2$. A continuación representaremos el eje X tal y como estaba en la figura anterior, pero sustituiremos el eje Y por el nuevo eje Z. Podríamos haber añadido un eje Z como altura respecto al plano X/Y, caso en el que la figura habría adoptado la forma de un cono invertido.

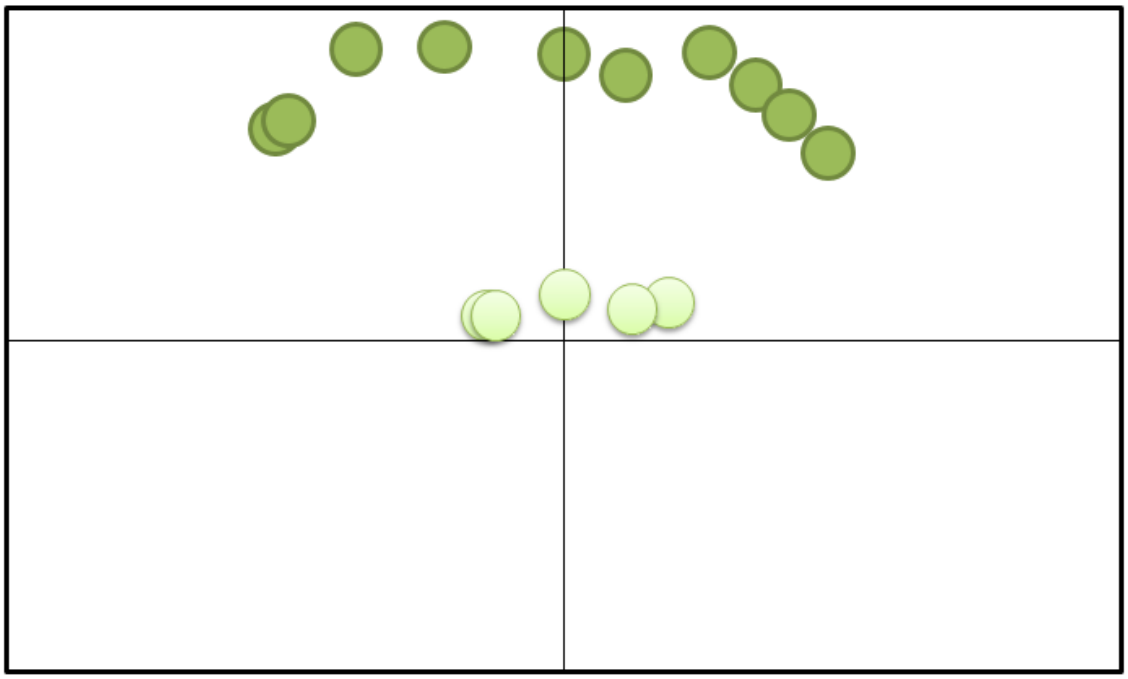


Ilustración 13: Visualización de transformación de datos

Como se observa en la figura anterior, con esta sencilla transformación en base a la distancia desde cada punto hasta el centro, hemos obtenido una nueva dimensión en la que es trivial separar ambos grupos mediante un hiperplano. Las distintas implementaciones de *SVM* (*Support Vector Machines*) incluyen diferentes tipos de *kernels* o transformaciones, así como la posibilidad de definir un *kernel* a medida. Estos añaden una capa de flexibilidad al algoritmo, ya que normalmente siempre hay un *kernel* óptimo para cada tipo de tarea o forma de los datos con los que se trabaja.

En resumen, se ha escogido este algoritmo para la tarea de predecir si un compuesto se une a un *target*, porque trabaja bien con datos de alta dimensionalidad, ofrece una considerable flexibilidad a la hora de adaptar el modelo a distintos casos, y porque es muy eficiente con el uso de los datos (no emplea todos los casos que se utilizan como ejemplo).

En la siguiente sección se explican en qué consistirán las características o *features* que se utilizarán en este caso: descriptores moleculares

3.2 Descriptores moleculares

La definición ‘clásica’ de los descriptores moleculares es ‘el resultado de un procedimiento lógico y matemático que transforma la información química codificada mediante una representación simbólica de una molécula en un número útil o el resultado de un experimento estandarizado’ [7]. Esta definición es exhaustiva en el sentido en el que contempla cualquier forma que puedan adoptar los descriptores moleculares. No obstante, de manera sencilla, los descriptores moleculares pueden ser considerados como números que caracterizan cierta propiedad de compuestos químicos.

De entrada, los descriptores moleculares pueden dividirse en medidas experimentales (refractividad molar, momento dipolar, polarizabilidad, y en general, *cualquier* propiedad fisicoquímica) y teóricos, que se derivan de la representación simbólica de la molécula y pueden ser clasificados en mayor detalle dependiendo de en qué representación molecular estén basados. En concreto, los descriptores moleculares teóricos se pueden clasificar en

- 0D : Descriptores constitucionales, p.e. contar número de átomos pesados
- 1D Cuentas de fragmentos estructurales, *fingerprints*
- 2D : Características topológicas (basadas en teoría de grafos)
- 3D : Descriptores basados en tamaño, superficie y volumen
- 4D : Combina características anteriores además de características cuánticas

Si bien teóricamente pueden existir infinitos descriptores moleculares, lo cierto es que hay una serie de propiedades que hacen que un descriptor molecular sea 'bueno', y por extensión, 'útil'. Una de las propiedades más importantes que deben poseer los descriptores moleculares es la llamada 'invarianza', que se define como la capacidad de un algoritmo (que calcule un descriptor molecular) de obtener un resultado de forma independiente a características específicas de la representación molecular. Por poner un ejemplo, un descriptor molecular debería ser el mismo sin importar que la estructura molecular esté girada en el espacio.

Otra propiedad importante es la 'degeneración' (entendida como la degeneración del código genético, en el que más de un codón codifica para el mismo aminoácido). En este caso, esta propiedad se refiere al hecho de que dos moléculas distintas no deberían tener el mismo descriptor molecular. Por lo tanto, la degeneración puede ser nula (no hay dos moléculas que tengan el mismo descriptor molecular), media, alta, o completa (todas las moléculas tienen el mismo descriptor molecular). Es deseable que la degeneración de un descriptor molecular sea tan baja como sea posible.

Por otra parte, existen una serie de requerimientos para que un descriptor molecular sea considerado óptimo. Estos incluyen que no estén basados en medidas experimentales, que tengan una interpretación estructural, que cambien de forma gradual junto con cambios graduales en las moléculas, y que discriminen entre isómeros, por poner algunos ejemplos. Por supuesto, es complicado encontrar un descriptor molecular que cumpla todos estos requerimientos, y aunque lo hubiera, a menudo interesa utilizar más de un descriptor molecular, a fin de obtener una representación de varias dimensiones de la molécula.

3.3 Docking decoys

Los datos que serán utilizados como *input* para el algoritmo SVM serán una serie de descriptores moleculares, calculados para compuestos que se unen (ejemplos positivos, codificados como 1) y compuestos que no se unen al *target* (ejemplos negativos, codificados como 0). Los ejemplos positivos se tomarán de la tabla creada en la parte de la búsqueda de similaridad, que incluye, para una serie de compuestos, el *target*. En cuanto a los ejemplos negativos, en lugar de utilizar *cualquier* compuesto que no se una, se utilizarán *docking decoys*.

En concepto, son, precisamente, señuelos (*decoy*) diseñados para valorar el desempeño de programas de predicción de *docking* (acoplamiento molecular). Los *decoys* son específicos de cada *target*, y consisten en compuestos que no se unen al target, pero con características fisicoquímicas parecidas a compuestos que sí se unen (por eso son *decoys*, señuelos). No obstante, estos *docking decoys* también pueden utilizarse para nuestros fines, como ejemplos negativos para entrenar un modelo de predicción de interacción.

La justificación de utilizar *docking decoys* en vez de *cualquier* compuesto que no se una es una pieza fundamental del proyecto. Por una parte, está el hecho de que al ser *decoys* (señuelos), han sido pensados para resultar difíciles de distinguir de las moléculas que si se unen, y por lo tanto si se entrena un modelo que sea capaz de distinguir entre compuestos activos y *decoys*, es intuitivo esperar que el clasificador funcione todavía mejor con el resto de moléculas. Esta intuición está sujeta a una serie de condiciones que deben cumplirse, las cuales se exploran más adelante.

Por otra parte, está la idea de que el objetivo final del clasificador es generalizar predicciones para cualquier compuesto a partir de los datos de entrenamiento. Volviendo al ejemplo con el que se explicaban los SVM (predecir si un artículo será publicado o no), está claro que interesa disponer de los casos que estén en la interfaz entre ambas clasificaciones. Si por ejemplo se cogiesen artículos de altísima calidad que estaba claro que iban a ser publicados, puede que el hiperplano calculado escorase hacia un criterio más exigente, y no generalizase bien la línea de separación ‘real’ entre publicar y no publicar un artículo. Esta idea se refleja en la siguiente figura. A la izquierda, se encuentra la representación de dos grupos que se encuentran en la interfaz de la clasificación. A la derecha, se observa que si se toman ejemplos positivos que exceden de sobra los criterios mínimos, el hiperplano calculado es distinto. Algo similar ocurre con los compuestos químicos, y utilizar los *docking decoys* es una buena forma de evitar este problema.

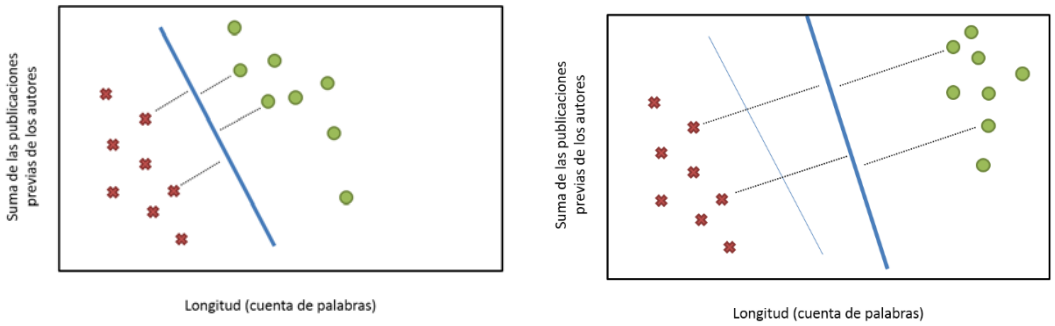


Ilustración 14: El hiperplano calculado cambia según los datos utilizados

En su momento, se valoró la posibilidad de incluir en el proyecto la creación de *decoys*, pero esto planteaba considerablemente más decisiones y tiempo de trabajo del inicialmente planteado, por lo que se optó por utilizar la base de datos DUDE, ampliamente reconocida y utilizada por la comunidad quimioinformática. DUDE (Directory of Useful Decoys, Enhanced) es una base de datos construida sobre su antecesor, DUD (Directory of Useful Decoys), mediante la inclusión de nuevos *targets* y *decoys* [8].

DUDE también ofrece la posibilidad de obtener compuestos activos. No obstante, en su lugar se utilizarán, como ya se ha explicado, los compuestos activos anotados como tal en la tabla de compuestos generada en la aproximación por búsqueda de similitud química. Esto es debido a que muchos compuestos activos de DUDE son muy parecidos, y no representan el espacio químico de los compuestos que si se unen.

3.4 Resultados

A este documento se adjunta el programa 1_machine_learning_target_prediction, tanto en formato html para poder visualizar el ejemplo, como en formato ipynb (Jupyter notebook) para poder probarlo. La ejecución del programa requiere del archivo 'FULL_TABLE_COMPOUNDS.csv', que se ha generado en el programa 1_similarity_search. No obstante, este archivo se adjunta a este documento a fin de que se pueda probar el programa con mayor facilidad. Por otra parte, el *script* get_decoys.sh se encarga de descargar los *docking decoys* de la base de datos DUDE.

En la primera sección, el programa carga la tabla de compuestos FULL_TABLE_COMPOUNDS.csv y cuenta cuantos targets tienen más de 20 compuestos asociados. Este es el resultado

target	molregno	pref_name	canonical_smiles
Androgen Receptor	27	27	27
Bacterial 70S ribosome	40	40	40
Bacterial penicillin-binding protein	57	57	57
Beta-2 adrenergic receptor	21	21	21
Cyclooxygenase	32	32	32
DNA	44	44	44
GABA-A receptor; anion channel	36	36	36
Glucocorticoid receptor	66	66	66
Histamine H1 receptor	47	47	47
Mu opioid receptor	32	32	32
Sodium channel alpha subunit	40	40	40

Ilustración 15: Targets que tengan al menos 20 fármacos asociados

Elegir un mínimo de 20 es arbitrario, pero la idea es escoger un *target* que tenga una cierta cantidad de fármacos activos asociados (ejemplos positivos para SVM), a fin de que estén correctamente representados en el espacio químico. Como se observa en la lista, no hay demasiados. Este proyecto se desarrolló inicialmente con el receptor beta adrenérgico 2, pero se han incluido pruebas con el receptor de andrógenos y el receptor glucocorticoide. Además, en esta sección se cargan los archivos con los *docking decoys* del target del cual se va a hacer un modelo de predicción de interacción.

En la segunda sección, el programa toma los datos cargados en la sección anterior, y crea el *dataframe* que será utilizado con el algoritmo. Este se construye a partir de los compuestos que se unen el *target* seleccionado (compuestos anotado como tales en FULL_TABLE_COMPOUNDS.csv) y de un número arbitrario de 200 *docking decoys* (se probaron diversas cantidades, y se determinó que alrededor de 150-200 es el número óptimo, ya que utilizando menos el modelo resulta peor, y utilizando más no mejor en absoluto). Para cada

uno de estos compuestos, se calculan hasta 12 descriptores moleculares, algunos de los cuales se describen a continuación [9] .

- TPSA o *Total Polar Surface Area*, se define como la superficie total de todos los átomos polares de una molécula.

- El índice J de Balaban es un índice topológico que da la medida de la complejidad, en términos topológicos, de la moléculas [9].

- MolLogP calcula el valor logP (coeficiente de partición de un compuesto entre dos fases inmiscibles) de Wildman-Crippen

- ExactMolWt o *Exact Molecular Weight* es, como su nombre indica, el peso molecular exacto del compuesto.

- Fr_Al_Oh es un descriptor molecular que cuenta el número de grupos hidroxilo alifáticos. También se utiliza el descriptor molecular fr_aldehyde, que cuenta el número de grupos aldehído alifáticos.

- MaxPartialCharge y MinPartialCharge son descriptores moleculares cuyo valor consiste en la carga parcial máxima y mínima de toda la molécula, respectivamente.

El resto de descriptores utilzados, que incluyen Estate_VSA1, Estate_VSA2, Ipc, y Kappa1, carecen de una interpretación sencilla.

Como se puede apreciar, los 12 descriptores moleculares seleccionados son muy variados. Sin duda hay descriptores objetivamente mejor que otros de acuerdo a los criterios ya comentados (redundancia, cambio progresivo…), si bien la capacidad de determinar la clasificación viene dada por el conjunto de ellos. Precisamente por esto, es complicado encontrar al 'conjunto ideal' de descriptores moleculares, ya que influirán en una clasificación peor o mejor dependiendo de cuales sean el resto de descriptores. Además, añadir tantos descriptores como se pueda no es la solución, ya que muchos de ellos están íntimamente relacionados y utilizarlos al mismo tiempo podría provocar problemas de colinearidad. Se han probado muchas combinaciones de número y tipo de descriptores moleculares, y la que se ha explicado anteriormente (y se ha empleado en este proyecto) es una que, en este caso, parece funcionar correctamente. Sería interesante encontrar el set de descriptores moleculares más apropiados para esta tarea, pero esto podría requerir suficiente trabajo como para un proyecto independiente completo.

El resultado de la segunda sección del programa es, por lo tanto, una matriz de N (número de compuestos activos y *decoys*) filas por 13 columnas (12 descriptores moleculares y la clase a la que pertenecen, 1:compuesto activo o 0:decoy).

En la tercera sección se utiliza parte de este set de datos para entrenar el modelo, y el resto para comprobarlo. El subconjunto utilizado para entrenar el modelo se denomina *training dataset* y está formado por el 80% de todos los

compuestos (activos y *decoys*), muestreados aleatoriamente. Con sus descriptores moleculares y la clase a la que pertenecen, el algoritmo SVM crea un modelo (calcula el hiperplano que separa óptimamente los dos grupos). A continuación se utiliza el resto de los datos (*testing dataset,* el 20% restante) sin las etiquetas de clase, a fin de que el modelo pueda predecir si interaccionan o no con el *target* seleccionado. Como la clase real del grupo de *testing* es ya conocida, se pueden comparar con las predicciones y construir una matriz de confusión, como la que se muestra a continuación.

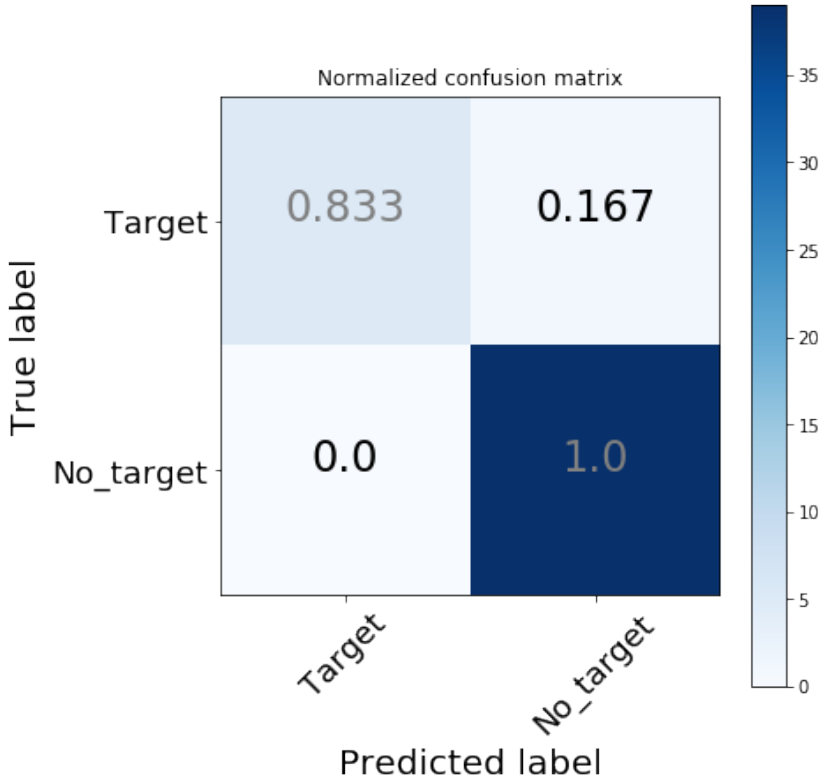


Ilustración 16: Matriz de confusión de predicción del grupo de testing

La interpretación es como sigue: el 83% de los compuestos activos han sido clasificados correctamente, mientras que el 17% de estos ha sido erróneamente clasificado como compuesto inactivo; el 100% de los compuestos no activos ha sido clasificado correctamente. A partir de esta matriz de confusión, que muestra los verdaderos positivos (arriba a la izquierda), verdaderos negativos (abajo a la derecha), falsos positivos (abajo a la izquierda) y falsos negativos (arriba a la derecha), se pueden calcular varios métricos que valoran el predictor, tales como la *accuracy* (porcentaje de predicciones correcta), tasa de falsos positivos, y tasa de falsos negativos entre otros. Uno puede utilizar estas medidas para encontrar los parámetros óptimos para el SVM.

Uno de los parámetros más importantes de los SVM, tal y como se explicaba en la sección 3.1, es qué *kernel* utiliza. Cada *kernel* realiza una transformación distinta en los datos, por lo suele haber un *kernel* específico que sea el óptimo para un conjunto de datos determinado. Además, cada *kernel* suele tener parámetros adicionales, que controlan cómo se realiza la transformación, el peso (weight) de cada punto de los datos, funciones de flexibilidad que evitan sobreajustar el modelo mediante la inclusión de una 'tolerancia al error', etc.

Por supuesto, cada parámetro no puede optimizarse por sí solo, ya que todos influyen en el modelo. Por ello deben encontrarse la combinación de parámetros que mejor funcione. Como este puede ser un proceso tedioso, muchos paquetes de *machine learning* incluyen rutinas para realizar este proceso de forma bastante automatizada. Uno determina una serie de parámetros que se quieren probar, junto con una serie de valores posibles para cada parámetro, y el programa prueba todas las posibles combinaciones y devuelve aquella que mejor optimiza para un métrico determinado, comúnmente, *accuracy* (porcentaje de predicciones correctas).

En el programa 1_machine_learning_target_prediction se incluye una celda con el fragmento de código empleado para escoger los parámetros más adecuados. Este, como se explicaba en el párrafo anterior, toma una serie de parámetros con una serie de posibles valores, como sigue

```
tuned_parameters = [
   {
      'kernel' : ['rbf', 'linear', 'sigmoid', 'poly'],
      'gamma' : [0.25, 0.5, 1e-1, 1e-2, 1e-3, 1e-4],
      'C' : [1, 10, 50, 100, 1000]
   }]
```

y devuelve la combinación de parámetros que mejor métrico elegido presenta (en este caso, *accuracy* o porcentaje de predicciones correctas).

```
Best parameters set found on development set:
{'kernel': 'rbf', 'C': 50, 'gamma': 0.001}
```

Debido a que el algoritmo utiliza muestreos aleatorios (*cross validation*) en el proceso, los parámetros óptimos pueden cambiar. En este caso concreto, el parámetro que permanece siempre como el mejor es el *kernel*, que entre los cuatro posibles, siempre aparece como *mejor rbf* (*radial basis function*). El resto de parámetros no parecen influir de una forma clara en el resultado de la clasificación, por lo que para *gamma* y C se han utilizado unos valores relativamente arbitrarios.

No obstante, métricos cómo la *accuracy* (porcentaje de predicciones correctas) pueden ser insuficientes para comparar entre distintos modelos (creados con distintos parámetros) y elegir el mejor. Por ello, es común representar el desempeño de clasificadores binarios (de dos categorías) con curvas ROC.

El resto de secciones del programa incluyen la valoración que se comenta en la siguiente sección, así como el código para generar ciertos gráficos.

3.5 Valoración

Las curvas ROC, acrónimo del inglés *Receiver Operating Characteristic* son una forma de representar un clasificador binario. Consisten en una representación gráfica del ratio de verdaderos positivos (TPR, *True Positive Rate*) respecto al ratio de falsos positivos (FPS*, False Positive Rate*), para distintos umbrales de discriminación. Estos métricos son el cociente entre el número de verdaderos positivos (*true positives*) y el número total de positivos, y el cociente entre el número de falsos positivos (*false positives*) y el número total de negativos, como se indica en las fórmulas a continuación.

$$TPR \mid True\ Positive\ Rate = \frac{TP}{P} \qquad FPR \mid False\ Positive\ Rate = \frac{FP}{N}$$

En lugar de que el algoritmo SVM retorne una clasificación *cruda*, es posible utilizar una opción para que calcule la probabilidad de pertenecer aun una clase determinada (el algoritmo SVM no calcula probabilidades de forma natural como parte del algoritmo, pero algunas implementaciones admiten la estimación de esta probabilidad). Por ejemplo, para una serie de compuestos ficticios, en la siguiente tabla se muestra la probabilidad calculada de que pertenezcan a la clase 1 (esto es, interaccionan con el *target*), entendiendo que ‘por defecto’ pertenecen a la clase 0 (no interaccionan con el *target*).

	Probabilidad (clase=1)	Clase verdadera
Compuesto 1	0.1	0
Compuesto 2	0.3	0
Compuesto 3	0.4	1
Compuesto 4	0.5	0
Compuesto 5	0.7	1
Compuesto 6	0.8	1
Compuesto 7	0.9	1

.

En la tabla anterior se observa cierta tendencia a que probabilidades altas corresponden con compuestos que pertenecen a la clase 1, y probabilidades bajas corresponden con compuestos que pertenecen a la clase 0. No obstante, es en el término medio donde hay ciertos problemas. Esto sucede porque rara vez un test (de cualquier tipo, computacional o diagnóstico médico) puede distinguir perfectamente entre dos grupos, ya que siempre ha cierto solapamiento. En función de qué interese más, si limitar número de falsos positivos o limitar número de falsos negativos, uno puede fijar un umbral de probabilidad por encima del cual se entenderá que queda asignada la clase 1.

Las curvas ROC se construyen calculando la tasa de falsos positivos y de verdaderos positivos para una serie de umbrales distintos, tal y como se observa en la tabla siguiente, en la que se calculan estos dos números para unos umbrales (utilizando los datos de la tabla anterior).

Ilustración 17: Ejemplo de construcción de curva ROC

Para acabar de construir la curva ROC, basta con graficar estos puntos y conectarlos para formar la curva como tal.

Cabe destacar que la curva ROC ideal pasa por el punto (0,1; esquina superior izquierda del gráfico), es decir, alcanza la tasa de verdaderos positivos máxima con una tasa de falsos positivos mínima. Esto implicaría que el clasificador es capaz de diferenciar completamente entre ambos grupos, y es lo deseable. Cuanto más se acerque a este punto la curva ROC, esto será indicativo de mejor clasificador. A fin de obtener un valor cuantitativo único, a menudo se calcula la AUC (*Area Under the Curve*), esto es, el área bajo la curva ROC. Esta es 1 como máximo.

Habiendo explicado cómo se construyen e interpretan este tipo de gráficos, procedemos a comentar los distintos modelos que se han probado.

Como el set de compuestos no es particularmente grande (200 *decoys* + 21 compuestos activos en el caso del receptor beta adrenérgico 2), ocurre que el número de compuestos sobre los que se puede probar el clasificador (grupo de *testing*, 20% del total*)* una vez ha sido entrenado con el resto de compuestos (grupo de *training*, 80% del total) es pequeño. Por ello, se ha optado por superponer 10 curvas ROC en cada prueba. Ya que los compuestos son aleatoriamente muestreados entre el cálculo de una curva ROC y la siguiente,

esto aporta una visión más completa (y honesta) del comportamiento del clasificador, es decir, si es consistentemente bueno en los resultados o varía en la calidad de su desempeño.

El primer modelo que se hizo utilizaba un *kernel* de tipo linear. Como se puede apreciar, es un modelo consistentemente bueno, ya que todas las curvas pasan cerca del punto (0,1) y las AUC asociadas son cercanas a 1 (de hecho, varias valen 1).

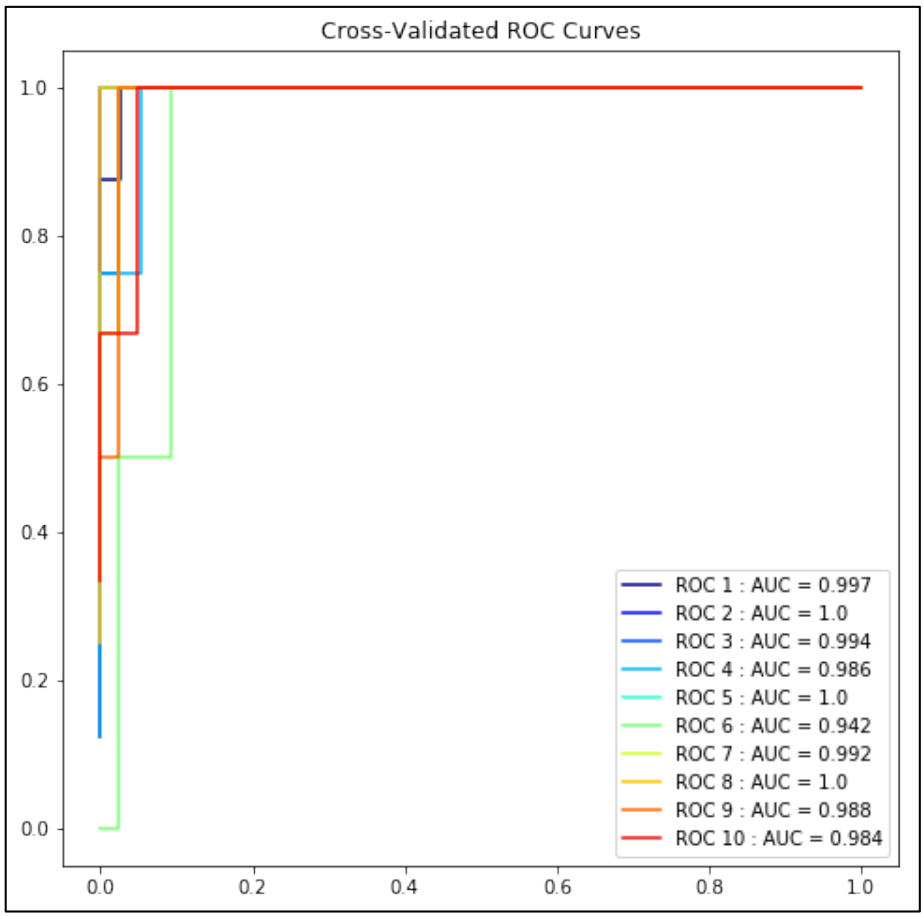
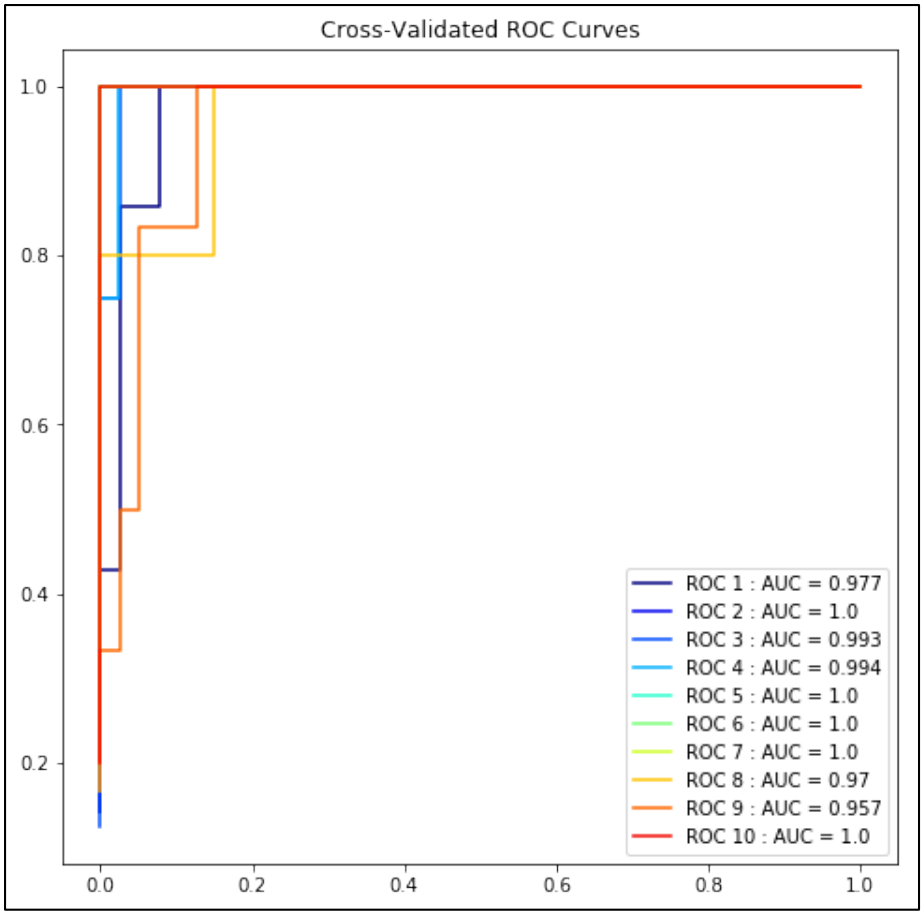


Ilustración 18: Curvas ROC de kernel 'linear'

Las siguientes curvas ROC corresponden con los *kernels* de tipo *poly* y *sigmoid*, respectivamente. Mientras el *kernel poly* funciona prácticamente tan bien como el *linear*, es evidente que el *sigmoid* no es adecuado para este set de datos en concreto. Además, este último ilustra perfectamente el por qué es útil mostrar 10 curvas ROC al mismo tiempo, y es que si solo se mostrase una, podría darse el caso de que por azar fuera una de las buenas.



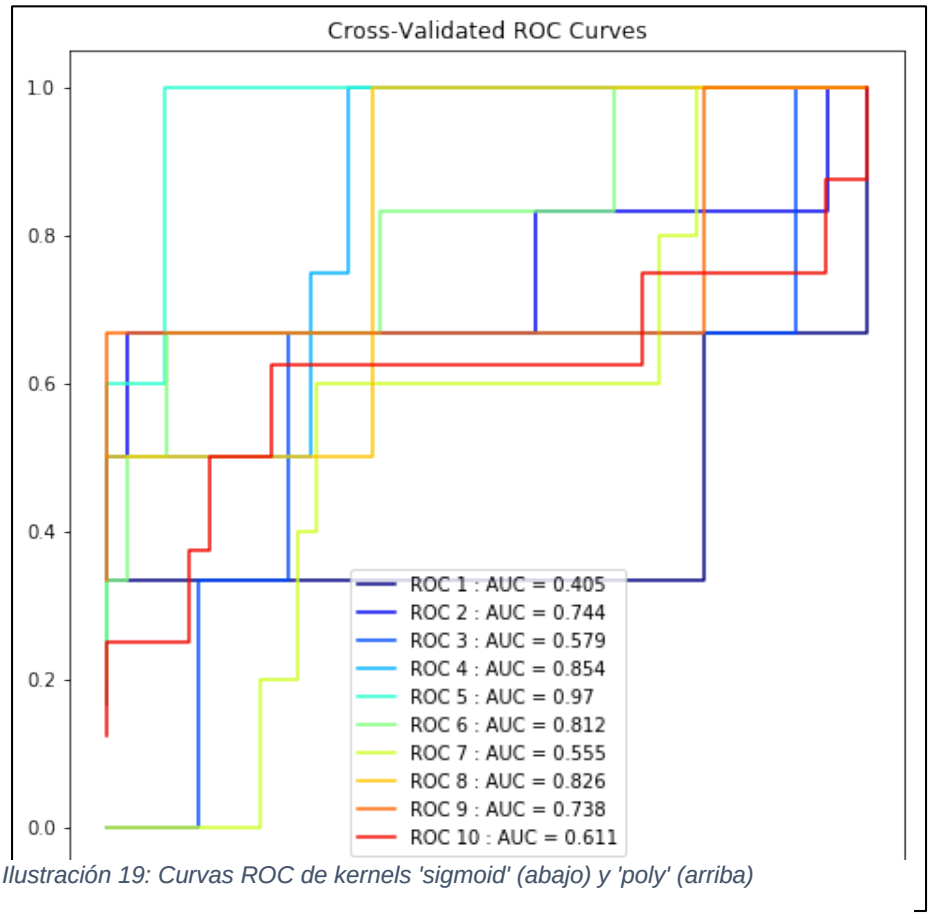


Ilustración 19: Curvas ROC de kernels 'sigmoid' (abajo) y 'poly' (arriba)

Por último, y como *kernel* que mejor funciona, tenemos el rbf o *radial basis function*. Si bien el gráfico no parece muy distinto, observando los valores AUC se aprecia que en el caso de este *kernel* son consistentemente más altos que en el caso de los *kernels poly* y *linear*. Esto no es una sorpresa, ya que mediante la optimización de parámetros ya se había determinado que este era el mejor *kernel*.

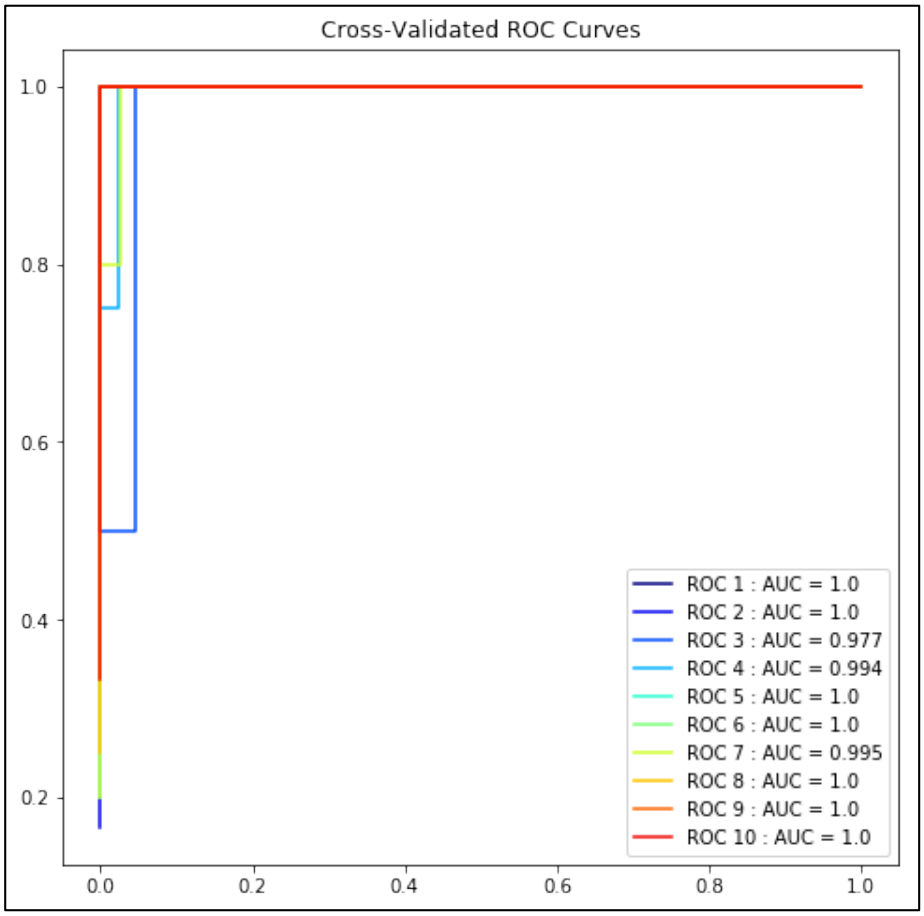


Ilustración 20: Curvas ROC de kernel rbf

Por último, el programa 1_machine_learning_target_prediction incluye una sección dedicada a realizar un test 'real' con el programa. Una vez entrenado con el set de datos inicialmente construido, se prueba a utilizarlo como una aplicación de *screening* virtual real, con 1000 compuestos de ChEMBL. La *accuracy* de este *virtual screening* es de 0.906, y la matriz de confusión se muestra a continuación.

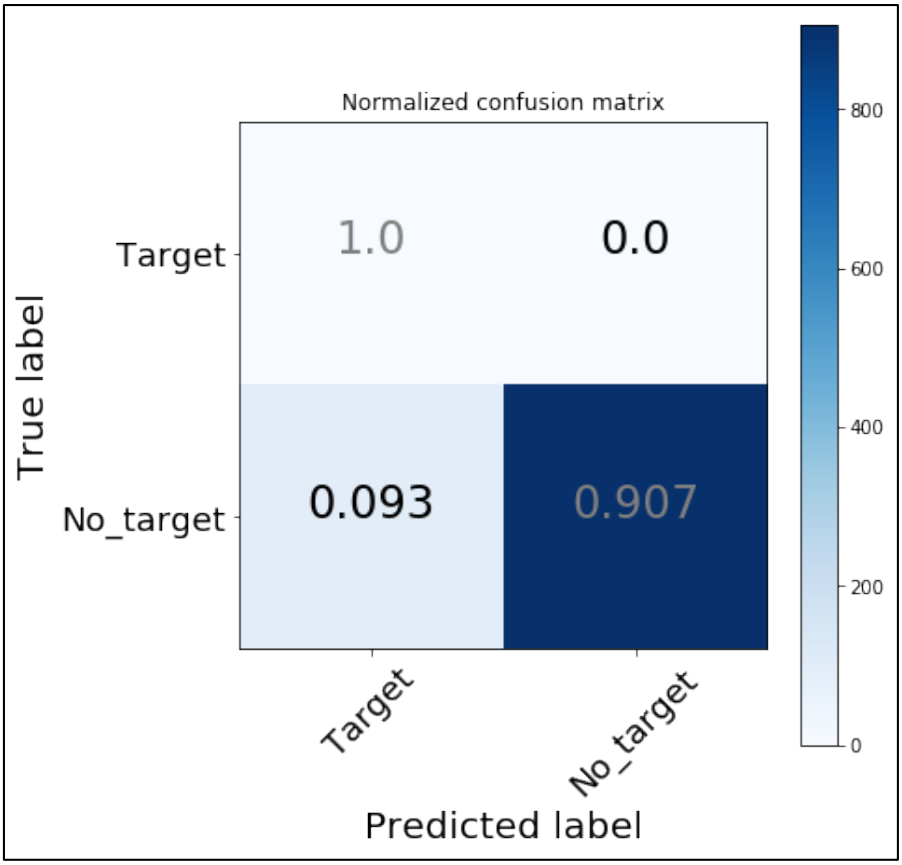


Ilustración 21: Matriz de confusión de test con ChEMBL

Cabe destacar que había un solo compuesto activo de entre los 1000 compuestos que han sido predichos, y que los otros 999 eran compuestos inactivos. Es interesante observar que el único compuesto que interacciona con el *target* (receptor adrenérgico beta2) ha sido correctamente predicho como tal, y que la fuente de error son todos los falsos positivos. Desde luego esta situación es preferible a la contraria (predecir incorrectamente compuestos inactivos como compuestos activos), si bien sería interesante comprobar cómo se puede optimizar la predicción cambiando el umbral de probabilidad para limitar el número de falsos positivos (sin incrementar el número de falsos negativos).

4. Conclusiones

En cuanto a las lecciones aprendidas de la primera parte del proyecto, la aproximación mediante búsqueda de similaridad: ha sido interesante aprender y utilizar las distintas técnicas de determinar la similaridad química. Determinar el grado de similitud entre dos moléculas es un problema mucho más complejo de lo que podría parecer a priori, y esto es un concepto que no me había planteado antes de este proyecto. En cuanto a las lecciones aprendidas de la segunda parte del proyecto, la aproximación mediante *machine learning*: Esta ha sido la primera vez que he aplicado las técnicas aprendidas de *machine learning* a un problema 'real', y debo decir que estoy muy satisfecho con los conocimientos adquiridos durante la asignatura homónima el trimestre pasado. Descubrir el mundo de los descriptores moleculares, su importancia e historia ha sido muy interesante. En cuanto a lecciones aprendidas durante el conjunto del proyecto: sin duda, la lección más importante que he aprendido es la importancia de planificar y organizar adecuadamente un proyecto. Por otra parte, nunca había utilizado Python para el desarrollo de un proyecto científico de estas características, y he de decir que trabajar con la interfaz Jupyter notebook es particularmente agradable e interactivo, con muchas posibilidades de cara a presentar el trabajo realizado de forma interactiva o a distribuir ejemplos.

Los objetivos planificados en la primera PEC de este proyecto han ido adaptándose al entendimiento que se ha ido adquiriendo sobre las herramientas que se han utilizado y sobre el mismo proyecto. Este fue definido de manera bastante flexible al principio, y algunos de los objetivos han sido probados no ser realistas. Por ejemplo, antes de verdaderamente entender qué datos se iban a utilizar y cuál iba a ser el resultado del modelo de *machine learning*, se hacía referencia al objetivo 'incluir en el modelo el resto de *targets'*, algo que, ahora es evidente, no es viable en este contexto. Al margen de esto, todos los objetivos han sido cumplidos de forma satisfactoria. Quizás el único objetivo que pueda considerarse queda parcialmente pendiente sería la evaluación exhaustiva del método de predicción por búsqueda de similaridad química. Sucede que, como ya se ha explicado, la idea de este método funciona, pero es completamente dependiente de lo completa que sea la base de datos. En este sentido, cualquier valoración numérica de la calidad del predictor estaría indirectamente evaluando la calidad del set de datos construido, que en principio es bastante limitado, ya que se ha utilizado una sola base de datos, y una serie de requerimientos (que tenga alguno de los campos de interés), que han hecho que la cuenta total de fármacos en la tabla de compuestos sea 3455, la cual es relativamente baja comparada con la cuantía total de compuestos bioactivos que existen.

Con respecto a la metodología empleada, decir que uno de los mayores riesgos del proyecto era que la tarea de aprender a utilizar los distintos módulos de Python especializados (rdkit, scikit-learn, pandas…) fuera demasiado difícil o costase demasiado tiempo. En este sentido, decir que, de acuerdo a lo planificado desde el principio del proyecto, los tiempos dedicados a aprender las herramientas y los fundamentos del proyecto han sido razonables, y no han supuesto en ningún caso 'perder tiempo de trabajo'. Respecto a la planificación

temporal, decir que se ha cumplido de forma aproximada, destacando una semana a finales de abril que se tuvo que abandonar el proyecto por un viaje imprevisto. No obstante, como el trabajo se había adelantado antes de este imprevisto, no ha habido necesidad de acortar el proyecto inicialmente planificado.

Mientras que en el apartado de la búsqueda por similaridad no hay mucho más que se pueda aportar al proyecto (al margen de ampliar la base de datos, pero eso en sí no es un objetivo meramente técnico), respecto al apartado de *machine learning* han quedado varias cuestiones que, de haber tenido más tiempo, me hubiera gustado explorar. La primera es el hecho de crear un solo modelo para varios *targets* (predicción multiclase). La implementación de SVM en scikit-learn soporta esta funcionalidad, y es algo que sería muy interesante haber probado. Por otra parte, me gustaría haber hecho un análisis estadístico de cómo afecta la inclusión de distintos descriptores moleculares al desempeño del modelo de predicción. De hecho, esto podría utilizarse para determinar un conjunto de descriptores moleculares óptimos para esta tarea de clasificación, en lugar de utilizar un conjunto porque 'se ha probado y funciona'.

El valor que personal y críticamente doy a este trabajo no es tanto por la innovación técnica o el desarrollo de nuevas ideas o estrategias, sino más bien por el componente instructivo. Elegí este proyecto de fin de máster porque el trimestre anterior había cursado la asignatura *machine learning* y quería tener la oportunidad de aplicarlo a un problema 'real'. Por otra parte, la quimioinformática es un área del conocimiento que jamás he estudiado ni llegado a conocer, y tenía mucho interés en introducirme en ese tema. En este sentido, el hecho de aprender a la par que he ido desarrollando el proyecto ha sido extremadamente satisfactorio y productivo. El hecho de ser un proyecto de fin de máster a distancia me ha ofrecido la oportunidad y el desafío de desarrollar un proyecto de forma mucho más independiente que otros proyectos que he realizado hasta ahora, lo cual creo, me ha hecho crecer como persona, y como científico.

5. Glosario

Diana molecular / *target*: Molécula biológica a la cual se une un compuesto químico.

Índice de similaridad de Tanimoto: Una forma de medir la la similaridad entre dos *fingerprints* de compuestos químicos

Docking decoy: “Señuelo de acoplamiento molecular”, son compuestos químicos que pueden ser reales o virtuales, diseñados para probar sistemas de predicción de acoplamiento molecular.

False Positive Rate (FPR): Es una medida de la proporción de falsos positivos esperados. Se obtiene dividiendo el número de falsos positivos por el número de negativos reales.

Fingerprint: Literalmente ‘huella dactilar’, se refiere al cálculo de una serie de características de una molécula, que como la huella dactilar en humanos, sirve para identificar compuestos químicos, pero no para reconstruir el compuesto al completo.

Machine learning: Conjunto de algoritmos e ideas que permiten hacer programas que ‘aprendan’ de ejemplos de datos para después realizar tareas de predicción o clasificación.

Matriz de confusión: Una forma de representar una predicción clasificatoria, en términos de ‘clase real’ y ‘clase predicha’.

Support Vector Machine (SVM): Un algoritmo de *machine learning*

True Positive Rate (TPR): Es una medida de la proporción de verdaderos positivos esperados. Se obtiene dividiendo el número de verdaderos positivos entre el número de positivos reales.

6. Bibliografía

[1] H. Kubinyi, «Chemical similarity and biological activities,» *Journal of the Brazilian Chemical Society, 13(6),* pp. 717-726, 2002.

[2] «Primer contacto con Rdkit,» [En línea]. Available: http://www.rdkit.org/new_docs/GettingStartedInPython.html. [Último acceso: 03 2017].

[3] «Support vector machines with Scikit-learn,» [En línea]. Available: http://scikit-learn.org/stable/modules/svm.html. [Último acceso: 04 2017].

[4] D. P. Leach, Introduction to chemoinformatics, Kluwer Academic Publishers, 2003.

[5] D. Bajusz, «Why is Tanimoto index and appropriate choice for fingerprint-based similarity calculations?,» *J Cheminform.,* vol. 7:20, 2015 May.

[6] A. G. A. H. L. B. J. C. M. D. F. K. Y. L. A.P. Bento, «The ChEMBL bioactivity database: an update,» *Nucleic Acids Res,* vol. 42, pp. 1083-1090, 2014.

[7] V. C. Roberto Todeschini, Handbook of MOlecular Descriptors, Wiley-VCH, 2000.

[8] C. M. I. J. S. B. J. Mysinger MM, «Directory of Useful Decoys, Enhanced (DUD-E): Better Ligands and Decoys for Better Benchmarking,» *Journal of Medicinal Chemistry,* vol. 55, nº 14, pp. 6582-6594, 2012.

[9] «Package rdkit :: Package Chem :: Modules Descriptors,» [En línea]. Available: http://www.rdkit.org/Python_Docs/rdkit.Chem.Descriptors-module.html. [Último acceso: 05 2017].

[10] A. Balaban, «Highly discriminating distance-based topological index,» *Chemical Physics Letters,* vol. 89, nº 5, pp. 399-404, 1982.

7. Anexos

1_similarity_search

```
#Entrar en el directorio donde se encuentran los archivos csv
%cd /home/mikeleitor/Escritorio/Pipeline/usable_csv/
%ls

#por convenio, al importar pandas se le asigna el nombre "pd"
import pandas as pd

#columnas molregno; código SMILES
df_compound_structures = pd.read_csv('compound_structures.csv',
                                     usecols=['molregno',
                                              'canonical_smiles'])
#columnas molregno; MESH_HEADING
df_drug_indication = pd.read_csv('drug_indication.csv')

#columnas molregno; MECHANISM_OF_ACTION
df_drug_mechanism = pd.read_csv('drug_mechanism.csv')

#columnas molregno; pref_name
df_molecule_dictionary = pd.read_csv('molecule_dictionary.csv',
                                     usecols=['molregno',
                                              'pref_name'],
                                     dtype={'pref_name':object})

#columnas tid; pref_name
df_target_dictionary = pd.read_csv('target_dictionary.csv')

#columnas molregno; TID
df_drug_mechanism_tid = pd.read_csv('drug_mechanism_targets.csv')

#Hay registros con efectos o indicaciones pero no presentes en la
#tabla compound_structures!
print "Registros totales de compound_structures :",
str(len(df_compound_structures))
print "****"
print "Registros en drug_indication presentes en compound_structures
:",
sum(df_drug_indication['molregno'].isin(df_compound_structures['molreg
no']))
print "Registros totales en drug_indication :",
len(df_drug_indication)
print "Registros en drug_mechanism presentes en compound_structures
:",
sum(df_drug_mechanism['molregno'].isin(df_compound_structures['molregn
o']))
print "Registros totales en drug_mechanism :", len(df_drug_mechanism)
print "*******"
print "Registros totales en target_dictionary :",
len(df_target_dictionary)
```

```
print df_compound_structures.shape
print df_drug_indication.shape
print df_drug_mechanism.shape

#Paso 1 -> Una tabla de registros que tengan efectos O indicaciones
temp_mechanism_indication1 = pd.merge(df_drug_indication,
                                      df_drug_mechanism,
                                      left_on=['molregno'],
                                      right_on=['molregno'],
                                      how='outer')

#Paso 2 -> Una tabla con efectos O indicaciones | y un TARGET_ID
temp_mechanism_indication2 = pd.merge(temp_mechanism_indication1,
                                      df_drug_mechanism_tid,
                                      left_on=['molregno'],
                                      right_on=['molregno'],
                                      how='outer')

#Paso 3 -> Una tabla con efectos O indicaciones | TARGET_ID y código
SMILES
temp_table = pd.merge(df_compound_structures,
                      temp_mechanism_indication2,
                      left_on=['molregno'],
                      right_on=['molregno'],
                      how='right')

#Paso 4 -> Una tabla con efectos o indicaciones | TARGET_ID, código
SMILES
#y nombre del compuesto
full_table = pd.merge(df_molecule_dictionary,
                      temp_table,
                      left_on=['molregno'],
                      right_on=['molregno'],
                      how='right')

#definimos un diccionario que relacione {tid : nombre_target}
tid2target_dict = dict(zip(df_target_dictionary['tid'],
                           df_target_dictionary['pref_name']))

#algunos compuestos tienen mas de un target_id (tid), y no se
#convertirlos directamente a dos nombres de target en pandas por ello,
#creamos la funcion convert, que realiza esto automaticamente para una
#linea (tecnicamente, serie) del dataframe de pandas

def convert(tid):
    try:
        #Si hay mas de un registro
        if ';' in tid:
            #crear una lista, luego un string, y separarlos por ";" ->
            #presenta la columna de pandas en formato adecuado
            temp_tid_list = "".join(list(test_tid)).split(";")

            #devuelve la lista de nombres de target para cada tid
            return [tid2target_dict[int(x)] for x in temp_tid_list]

        #si no hay mas de un registro, devuelve un string con el
#nombre del target
        return tid2target_dict[int(tid)]
    except:
        #si no se conoce el nombre del target, o no el compuesto no
#tiene un target_id asociado
```

```
        return 'Unknown'

#columna 'target' se crea aplicando la funcion convert a toda la
#columna de TIDs
target = full_table['tid'].apply(convert)

#la columna 'target se incluye en la tabla de compuestos
full_table['target'] = pd.DataFrame(target)

#se exporta la tabla creada a un archivo csv, que puede ser cargado
#directamente si se ejecuta este programa por segunda vez o para otros
#programas
full_table.to_csv('FULL_TABLE_COMPOUNDS.csv', header=True)

#importamos funcionalidad base de rdkit
from rdkit import Chem

#SE RETIRAN LOS REGISTROS QUE CAREZCAN DE CODIGO SMILES
full_table = full_table[full_table['molregno'].isin(
    df_compound_structures['molregno'])]

#Por otra parte, se convierten los codigos SMILES en objetos de tipo
#molecula
table_molecules = [Chem.MolFromSmiles(x) for x in
full_table['canonical_smiles']]

#Aqui se obtiene el codigo SMILES de la molecula problema y se
#convierte a la clase 'molecule'

#en este caso se ha utilizado el codigo SMILES de la Simvastatina
#(inhibe produccion de colesteron endogeno)
query_smiles =
"[H][C@]12[C@H](C[C@@H](C)C=C1C=C[C@H](C)[C@@H]2CC[C@@H]1C[C@@H](O)CC(
=O)O1)OC(=O)C(C)(C)CC"
query_molecule = Chem.MolFromSmiles(query_smiles)

#importamos funcion para crear fingerprints a partir de 'molecules'
from rdkit.Chem.Fingerprints import FingerprintMols

#Se convierte la tabla de moleculas, asi como la molecula problema,
#a fingerprints
table_fingerprints = [FingerprintMols.FingerprintMol(x) for x in
table_molecules]
query_fingerprint = FingerprintMols.FingerprintMol(query_molecule)

#importamos modulo para calcular distancias entre fingerprints
from rdkit import DataStructs

#el dataframe "comparisons" se obtiene a partir de comparar el
#fingerprint de la molecula problema con el resto de fingerprints
#de la base de datos
comparisons = pd.DataFrame(
    [DataStructs.FingerprintSimilarity(x, query_fingerprint)
    for x in table_fingerprints],
    columns=['similarity'])

#este el el umbral de similaridad minimo para resultados.
#Recomendado: >= 0.7
threshold = 0.7

#se seleccionan aquellos registros con una similaridad superior
```

```
#al umbral establecido
selection = comparisons['similarity'] > threshold
predicted_sim = full_table[selection.values]
sim_measure = comparisons[selection.values]

#para unir la medida de similaridad con el resto de la tabla,
#se crea un nuevo indice
new_index = range(len(sim_measure))
sim_measure = sim_measure.set_index([new_index])
predicted_sim = predicted_sim.set_index([new_index])

#se unen en un solo dataframe la medida de similaridad con los datos
#de los compuestos que superan el umbral
result_table = pd.concat([sim_measure, predicted_sim], axis=1)

#mostrar resultados -> moleculas parecidas a la introducida como QUERY
#al principio
result_table

#Import modules for molecule drawing
from rdkit.Chem import Draw

#Import modules to show all the outputs
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

#draw the query molecule
Draw.MolToImage(query_molecule, size=(400,400), fitImage=True)

#draw all the resulting molecules with their names
result_molecules = [Chem.MolFromSmiles(x) for x in
result_table['canonical_smiles']]
nombres = result_table['pref_name']
Draw.MolsToGridImage(result_molecules,
                     molsPerRow=3,
                     subImgSize=(300,300),
                     legends=nombres)
```

2_similarity_search_evaluation

```
#Entrar en el directorio donde se encuentran los archivos csv
%cd /home/mikeleitor/Escritorio/Pipeline/usable_csv/
%ls

#por convencion, pandas se importa con el renombre pd
import pandas as pd

#cargamos la tabla generada en el programa numero 1
full_table = pd.read_csv('FULL_TABLE_COMPOUNDS.csv', index_col=0)

#Cargamos las funciones y librerias necesarias de rdkit
from rdkit import Chem
from rdkit.Chem.Fingerprints import FingerprintMols
from rdkit import DataStructs

#comprobamos que la tabla ha sido correctamente cargada
full_table.head(3)
```

```
#escogemos n (10) registros de forma aleatoria y hacemos un nuevo
dataset
sample_dataset = full_table.sample(n=10)

#hacemos otro dataset con todos los demas
test_dataset =
full_table.loc[~full_table.index.isin(sample_dataset.index)]

#print full_table.shape
#print sample_dataset.shape
#print test_dataset.shape

#instanciamos la clase 'molecule' con los codigos SMILES
#test_molecules = [Chem.MolFromSmiles(x) for x in
sample_dataset['canonical_smiles']]
table_molecules = [Chem.MolFromSmiles(x) for x in
test_dataset['canonical_smiles']]

#calculamos los fingerprints para el test_dataset
table_fingerprints = [FingerprintMols.FingerprintMol(x) for x in
table_molecules]

#para cada uno de las n moleculas aleatorias
for query_molregno in sample_dataset['molregno']:

    #se extrae el registro completo (SMILES, efectos, indicaciones,
target)
    query_record =
sample_dataset[sample_dataset['molregno']==query_molregno]

    #List comprehension porque no funciona de otra manera ;-;
    #Instanciamos molecula y calculamos fingerprint
    query_molecule = [Chem.MolFromSmiles(x) for x in
query_record['canonical_smiles']]
    query_fingerprint = [FingerprintMols.FingerprintMol(x) for x in
query_molecule]

    #creamos la tabla de comparaciones
    table_comparisons = pd.DataFrame(
        [DataStructs.FingerprintSimilarity(query_fingerprint[0], x)
for x in table_fingerprints],
        columns=['similarity'])

    #establecemos un umbral de similaridad minima
    threshold = 0.8

    #seleccionamos las moleculas que superen la similaridad minima
establecida
    selection = table_comparisons['similarity'] > threshold
    predicted_records = test_dataset[selection.values]

    #PROCESAMIENTO PARA LA PRESENTACION DE RESULTADOS

    #Se cogen las indicaciones unicas, y se procesan para mostrarlas
    #en un formato limpio
    indication= pd.unique(predicted_records['mesh_heading'])
    indication = list(indication)
    indication = ";".join(str(v) for v in indication)
    indication = indication.replace(",", "-")
    indication = indication.replace("; ", ";")
    indication = indication.split(";")
```

```
    indication = list(set(indication))

    #One-liner para quitar cualquier 'nan'
    while 'nan' in indication: indication.remove('nan')

    #Se cogen los efectos, y se procesan para mostrarlos
    #en un formato limpio
    effect = pd.unique(predicted_records['mechanism_of_action'])
    effect = list(effect)
    effect = ";".join(str(v) for v in effect)
    effect = effect.replace(",", "-")
    effect = effect.replace("; ", ";")
    effect = effect.split(";")
    effect = list(set(effect))
    while 'nan' in effect: effect.remove('nan')


    #Se cogen las indicaciones y efectos unicos REALES del compuesto
    #seleccionado, y se procesan para mostrarlos en un formato limpio

    actual_indication = list(query_record['mesh_heading'])
    actual_indication = ";".join(str(v) for v in actual_indication)
    actual_indication = actual_indication.replace(",", "-")
    actual_indication = actual_indication.replace("; ", ";")
    actual_indication = actual_indication.split(";")
    while 'nan' in actual_indication: actual_indication.remove('nan')

    actual_effect = list(query_record['mechanism_of_action'])
    actual_effect = ";".join(str(v) for v in actual_effect)
    actual_effect = actual_effect.replace(",", "-")
    actual_effect = actual_effect.replace("; ", ";")
    actual_effect = actual_effect.split(";")
    while 'nan' in actual_effect: actual_effect.remove('nan')

    print "ACTUAL Medical indications:"
    if actual_indication == []:
        print "\tNo information available"
    else:
        for ind in actual_indication:
            print "\t" + ind


    print "ACTUAL Mechanism:"
    if actual_effect == [] or actual_effect == ['']:
        print "\tNo information available"
    else:
        for eff in actual_effect:
            print "\t" + eff

    print "//////////^\\\\\\\\\\ "

    print "Molecule registry number: ", str(query_molregno)
    print "Medical indications:"
    if indication == ['']:
        print "\tNo information available"

    for ind in indication:
        print "\t" + ind

    print "Mechanism"
    if effect == ['']:
```

```
        print "\tNo information available"

    for eff in effect:
        print "\t" + eff
    #print indication
    #print effect
    print
"\n#####################################################################\n"
```

1_machine_learning_target_prediction

```
#directorio en el que se encuentra el archivo 'FULL_TABLE_COMPOUNDS.csv', generado en el script anterior
%cd /home/mikeleitor/Escritorio/Pipeline/usable_csv/
%ls

# importamos la libreria pandas, que se encarga de proporcionar la estructura logica para leer,
#>almacenar, y manipular datos
import pandas as pd

#cargamos la tabla FULL_TABLE_COMPOUNDS,
df_compounds_table = pd.read_csv('FULL_TABLE_COMPOUNDS.csv', index_col=0)

#mostramos cuantos targets tienen mas de 20 compuestos asociados
targets = df_compounds_table['target']
count_targets = df_compounds_table.groupby('target').count()
count_targets[count_targets['molregno'] > 20]

#seleccionamos los compuestos activos, que son los que figuran como tales en ChEMBL
#chembl_compounds = df_compounds_table[df_compounds_table['target']=='Beta-2 adrenergic receptor']['canonical_smiles']

chembl_compounds = df_compounds_table[df_compounds_table['target']=='Beta-2 adrenergic receptor']['canonical_smiles']


#CARGAR DATOS DE 'DUDE' PARA RECEPTOR BETA ADRENERGICO 2

#directorio donde se encuentran los compuestos
#%cd /home/mikeleitor/Escritorio/beta_adrenergic2\ receptor
#%ls
%cd /home/mikeleitor/Escritorio/beta_adrenergic2\ receptor

#cargamos compuestos activos y decoys, y mostramos cuantos hay de cada tipo
import pandas as pd

df_actives = pd.read_csv('actives_final.ism', sep=" ", header=None)
```

```
df_actives.columns = ['smiles', 'unknown_id', 'chembl_id']

df_decoys = pd.read_csv('decoys_final.ism', sep=" ", header=None)
df_decoys.columns = ['smiles', 'unknown_id2']

print df_actives.shape
print df_decoys.shape

#df_actives_sample = pd.DataFrame.sample(df_actives, n=,replace=False) //no
se utilizaran los compuestos activos de DUDE

#se utilizaran todos los compuestos activos disponibles en ChEMBL
df_actives_sample = chembl_compounds

#se utilizara una muestra de n=200 de los decoys disponibles en DUDE
df_decoys_sample = pd.DataFrame.sample(df_decoys, n=200, replace=False)


print df_actives_sample.shape
print df_decoys_sample.shape

#importamos rdkit, funcionalidad de calcular descriptores moleculares
from rdkit.Chem import Descriptors

#importamos numpy para manipular matriz de 'features'
import numpy as np

#la funcion calculate_molecular_descriptors toma una lista de moleculas como
argumento, y devuelve un array con
#> una fila por molecula, y una columna por descriptor molecular. Estos
descriptores se especifican en el diccionario
#> descriptor_funcs, dentro de la funcion.
def calculate_molecular_descriptors(list_molecules):
    descriptor_funcs = {1: Descriptors.TPSA,
                        2: Descriptors.BalabanJ,
                        3: Descriptors.MaxPartialCharge,
                        4: Descriptors.EState_VSA1,
                        5: Descriptors.MolLogP,
                        6: Descriptors.ExactMolWt,
                        7: Descriptors.EState_VSA2,
                        8: Descriptors.fr_Al_OH,
                        9: Descriptors.fr_aldehyde,
                        10: Descriptors.Ipc,
                        11:Descriptors.Kappa1,
                        12:Descriptors.MinPartialCharge

                       }

    #para cada funcion de descriptor_funcs
    for func_n in range(1,len(descriptor_funcs)+1):
```

```
        #si es la primera funcion, crear una lista con el calculo de ese descriptor molecular para cada compuesto
        if func_n==1:
            descriptor_array = [descriptor_funcs[func_n](x) for x in list_molecules]

        #si no es la primera funcion, calcula una lista con el calculo del nuevo descriptor molecular, y lo adjunta
        #> a lo que ya se habia calculado
        else:
            new_vector = [descriptor_funcs[func_n](x) for x in list_molecules]
            descriptor_array = np.column_stack([descriptor_array, new_vector])

    return descriptor_array

import numpy as np
from rdkit import Chem

#convertimos codigos SMILES a instancias de la clase 'molecule'
list_active_molecules = [Chem.MolFromSmiles(x) for x in df_actives_sample]
list_decoy_molecules = [Chem.MolFromSmiles(x) for x in df_decoys_sample['smiles']]

#creamos listas con la 'clase' de ls moleculas
class_active = [1]*len(list_active_molecules)
class_decoy = [0]*len(list_decoy_molecules)

#calculamos los arrays de descriptores moleculares
active_array = calculate_molecular_descriptors(list_active_molecules)
decoy_array = calculate_molecular_descriptors(list_decoy_molecules)

#incluimos una primera columna con la 'clase' del compuesto, 0 si es un decoy y 1 si es compuesto activo
active_array_with_id = np.column_stack([class_active, active_array])
decoy_array_with_id = np.column_stack([class_decoy, decoy_array])

#cargamos el modelo de preprocesado de datos de sklearn
from sklearn import preprocessing

#unimos los arrays de features de compuestos activos e inactivos (SIN LOS IDs de clase [0,1]!)
full_array = np.concatenate((active_array, decoy_array))

#escalamos la matriz de features // esto puede dar un mensaje de alerta, pero no da problemas
full_array_norm = preprocessing.scale(full_array)

#anadimos una columna con los identificadores de clase
ids = class_active + class_decoy
full_array_norm_with_id = np.column_stack((ids, full_array_norm))
```

```
#cargamos la funcion para muestrear aleatoriamente los grupos de training y testing
from sklearn.model_selection import train_test_split
#cargamos la clase svm, que se utilizara para implementar un support vector machine
from sklearn import svm

#porcentaje de dataset que sera utilizado para training (testing = 1 - training)
train_percentage = 0.8
train_array_norm, test_array_norm = train_test_split(full_array_norm_with_id, test_size=1-train_percentage)

#la 'clase' esta codificada en la primera columna
y = train_array_norm[:,0]

#el resto de features
X = train_array_norm[:,1:]

#instanciamos la clase SVC del modelo svm, con los parametros oportunos
clf = svm.SVC(kernel='rbf', C=100, gamma='auto')

#entrenamos el modelo con el dataset de training
clf.fit(X, y)

#para cada uno de los casos del dataset de testing, predecimos su clase
predictions = clf.predict([test_array_norm[x,1:] for x in range(len(test_array_norm))])

#importamos funciones para valorar predicciones
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

#clase predicha
y_pred = predictions

#clase verdadera
y_true = test_array_norm[:,0]

print accuracy_score(y_true, y_pred)
print confusion_matrix(y_true, y_pred, labels=[1, 0])
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn.multiclass import OneVsRestClassifier

import matplotlib.pyplot as plt

from sklearn.metrics import roc_curve
from sklearn.metrics import auc
import numpy as np
```

```
from rdkit import Chem


plt.figure(figsize=(8,8))
cmap = plt.get_cmap('jet')

for i in range(10):

    df_actives_sample = chembl_compounds
    df_decoys_sample = pd.DataFrame.sample(df_decoys, n=200, replace=False)




    #convertimos codigos SMILES a instancias de la clase 'molecule'
    list_active_molecules = [Chem.MolFromSmiles(x) for x in df_actives_sample]
    list_decoy_molecules = [Chem.MolFromSmiles(x) for x in df_decoys_sample['smiles']]

    class_active = [1]*len(list_active_molecules)
    class_decoy = [0]*len(list_decoy_molecules)

    active_array = calculate_molecular_descriptors(list_active_molecules)
    decoy_array = calculate_molecular_descriptors(list_decoy_molecules)

    #active_array_with_id = np.column_stack([class_active, active_array])
    #decoy_array_with_id = np.column_stack([class_decoy, decoy_array])

    full_array = np.concatenate((active_array, decoy_array))
    full_array_norm = preprocessing.scale(full_array)

    ids = class_active + class_decoy
    full_array_norm_with_id = np.column_stack((ids, full_array_norm))


    from sklearn.model_selection import train_test_split
    from sklearn import svm

    train_percentage = 0.8
    train_array_norm, test_array_norm = train_test_split(full_array_norm_with_id, test_size=1-train_percentage)

    #test_array_norm = np.row_stack([active_array_with_id, decoy_array_with_id[:4,:]])
    y_train = train_array_norm[:,0]
    X_train = train_array_norm[:,1:]
    y_true = test_array_norm[:,0]
    X_test = test_array_norm[:,1:]
```

```
    classifier = OneVsRestClassifier(svm.SVC(kernel='rbf', probability=True,
C=100, gamma="auto"))
    y_score = classifier.fit(X_train, y_train).decision_function(X_test)

    fpr, tpr, thresholds = roc_curve(y_true, y_score)
    auc_area = auc(fpr, tpr)

    plt.plot(fpr, tpr, color=cmap(i/10.0), label='ROC '+str(i+1)+' : AUC =
'+str(round(auc_area,3)))

plt.title('Cross-Validated ROC Curves')
plt.legend()
plt.show()

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC

tuned_parameters = [
    {
        'kernel' : ['rbf', 'linear', 'sigmoid'],
        'gamma' : [0.25, 0.5, 1e-1, 1e-2, 1e-3, 1e-4],
        'C' : [1, 10, 100, 1000]
    }]

scores = ['precision', 'recall']

for score in scores:
    print "# Tuning hyper-parameters for %s" % score
    print "."

    clf = GridSearchCV(SVC(C=1), tuned_parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X, y)

    print "Best parameters set found on development set:"
    print clf.best_params_
    print "Detailed classification report:"
    print "."
    print "The model is trained on the full development set:"
    print "The scores are computed on the full evaluation set."
    print ":"
    y_true, y_pred = test_array_norm[:,0], clf.predict(test_array_norm[:,1:])
    print classification_report(y_true, y_pred)

df_compounds_table[df_compounds_table['target']=='Glucocorticoid
receptor'].head(5)

true_class = df_compounds_table['target']=='Beta-2 adrenergic receptor'
```

```
list_chembl_molecules = [Chem.MolFromSmiles(x) for x in df_compounds_table['canonical_smiles']]
array_chembl = calculate_molecular_descriptors(list_chembl_molecules[:1000])

#tenemos una matriz de 1000 compuestos, con 12 descriptores moleculares cada uno
array_chembl.shape

#adjuntamos su clase (si se unen o no al target)
array_chembl_with_class = np.column_stack([true_class[:1000], array_chembl])

#quitamos aquellos registros que dan error al calcular descriptores moleculares
array_chembl = array_chembl_with_class[~np.isnan(array_chembl_with_class).any(axis=1)]

#estos son los que quedan
array_chembl.shape

#realizamos la prediccion para todos los compuestos
prediction = []
for compound in range(len(array_chembl)):
    experimental_array = np.row_stack([full_array, array_chembl[compound,1:]])
    experimental_array_norm = preprocessing.scale(experimental_array)
    prediction.append(clf.predict([experimental_array_norm[221,:]]))

prediction[:10]

#importamos funciones para valorar predicciones
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix

#clase predicha
y_pred = prediction

#clase verdadera
y_true = array_chembl[:,0]

print accuracy_score(y_true, y_pred)
print confusion_matrix(y_true, y_pred, labels=[1, 0])

import matplotlib.pyplot as plt
import itertools

def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
```

```
    """


    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, round(cm[i, j], 3),
                 horizontalalignment="center",
                 fontsize=24,
                 color="gray" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.tick_params(axis='both', labelsize=18)
    plt.ylabel('True label', fontsize=20)
    plt.xlabel('Predicted label', fontsize=20)

# Compute confusion matrix
cnf_matrix = confusion_matrix(y_true, y_pred, [1, 0])
np.set_printoptions(precision=2)
class_names = ["Target", "No_target"]

# Plot non-normalized confusion matrix
plt.figure(figsize=(6,6))
plot_confusion_matrix(cnf_matrix, classes=class_names,
                      title='Confusion matrix')

# Plot normalized confusion matrix
plt.figure()
plot_confusion_matrix(cnf_matrix, classes=class_names, normalize=True,
                      title='Normalized confusion matrix')

plt.show()
```