Empleo de técnicas de Machine Learning para la predicción de propiedades ADME-Tox: Toxicidad

Autor: Alberto Vela Castro

Máster en Bioinformática y Bioestadística

Trabajo Final de Máster Area 1

Tutora: Marta Enciso Carrasco

Profesor responsable de la asignatura: Javier Luis Cánovas Izquierdo

Barcelona, 4 de junio de 2019

Tabla de contenido

Índice de figuras y tablas

Resumen

Existe una creciente predilección por la aplicación de técnicas in silico en el desarrollo y descubrimiento de nuevos fármacos frente a las costosas y laboriosas técnicas de laboratorio. Estas son técnicas de machine learning.

El Trabajo de Final de Máster (TFM) consistirá en analizar cuáles son las mejores técnicas actuales de machine learning para la predicción de la propiedad ADME-Tox, toxicidad. Una vez seleccionadas se realizará una comparativa práctica, además de una teórica con una base de datos real donde se podrán observar las distintas eficacias en la predicción de los distintos modelos propuestos.

La metodología se llevó a cabo con el software libre de R y el paquete "rcdk" para la generación de los descriptores, le siguió un pre-procesamiento de los datos y una posterior generación de los algoritmos con su debida comparación.

El algoritmo que se diferenció del resto por sus características fue el *Árbol de decisión* con una precisión del 0.88 y un índice kappa de 0.72 para este tipo de datos. Gracias a que es capaz de operar con bajos volúmenes de datos, pocos niveles y sobre todo por su capacidad de excluir características sin importancia. Se podría concluir que para bases de datos con un gran número de descriptores numéricos y pocos valores el algoritmo idóneo sería el *Árbol de decisión*.

Palabras clave

Máquinas de aprendizaje supervisado y no supervisado, máquina de vectores de soporte (*Support Vector Machine*, *SVM*), redes neuronales artificiales (*Artificial Neuronal Network, ANN*), propiedades ADME-Tox, diseño in silico, características moleculares, base de datos, programación, análisis estadístico, modelado de medidas experimentales, toxicidad.

Abstract

There is a growing predilection for the application of in silico techniques in the development and discovery of new drugs opposite of costly and laborious laboratory techniques. These are machine learning techniques.

The Master Final Project (TFM) consisted in the analysis of the best current techniques of machine learning for the prediction of ADME-Tox property, toxicity. Once a theoretical comparison has been made, a practice will be carried out with a real database where it will be possible to observe the different efficiencies in the prediction of the different proposed models.

The methodology was carried out with the free software of R and the "rcdk" package for the generation of the descriptors, followed by a pre-processing of the data and a subsequent generation of the algorithms with their proper comparison.

The algorithm that was differentiated from the rest by its characteristics was the *Decision Tree* with an accuracy of 0.88 and a kappa index of 0.72 for this type of data. Thanks to the fact that it is capable of operating with low data volumes, few levels and above all because of its ability to exclude unimportant features. It could be concluded that for databases with a large number of numerical descriptors and few values, the ideal algorithm would be the *Decision Tree*.

1. Introducción

Actualmente es sabido que la alta potencia de un fármaco no determina por completo su eficacia. Existen una serie de propiedades que también deben tomarse en cuenta, destacan la absorción, distribución, metabolismo, excreción y toxicidad [8]. Estas son las propiedades ADME-Tox, fundamentales en el descubrimiento de nuevos fármacos eficaces y seguros. Al ser estas propiedades las principales causas de fracaso en el desarrollo de nuevos fármacos, es comprensible que surjan algunas técnicas como es el caso de machine learning, que utiliza algunas variables predictoras como características moleculares para la obtención de modelos que determinen algunas de estas propiedades ADME-Tox.

Estos modelos in silico están en auge por el desorbitado gasto que conllevaba el descubrimiento de un nuevo fármaco usando los métodos tradicionales de prueba y error [2]. Y sobradamente, han demostrado ser un enfoque eficaz para aumentar la eficiencia en los procesos de descubrimiento y desarrollo de fármacos.

1.1. Justificación del Trabajo

El motivo por el que se realiza este TFM es porque las técnicas de machine learning, como *Support Vector Machine (SVM)* o *Artificial Neuronal Network (ANN)* se han estado empleando con un alto éxito en estudios farmacocinéticos para el desarrollo de nuevos fármacos candidatos a producirse, gracias a que son unas técnicas muy económicas y populares, además de relativamente rápidas ahorrándose enormes gastos de laboratorio [6].

Se encuentra una gran bibliografía de artículos sobre la predicción de las distintas propiedades ADME-Tox donde han escogido *SVM* o *ANN* principalmente [4], pero no hacen referencia a las restantes técnicas de machine learning igualmente aplicables y menos difundidas en este tipo análisis pero también muy conocidas en otros tipos de análisis que merecen su debida atención. Estás son; *Naive Bayes, Random Forest, Decision Tree, K-Nearest Neighbors*, más las anteriormente mencionadas *Support Vector Machine* y *Neuronal Artificial Network* [9].

Cuando entrenas un modelo predictivo con una base de datos que cuenta con al menos una serie de variables mínimas como el nombre, fórmula

del compuesto y el valor de la propiedad, obtienes una eficacia. Existe una amplia bibliografía sobre la predicción de estas propiedades ADME-Tox usando una técnica de las anteriormente mencionadas (*SVM* o *ANN*), pero no se encuentran informes experimentales informáticos donde usen otras técnicas y comparen sus eficacias obtenidas en la predicción para seleccionar una u otra técnica.

Además, existe una gran diferencia entre saber aplicar estas técnicas y conocerlas, siendo un problema cuando llega la hora de justificar porque con unas técnicas u otras se ha alcanzado una mayor o menor eficacia en la predicción estando incluso omitido este punto en multitud de artículos. La predicción de la toxicidad para un fármaco es extremadamente difícil y existe una necesidad urgente de desarrollar modelos nuevos y mejorados, por lo que sería interesante la obtención de un modelo todo lo fiable que pudiese serlo para la base de datos escogida y conocer cuáles son las variables mínimas para conseguir el mejor modelo para la toxicidad e intentar ofrecer una explicación lógica.

Con este TFM se intentará encontrar cuál es la técnica que adquiere una mayor eficacia con nuestra base datos, además de mostrar el resto de técnicas e intentar explicar a nivel interno (código y matemáticas) el por qué con unas técnicas se ha alcanzado una mayor o una peor predicción. Se ha elegido este tema debido a su alto contenido estadístico y aplicación práctica a nivel farma. Más la posibilidad de lograr una mejoría en el conocimiento del propio autor en este tipo de técnicas tan solicitadas tanto a nivel estructural como experimental de estas técnicas. Al ser el primer contacto del autor con estas propiedades ADME-Tox se irá describiendo todos los pasos que hagan progresar en el desarrollo del modelo para este fin.

1.2. Objetivos detallados

Las técnicas de machine learning como *SVM* y *ANN* se han estado empleando con un alto éxito en estudios farmacocinéticos para el desarrollo de nuevos fármacos candidatos a partir del empleo de modelos de machine learning. En el caso que trataremos más adelante, para la predicción de propiedades ADME-Tox a partir de relacionar características frente a las propiedades. Este informe se centrará en una de ellas, la toxicidad, responsable de que el noventa por ciento de los medicamentos sean retirados del mercado.

Se comenzará obteniendo una base de datos a partir de webs libres con un número de entradas (moléculas) considerable, ya que necesitamos alcanzar una alta eficacia en la predicción de los modelos. Para ello se debería obtener incluso varias bases de datos si se pudiese, y suponiendo que las condiciones experimentales son distintas se deberá realizar una limpieza de los datos según la propiedad que nos interese, la toxicidad. Utilizar machine learning no supervisado y herramientas de visualización de datos ayudaría a entender la estructura de los datos y decidir cómo habría que curarlos, pero la técnica de machine learning no supervisado no se realizará aquí.

Una vez la base de datos se cure, será el momento de aplicar machine learning supervisado. El objetivo final será encontrar un modelo con una alta eficacia en la predicción de la toxicidad para que con las moléculas que se entregarán posteriormente conocer si presentan toxicidad, y sobre todo, determinar con que técnica de machine learning (*Artificial Neuronal Network*, *Support Vector Machine*, *Naive Bayes*, *Random Forest*, *Decision Tree*, *K-Nearest Neighbors*…) es con la que se obtuvo la mayor eficacia en la predicción.

A continuación se exponen los objetivos que se quieren alcanzar con el Trabajo de Final de Máster.

Objetivos generales

Los objetivos generales del TFM son:

1. Hallar la técnica con la que se obtendrá la mejor predicción para una base de datos sencilla para poder hacerlo lo más reproducible posible, además de comentar todos los valores.
2. Explicar a nivel estadístico, cuando es mejor usar una técnica u otra para nuestros datos.

Objetivos específicos

Debido a que estos objetivos son muy generales se han desgranado en otros más específicos.

1. Técnicas de machine learning empleadas:
   i. Identificar las diferentes técnicas de machine learning que existen actualmente para el tratamiento de objetos.
   ii. Realizar una búsqueda extensa para hallar la base de datos correcta, generar las variables (descriptores) a partir de las moléculas y realizar la técnica de curación de los datos.
   iii. Realizar una comparativa entre las diferentes técnicas identificando con cual se obtiene una mayor eficacia.
   iv. Encontrar el modelo óptimo para este tipo de datos.

2. Estructura interna de los modelos:

   i. Desgranar los modelos hasta su comprensión.

   ii. Determinar cuándo sería necesario usar de forma óptima un modelo u otro para estos datos u otros.

   iii. Indicar cuales son las ventajas y desventajas de los modelos creados a partir de tales técnicas.

1.3. Método a seguir y planificación del trabajo

Se pueden realizar varias estrategias para llevar a cabo el trabajo, pero lo primero que se tiene que hacer es investigar cuales son los algoritmos que existen. Esta fase no requiere que sea muy detallada, sólo es una primera toma de contacto de los algoritmos.

Ya en un primer análisis se han detectado 4 candidatos, además de los algoritmos *Support Vector Machine* y *Artificial Neuronal Networks*.

A partir de aquí, se puede plantear el trabajo de varias formas:

- Primero, desarrollar la parte teórica del proyecto (búsqueda de revisiones bibliográficas, comparativas, etc), después la parte práctica (funciones disponibles en R, obtención base de datos), analizar los resultados obtenidos para identificar cual es el mejor algoritmo.

- O ir haciendo ambas partes del proyecto (teórica y práctica) a la vez.

- Una opción mixta entre las anteriores; se desglosarían las tareas por algoritmo (teoría, ventajas/ desventajas, funciones disponibles en R), una vez hecha la investigación, centrarse en la parte práctica del proyecto y por último analizar los resultados para llegar a la conclusión del estudio.

La opción seleccionada es la última, ya que se crean subtareas, lo que ayuda a centrar al investigador, ajustando los tiempos y proporcionando un soporte a la parte práctica, que va a ser la parte más complicada del proyecto, puesto que se tiene que hallar una base de datos, generar los descriptores, curarla y ajustarla para poder realizar los modelos.

De esta forma, si por alguna razón fuese imposible analizar todos los algoritmos mencionados, se podría decidir cuántos entran en el estudio, sin alargar la parte teórica del proyecto y ofreciendo el tiempo suficiente para realizar la parte práctica. También da la posibilidad de priorizar los algoritmos a analizar, investigando primero los más completos y, tal vez complejos, y dejando para los últimos aquellos que no van a ser los óptimos debido a las restricciones o inconvenientes que poseen.

1.4. Análisis de riesgos

Como en todos los proyectos hay factores que afectan negativamente en la planificación y en la ejecución del trabajo, lo que puede ocasionar que no se lleve a cabo el proyecto con la eficacia que debería.

A continuación se enumeran algunos de los que se pueden encontrar en el estudio planteado:

1. El proyecto se realiza en un tiempo muy ajustado.
2. El TFM englobe muchos modelos, dificultando llegar al objetivo marcado.
3. Haber realizado mal el plan de trabajo; concretamente en la priorización y el tiempo que se debe dedicar a cada tarea.
4. Utilizar una base de datos en que las variables no tengan suficiente relación.
5. Las variables no engloben el tipo de relaciones que se desean estudiar.
6. Que la base de datos utilizada tenga restricciones de uso.
7. Falta de conocimiento en la elaboración de los modelos.
8. No poder explicar correctamente el entramado de cada modelo porque el autor no es especialista en ello.
9. No haber generado bien las variables descriptoras.

2. Construcción del modelo

El procedimiento general para construir un modelo predictivo contiene aproximadamente cuatro pasos: recopilación de datos, descripción de datos, construcción de modelos y evaluación de modelos. Cada paso tiene sus propios requisitos para garantizar la fiabilidad y precisión de los modelos.

2.1. Búsqueda de la base de datos

La primera tarea se corresponde con la búsqueda de la base de datos para elaborar los distintos modelos. Después de una búsqueda concienzuda de artículos en buscadores como los de *web of science*, *scopus* y *google scholar* sobre las distintas bases de datos y directamente sobre el buscador mediante la clave “toxicity dataset ADME-Tox”. El autor pudo encontrar el siguiente artículo [10] que coleccionaba los servidores webs para los dataset de toxicidad ordenados según la calidad de los datos experimentales por el autor del artículo. Se puede decir que existen multitud de fuentes de bases de datos, muchas de ellas conectadas entre sí y no exentas de problemas para el usuario bioinformático que quiera hacerse con una base de datos para realizar modelos de clasificación. Si por ejemplo, hablamos de TOXNET, que está compuesta a su vez por dos bases de datos muy completas: TOXLINE y ChemIDplus, son útiles si se busca una única molécula y se quiere observar su estructura. También se encuentran fuentes de artículos de esa molécula, siempre hablando de la toxicidad por supuesto [Figura 1].

Data sources for prediction of chemical toxicity.

Database name	Type[a]	URL
TOXNET	CTA	https://toxnet.nlm.nih.gov/
ToxBank Data Warehouse	CTA	http://www.toxbank.net/data-warehouse
admetSAR	CTA	http://lmmd.ecust.edu.cn/admetsar1/
Pharmaco Kinetics Knowledge Base (PKKB)	CTA	http://cadd.zju.edu.cn/pkkb/
ToxCast	CTA	https://www.epa.gov/chemical-research/toxicity-forecasting
Tox21	CTA	https://tripod.nih.gov/tox21
CTD (Comparative Toxicogenomics Database)	CTA	http://ctdbase.org/
ECOTOX	CTA	https://cfpub.epa.gov/ecotox/
SuperToxic	CTA	http://bioinformatics.charite.de/supertoxic/
DSSTox	CTA	https://www.epa.gov/chemical-research/distributed-structure-searchable-toxicity-dsstox-database
ACToR	CTA	https://actor.epa.gov/actor/home.xhtml
T3DB	CTA	http://www.t3db.ca
eChemPortal	CTA	https://www.echemportal.org/echemportal/index.action
PubChem	CPI	http://pubchem.ncbi.nlm.nih.gov/
ChEMBLdb	CPI	https://www.ebi.ac.uk/chembldb/
BindingDB	CPI	http://www.bindingdb.org/bind/index.jsp

Figura 1. Listado de webs con acceso a bases de datos de toxicidad.

El inconveniente es que para este TFM se buscaba una base de datos que contuviese multitud de estas moléculas, ya que es un proceso laborioso la construcción de estas bases de datos y no se cuenta con el suficiente tiempo y recursos como para lograr su elaboración por el propio autor. A pesar de ser bases públicas muchas de ellas requieren de permisos por parte del autor de la web antes de poder acceder a ellas. Esto puede extenderse varios días y seguido de varios mensajes antes de tener acceso a una base de datos que al final puede que no sea la deseada, resumiéndose en un gasto de tiempo y recursos.

Otro problema destacable de algunas bases de datos encontradas en el artículo seria la ausencia de un documento “Readme” que indique una descripción de las variables del archivo descargable que proporciona la página, siendo necesario enviar un e-mail que puede encontrar o no respuesta en un periodo de tiempo desconocido porque la actualización de muchas de estas bases de datos no es diaria. Este problema se pudo encontrar en la base de datos DSSTOX.

Por último, otro problema común sería encontrar archivos comprimidos descargables que requieren de una contraseña para su descompresión. Incluso solicitando a los propios autores la contraseña esto puede demorar un largo tiempo. Estos son algunos de los problemas reales que el autor de este TFM se encontró en la etapa de búsqueda de una base de datos. Ya se remarcaba en el plan de trabajo la difícil tarea que resultaría la búsqueda de una base de datos que contase con el nombre de la molécula o código en su defecto, formula en SMILES y propiedad de estudio.

2.2. Descripción de los datos

Finalmente el autor se hizo con una base de datos procedente de la web admetSAR [11]. Esta base de datos ya ha sido usada para realizar modelos de clasificación con anterioridad [3], por lo que podemos intuir que exista cierta curación en los datos, aun así no exime la necesidad de realizar dichas comprobaciones y realización de la curación. La base de datos elegida hace referencia a una serie de 554 moléculas donde se estudia la toxicidad. El organismo en el cual se ha realizado el estudio es el pez *Pimephales Promela,s* y la variable predictiva sería LC50, concentración letal del 50%. De los 554 compuestos, se incluían 336 compuestos de alta toxicidad y 188 compuestos de bajo toxicidad. Si un compuesto presentaba un valor LC50 mayor de 0.5 mmol / L se asignaba como compuesto de alta toxicidad, mientras que a los menores se les asigno como compuesto de baja toxicidad. Es cierto que la base de datos no es de un gran tamaño, no obstante, las distintas dificultades mencionadas en este tipo de webs han llevado a escogerla.

La base de datos presenta esta estructura:

```
library(readxl)
myfile1 <- "T_FHMT_I.xls"
muestra <- read_xls(myfile1)
head(muestra)
```

```
## # A tibble: 6 x 4
##   CAS        SMILES                                        Labels `End p
oint`
##   <chr>      <chr>                                          <dbl> <chr>
## 1 61096-84-2 C1=CC(C=O)=CC(OC)=C1OCCCCCC                        1 highAT
## 2 98434-34-5 C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O            1 highAT
## 3 39145-47-6 C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2          1 highAT
## 4 71862-02-7 CC1=C(NC=O)C=CC=C1Cl                               1 highAT
## 5 3126-90-7  CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1                  1 highAT
## 6 3923-52-2  C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C                  1 highAT
```

Como se puede ver, no aparece el nombre de la molécula sino su código CAS (Código asignado por la Sociedad Americana de Química a cada compuesto químico), seguido de su fórmula en SMILES, los dos tipos de niveles (toxicidad y no toxicidad, 1 y 0 respectivamente) y el nivel expresado en formato character, alta y baja toxicidad.

Este pez ha adquirido una gran importancia para los investigadores que estudian la alteración endocrina, siendo clave en estudios de toxicología acuática como los ratones y las ratas lo son para los ensayos de drogas.

La extrapolación no es lo ideal, puesto que los peces se exponen mucho más al agua contaminada que las personas, viven en el agua, mientras nosotros solo bebemos o nadamos en ella. Pero por ética, no se van a realizar dichos experimentos en personas.

Afortunadamente, los sistemas endocrinos de peces y personas son fundamentalmente similares, dijo Dalma Martinovic-Weigelt, investigadora de la Universidad de St. Thomas, en St. Paul [12].

“Normalmente, la hembra pone los huevos en la parte inferior de una repisa natural del río, luego el macho hace todo el trabajo de cuida de los huevos. Frota los huevos contra la almohadilla de grasa y las protuberancias en forma de granos de su cabeza para mantenerlos limpios [Imagen 1]. Las protuberancias son características sexuales secundarias, como axilas con mucho bello en humanos.”

Imagen 1. *Pimephales Promelas* y sus protuberancias en la cabeza.

Pero los peces son vulnerables a los químicos. Si una mujer se expone a la testosterona ella comenzará a desarrollar una capa de grasa. Al igual que si se agregan químicos estrógenos al agua del pez macho, sus protuberancias se encogerán.

También aparecen una serie de cambios menos apreciables. El macho no puede cuidar tan bien de su nido. No es tan competitivo ni agresivo hacia otros machos. Sus testículos no se desarrollarán, y él producirá menos esperma, o puede ser más lento para huir de los depredadores. Martinovic y sus colaboradores descubrieron que los peces machos expuestos al efluente de la planta de tratamiento de aguas residuales actuaban como si hubieran estado expuestos a los estrógenos. Los machos lograron aparearse si otros machos no estaban cerca. Pero si tenían que competir con los machos control, sin embargo, sufrían un fracaso reproductiva casi total.

En este artículo la propiedad escogida es la LC50, que sería la medida estándar de la toxicidad del medio para la muerte de la mitad de la población de estudio al exponerse a un factor.

Una vez se seleccione la base de datos y se describa tanto contextualmente (LC50, especie…) como su estructura interna (fórmula, SMILES, CAS…) será el momento de pasar a la elaboración de los modelos.

Un requisito clave para el modelado predictivo de las propiedades y actividades moleculares son los descriptores moleculares, las caracterizaciones numéricas de la estructura molecular.

Se procedió a conseguir las librerías correspondientes que pudiesen generar los descriptores a partir de nuestras moléculas. Se probó con paquetes como Rcpi de BiocManager para crear los los distintos descriptores, siendo necesario actualizar R a su última versión, aun así el autor se encontró con serias dificultades para conseguirlo por su elevada complejidad, se buscaron otras alternativas como los paquetes ChemmineOB y rcdk. Con ChemmineOB del proyecto OpenBabel se llegó de nuevo al fracaso. Sin embargo, fue con el paquete rcdk que el autor se pudo hacer con los descriptores.

El paquete rcdk implementa una variedad de descriptores moleculares categorizados en; topológicos, constitucionales, geométricos, electrónicos e híbridos. Es posible evaluar todos los descriptores disponibles a la vez, o evaluar descriptores individualmente. Las distintas fórmulas en SMILES se caracterizaron con un total de 50 descriptores moleculares provenientes de las cinco categorías anteriormente mencionadas.

La construcción de cada modelo se encuentra más detallada en el anexo que se añade a este TFM. Mientras que la evaluación y resultados de los modelos se tratarán en los siguientes puntos.

Como actividad no prevista cabe mencionar la creación de una correlación de los descriptores y la variable de toxicidad para observar si algunas variables estaban más relacionadas que otras con la toxicidad, se destacó la variable XlogP, un nuevo método de adición de átomos para calcular los coeficientes de partición.

2.3. Curación de los datos

La limpieza y preparación de los datos constituyen una parte significativa de tiempo y esfuerzo invertidos en el análisis de datos. En muchos casos puede ser tentador omitir este proceso y dar el paso al modelado sin mirar detenidamente el conjunto de los datos. Teniendo en cuenta que ningún set de datos es perfecto (datos incorrectos, malinterpretación, falta de datos…) se deben de tomar las precauciones oportunas realizando una curación previa.

Se siguió el siguiente índice para la curación de los datos:

1.- Cargar los datos en R

2.- Disposición de los datos

3.- Tipos de variables

4.- Valores erróneos o faltantes

5.- Variables categóricas

6.- Aprendizaje no supervisado

Es muy probable que se necesite transformas columnas de variables antes del modelado. Ya que en el peor de los casos se podría construir un modelo que devolviese predicciones incorrectas y no se pudiese explicar el por qué. En cambio, si se aborda de forma temprana los problemas contenidos en los datos se podría ahorrar un trabajo innecesario. A continuación se demostrarán algunos problemas que podrían ocurrir [13], además de realizarse los ajustes pertinentes en nuestros datos.

Hay varios problemas que pueden surgir una vez se están preparando los datos en R:

- Al cargar los datos en R desde bases de datos, hojas de cálculo u otros formatos.

- Tipos de variables incorrectas.

- La gestión de valores incorrectos: valores NaN (no numérico) o que están fuera del rango.

- Tratar valores perdidos (NA).

- Gestionar nuevos valores categóricos.

- Valores categóricos con demasiados niveles.

Se intentará tratar todos estos puntos. Algunos se abordarán de manera más específica mientras que en otros se hará de forma más general.

2.3.1. Cargar los datos en R

Hay una gran variedad de funciones y paquetes para cargar los datos en R. En este análisis se usará read_xls() teniendo en cuenta que estamos importando una hoja de cálculo.

```
myfile1 <- "T_FHMT_I.xls"
muestra <- read_xls(myfile1)
head(muestra)
```

```
# A tibble: 6 x 4
  CAS        SMILES                                          Labels `End poi
nt`
  <chr>      <chr>                                            <dbl> <chr>
1 61096-84-2 C1=CC(C=O)=CC(OC)=C1OCCCCCC                          1 highAT
2 98434-34-5 C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O              1 highAT
3 39145-47-6 C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2            1 highAT
4 71862-02-7 CC1=C(NC=O)C=CC=C1Cl                                 1 highAT
5 3126-90-7  CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1                    1 highAT
6 3923-52-2  C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C                    1 highAT
```

2.3.2. Disposición de los datos

La forma más eficiente de almacenar los datos no tiene porqué ser la mejor forma para posteriormente analizarlos. Por ejemplo, se podría haber obtenido una base de datos donde una sola molécula ocupase varias filas por comodidad para el encargado del registro de datos. Por lo general, para el análisis de datos es preferible exponer los datos en un formato más amplio, con multitud de variables y una única fila para una única identidad. El paquete reshape2 sería un paquete adecuado para llevar a cabo esta función o realizar también el proceso inverso. En nuestro caso cada molécula química identificada por un número CAS se encuentra en una única fila.

2.3.3. Tipos de variables

Una vez se han cargado los datos en R es necesario realizar una exploración para comprobar que las variables son del tipo esperado, se encuentren en una distribución o falten valores. Una forma de comprobarlo sería mediante la función *summary().*

Variables categóricas también se pueden encontrar como numéricas. Esto puede ser un problema para algunos algoritmos de aprendizaje automático, ya que tratarían este tipo de variables como si fuesen valores numéricos continuos. La solución en R sería convertir las columnas a factor. O por el contrario, hallamos factores donde se esperaría encontrar variables numéricas. Lo más común sería convertir la columna mediante la función *as.numeric().*

Después de extraer los descriptores aplicamos la función *summary()*, para obtener un resumen de ellos:

Se debe tener en cuenta que la mayoría de salidas de código R están recortadas para no hacer tediosa la lectura.

```
SMILES<-muestra$SMILES
mols <- parse.SMILES(SMILES)

a<-unlist(lapply(mols, get.SMILES))

dc <- get.desc.categories()
```

```
descNames <- unique(unlist(sapply(get.desc.categories(), get.desc.name
s)))

allDescsfinal <- eval.desc(mols, descNames)
summary(allDescsfinal)
```

```
 Wlambda1.unity Wlambda2.unity Wlambda3.unity Wnu1.unity
 Mode:logical   Mode:logical   Mode:logical   Mode:logical
 NA's:554       NA's:554       NA's:554       NA's:554
```

```
WD.unity          BCUTw.1l        BCUTw.1h         BCUTc.1l
 Mode:logical   Min.   :11.69   Min.   : 12.10   Min.   :-0.4378
 NA's:554       1st Qu.:11.85   1st Qu.: 15.99   1st Qu.:-0.3639
                Median :11.89   Median : 16.00   Median :-0.3274
                Mean   :11.92   Mean   : 23.10   Mean   :-0.3190
                3rd Qu.:11.99   3rd Qu.: 31.97   3rd Qu.:-0.2973
                Max.   :12.00   Max.   :126.91   Max.   :-0.1124
```

Aquí podemos observar el rango de nuestros valores, y con la función *str()* el tipo de variable.

Como rcdk es un paquete preparado por especialistas y apoyado por su gran comunidad, no entraremos en si las variables obtenidas a partir de las fórmulas en SMILES son del tipo adecuado o no, ya que años de soporte a este paquete y reconocimiento por la comunidad investigadora lo avalan.

```
str(allDescsfinal)
```

```
'data.frame':   554 obs. of  286 variables:
 $ Wlambda1.unity  : logi  NA NA NA NA NA NA ...
 $ Wlambda2.unity  : logi  NA NA NA NA NA NA ...
 $ BCUTw.1l        : num  11.9 12 11.9 12 11.9 ...
 $ BCUTw.1h        : num  16 78.9 35 35 16 ...
 $ BCUTc.1l        : num  -0.347 -0.381 -0.308 -0.284 -0.313 ...
 $ BCUTc.1h        : num  0.15 0.17 0.208 0.191 0.294 ...
```

2.3.4. Valores erróneos o faltantes

Pueden faltar valores en la base de datos (NA) o aparecer valores problemáticos (NaN), niveles de categorías inválidos, valores inverosímiles (fuera de rango). La identificación de estos valores erróneos a menudo requiere de cierto conocimiento sobre estas variables.

En cuanto a los NA, si el número de valores faltantes es pequeño, sería seguro eliminar esas filas. Con la función *complete.cases()* se puede saber si la matriz presenta todos los datos (True) o hay valores NA (False). Sin embargo,

la ausencia de valores en una larga fila puede hacer que perdamos perfectamente la mitad de la totalidad de los datos y acabemos generando por ello un mal modelo. A veces compensa dejar pasar esos valores faltantes para mantener una gran matriz de datos. También es posible usar los Nas en variables categóricas como una categoría adicional.

En el caso de variables faltantes, en variables numéricas, es posible reemplazar un valor por un valor esperado o promedio de los datos no perdidos.

En nuestro caso, como se vio con las anteriores funciones hay multitud de NAs, pero viendo que todos se encuentran en la misma columna, en lugar de eliminar las filas (moléculas) que daría lugar a la eliminación de todas ellas, procedemos a eliminar las columnas. Esto no perjudicará al resultado. Ya que el paquete rcdk está preparado para introducir variables en función de la distribución 3D de la molécula. Y puesto que no se ha dado tal característica por nuestra parte, ha aparecido esa gran cantidad de NAs. Las columnas con Nas serán eliminadas mediante el siguiente código:

```
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
any(is.na(x)) )]
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
length(unique(x)) == 1 )]

summary(allDescsfinal)
```

```
    BCUTw.1l        BCUTw.1h        BCUTc.1l          BCUTc.1h      
 Min.   :11.69   Min.   : 12.10   Min.   :-0.4378   Min.   :0.002364  
 1st Qu.:11.85   1st Qu.: 15.99   1st Qu.:-0.3639   1st Qu.:0.098466  
 Median :11.89   Median : 16.00   Median :-0.3274   Median :0.156067  
 Mean   :11.92   Mean   : 23.10   Mean   :-0.3190   Mean   :0.172399  
 3rd Qu.:11.99   3rd Qu.: 31.97   3rd Qu.:-0.2973   3rd Qu.:0.232474  
 Max.   :12.00   Max.   :126.91   Max.   :-0.1124   Max.   :0.468828  
    BCUTp.1l         BCUTp.1h          XLogP               MW        
 Min.   : 2.595   Min.   : 3.112   Min.   :-2.0640   Min.   : 32.03  
 1st Qu.: 4.001   1st Qu.: 7.396   1st Qu.: 0.8055   1st Qu.:116.09  
 Median : 4.692   Median : 8.353   Median : 1.5600   Median :149.12  
 Mean   : 4.810   Mean   : 8.413   Mean   : 1.7330   Mean   :159.92  
 3rd Qu.: 5.412   3rd Qu.: 9.285   3rd Qu.: 2.5747   3rd Qu.:190.87  
 Max.   :12.835   Max.   :18.823   Max.   : 7.2440   Max.   :483.59  
```

2.3.5. Variables categóricas

Variables categóricas con demasiados niveles o niveles anormales.

Aunque una variable categórica pueda tomar muchos niveles porque la matriz es lo suficientemente grande, la mayoría de niveles serán anormales, y por lo tanto difíciles de predecir. Este problema se mantiene incluso si los niveles no son demasiados pero si extraños.

```
CAS<-muestra$CAS
Labels<-muestra$Labels
datafinal<-cbind(CAS,allDescsfinal,Labels)
head(datafinal$Labels,30)

 [1] 1 1 1 1 1 1 1 1 0 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 0 1 0 1 0
```

En este caso se ha elegido un total de dos variables categóricas (toxicidad y no toxicidad), siendo pocos y con niveles bastante comunes.

Se ha analizado una serie de puntos conflictivos que surgen al preparar los datos para su análisis. Se ha mostrado como detectarlos y abordarlos, tanto de forma automatizada cuando era posible o caso por caso cuando no.

2.4. Aprendizaje no supervisado

Es común que se realice la tarea de aprendizaje no supervisado, cuya tarea principal es la de encontrar estructuras a priori desconocidas entre los datos. Esto es debido a que se desconoce la estructura intrínseca de los datos por la ausencia de un atributo que supervise la formación de dichas estructuras. En resumen, función es básicamente exploratoria.

El clustering es una de las técnicas más populares aplicadas en el aprendizaje no supervisado, pero en este TFM no se tratará porque es laborioso y se ha decidido repartir el esfuerzo en otras tareas.

2.5. Construcción del modelo

2.5.1. Ejemplo realizado: Knn

Para generar con éxito el modelo se siguó el siguiente índice:

- Importar datos
- Generar descriptores
- Preparación de los datos
- Generar modelo

Importar datos

Se asigna al fichero que contiene las variables un nombre para trabajar más fácilmente:

```
myfile1 <- "T_FHMT_I.xls"
```

Ahora se importan los datos en formato data.frame para poder trabajar con ellos, y se visualizará a las primeras filas del dataframe:

```
muestra <- read_xls(myfile1)
head(muestra)

# A tibble: 6 x 4
  CAS        SMILES                                        Labels `End poi
nt`
  <chr>      <chr>                                          <dbl> <chr>
1 61096-84-2 C1=CC(C=O)=CC(OC)=C1OCCCCCC                        1 highAT
2 98434-34-5 C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O            1 highAT
3 39145-47-6 C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2          1 highAT
4 71862-02-7 CC1=C(NC=O)C=CC=C1Cl                               1 highAT
5 3126-90-7  CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1                  1 highAT
```

Como se puede observar, se obtuvo un total de 554 moléculas. No aparece el nombre de la molécula sino su código Cas, seguido de su fórmula en SMILES, los dos tipos de niveles (toxicidad y no toxicidad, 1 y 0 respectivamente) y el nivel expresado en formato character.

Generar descriptores

Finalmente, se generó una serie de características a partir de la molécula en SMILES usándose la función *parse.SMILES(),* ya que anteriormente se había mostrado simplemente en formato character, no interpretable o manipulable por los paquetes que se usaron de aquí en adelante:

```
SMILES<-muestra$SMILES
mols <- parse.SMILES(SMILES)
get.SMILES(mols[[1]])

[1] "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
```

Pero para generar todos los SMILES de una vez, mejor usar este código:

```
a<-unlist(lapply(mols, get.SMILES))
head(a)

                C1=CC(C=O)=CC(OC)=C1OCCCCCC
               "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
    C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O
  "C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O"
  C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2
"C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2"
                      CC1=C(NC=O)C=CC=C1Cl
                     "CC1=C(NC=O)C=CC=C1Cl"
          CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1
        "CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1"
          C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C
        "C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C"
```

Hay que tener en cuenta que las moléculas resultantes no tendrán coordenadas 2D o 3D, por lo que no se tendrá ninguna característica de aromaticidad, tipificación de átomos o configuración isotópica.

Un requisito clave para el modelado predictivo de las propiedades y actividades moleculares son los descriptores moleculares, estas son las caracterizaciones numéricas de la estructura molecular. El paquete rcdk implementa una variedad de descriptores moleculares categorizados en; topológicos, constitucionales, geométricos, electrónicos e híbridos. Es posible evaluar todos los descriptores disponibles a la vez, o evaluar descriptores individualmente.

```
dc <- get.desc.categories()
dc
```

```
[1] "hybrid"         "constitutional" "topological"    "electronic"
[5] "geometrical"
```

Como un descriptor puede pertenecer a más de una categoría se deberá conseguir que no se repita entre ellas, y extraerse todos los descriptores para las 5 categorías para aplicarlos más adelante.

```
descNames <- unique(unlist(sapply(get.desc.categories(), get.desc.name
s)))
descNames
```

```
 [1] "org.openscience.cdk.qsar.descriptors.molecular.WHIMDescriptor"
 [2] "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor"
 [3] "org.openscience.cdk.qsar.descriptors.molecular.XLogPDescriptor"
 [4] "org.openscience.cdk.qsar.descriptors.molecular.WeightDescriptor"
```

```
allDescsfinal <- eval.desc(mols, descNames)
```

Se consiguió un total de 50 descriptores moleculares generados a partir del paquete mencionado.

Ahora que se tiene una matriz de descriptores, se eliminarán las columnas que contienen valores NAs, variables correlacionadas y constantes.

```
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
any(is.na(x)) )]
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
length(unique(x)) == 1 )]
```

Se crea la matriz que se utilizará para crear el algoritmo:

```
CAS<-muestra$CAS
Labels<-muestra$Labels
datafinal<-cbind(CAS,allDescsfinal,Labels)
```

Preparación de los datos

Se eliminará la primera columna que contiene una numeración inservible:

```
rownames(datafinal) <- c()
```

También la variable Cas ya que no es necesario para la creación del modelo:

```
datafinal<-datafinal[-1]
```

Se creará una matriz de correlación y se guardará para observar a simple vista si hay una gran correlación entre los descriptores y la variable clase de toxicidad.

```
matrizc<-cor(datafinal)
write.xlsx(matrizc, file="correlación.xlsx",row.names=TRUE)
```

Entre los descriptores de esta matriz de correlación destaca XlogP con un 0.55. Siendo uno de los principales descriptores para este tipo de modelos.

Se convierte la clase a predecir a factor, ya que será necesario más adelante para poder aplicar el paquete de nuestro modelo.

```
Label<-as.factor(datafinal$Labels)
datafinal_fac<-cbind(datafinal[-175],Label)#Añadir columna
```

Para algunos modelos es necesario normalizar los datos. Para ello, incluir la variable clase.

```
# Normalizamos
normalize <- function(x) {
return((x - min(x)) / (max(x) - min(x)))
}

datafinal_facn <- as.data.frame(lapply(datafinal_fac[1:174], normalize
))
```

Generar modelo

Al elaborar un modelo se comienza eligiendo una muestra aleatoria mediante la función *set.seed()* y realizando una partición de los datos para llegar a obtener los datos de entrenamiento y test [7]. Finalmente aplicamos la función *knn()* a nuestros datos de entrenamiento y elegimos una constante, K. Posteriormente se realiza la prueba con los datos test y se verán los resultados de la predicción con *confusionmatrix()*. Como consejo en cuanto a la obtención de la constante K, realizar la raíz cuadrada del número de objetos del data_train. En este caso se obtuvo 19.

```r
set.seed(1234)
n_train <- 2/3
n <- nrow(datafinal_facn)
train <- sample(n,floor(n*n_train))
data_nrm.train <- datafinal_facn[train,]
data_nrm.test  <- datafinal_facn[-train,]



datafinal_training<-Label[train]
datafinal_test<-Label[-train]

test_pred <- knn(train =data_nrm.train, test = data_nrm.test, cl = dat
afinal_training, k=19)
#k se obtiene de la raíz cuadrada del training data.
datafinal_test<-as.factor(datafinal_test)
confusionMatrix(test_pred,datafinal_test)
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  45  11
         1  18 111

               Accuracy : 0.8432
                 95% CI : (0.7827, 0.8924)
    No Information Rate : 0.6595
    P-Value [Acc > NIR] : 1.617e-08

                  Kappa : 0.6414

 Mcnemar's Test P-Value : 0.2652

            Sensitivity : 0.7143
            Specificity : 0.9098
         Pos Pred Value : 0.8036
         Neg Pred Value : 0.8605
             Prevalence : 0.3405
         Detection Rate : 0.2432
   Detection Prevalence : 0.3027
      Balanced Accuracy : 0.8121

       'Positive' Class : 0
```

Se ha obtenido un 0.84 de precisión en cuanto a la detección de la toxicidad en las moléculas con un índice kappa del 0.64 que nos indica como nuestro modelo se acerca a la realidad, siendo 1 el valor máximo.

El resto de Modelos lo encontramos en el apartado de Anexo.

2.5.2. Listado de los resultados parciales obtenidos hasta el momento

Los seis modelos tienen una precisión entorno al 0.85 garantizando el éxito de ellos y por consiguiente de la fiabilidad de la base de datos y la generación de descriptores. Por el momento solo se han obtenido los resultados, en esta primera parte del TFM no se ha comentado nada con respecto a ellos.

Los modelos fueron generados con R en su versión 3.5.3. Solo se comentó el código del algoritmo *Knn*, el resto se incluirá pero no habrá una descripción detallada ya que no es el objetivo.

3. Descripción de los algoritmos

3.1. Comparativa de los modelos realizados

En este apartado se encuentra el foco de este trabajo de fin de máster. Antes de comenzar con la evaluación de las predicciones sería necesario realizar una breve pausa para explicar los distintos índices destacables que nos ofrece esta matriz de confusión generada a partir de los modelos.

Para poder realizar la evaluación de los modelos de clasificación los datos se separaron en:

1. Entrenamiento (66%)

2. Test (33%)

El entrenamiento se realizó utilizando el set de entrenamiento para luego evaluar el modelo generado con los datos test. Esto nos permitirá comprobar cómo se comporta el modelo cuando se le introducen nuevos datos. La matriz que se obtuvo con la función *confusionmatrix()* presenta la siguiente forma [Figura 2]:

		Predicción	
		Positivos	**Negativos**
Observación	**Positivos**	Verdaderos Positivos (VP)	Falsos Negativos (FN)
	Negativos	Falsos Positivos (FP)	Verdaderos Negativos (VN)

- **VP** es la cantidad de *positivos* que fueron *clasificados correctamente* como positivos por el modelo.
- **VN** es la cantidad de *negativos* que fueron *clasificados correctamente* como negativos por el modelo.
- **FN** es la cantidad de *positivos* que fueron *clasificados incorrectamente* como negativos.
- **FP** es la cantidad de *negativos* que fueron *clasificados incorrectamente* como positivos.

Figura 2. Estructura de una matriz de confusión.

Para el resto de índices [14]:

- Acurracy (exactitud): Para el porcentaje de datos de tipo clase clasificados correctamente.

- Sensibilidad (sensibility): Porcentaje de clases positivos que fueron clasificados como positivos.

- Especificidad (specificity): Porcentaje de clases negativos que fueron clasificados como negativos.

- Kappa: Es la diferencia entre la clasificación correcta lograda por el modelo sin azar y la clasificación correcta debida al azar. Un índice alto representaría un buen modelo de clasificación no basado en la aleatoriedad.

Realizada esta introducción de índices, llega el momento de ver los resultados [Tabla 1]:

Modelo de clasificación	**Exactitud**
Artificial neuronal network	0.83
Decision tree	0.88
Naive bayes	0.76
k-nearest neighbor	0.84
Ramdon forest	0.85
Support vector machines	0.84

Tabla 1. Lista de resultados obtenidos en los distintos modelos.

Vistos los resultados se podrían obtener tres grupos: grupo de mayor clasificación, que en su caso fue el algoritmo de *decision tree* con una precisión de 0.88. Grupo de mediana clasificación, en este caso se obtendrían varios algoritmos, *Artificial neuronal network (Ann)*, *k-nearest neighbor(Knn), Ramdon forest* y *Support vector machine*s(svm) con una media de 0.84. Por último tenemos un modelo con una clasificación inferior pero nada desdeñable, *Naive bayes*, con un valor de 0.76.

Estas precisiones se consideran lo suficientemente elevadas como para llevarlas a la práctica, si es cierto que son modelos interpretativos y dictan mucho de asegurar una precisión absoluta (100%), no obstante, sería beneficiosa su aplicación.

A posteriori, se pasará a revisar el resto de índices: la sensibilidad y especificidad suelen mantenerse por encima del valor 0.75, destacándose el algoritmo *desicion tree* como anteriormente se pudo apreciar. La exactitud del modelo está relacionada con estos dos índices, por lo que se podrían predecir estos altos valores. Finalmente, se hablará del índice kappa, clases que por azar se clasificaron correctamente. Lo ideal sería obtener índices lo más cercanos a uno dentro de lo posible [Tabla 2]. El algoritmo de *decision tree* cuenta con un índice del 0,72. El resto se sitúa alrededor del 0,65.

kappa	grado de acuerdo
< 0,00	sin acuerdo
>0,00 - 0,20	insignificante
0,21 - 0,40	discreto
>0,41 - 0,60	moderado
0,61 - 0,80	sustancial
0,81 - 1,00	casi perfecto

Tabla 2. Grado de acuerdo del índice kappa

Viendo la siguiente tabla se puede afirmar que el índice es sustancial [15], y por lo tanto, además de servir el modelo como una interpretación teórica para ver las tendencias de posibles relaciones entre variables, también se puede usar en la práctica para predecir.

3.2. Modelo óptimo para este tipo de datos

De los seis modelos que se realizaron, uno se destacó por encima del resto. Cada algoritmo presenta unas características que le hace diferenciarse del resto, encajando por consiguiente mejor con unos datos u otros. Se hablará de aquí en adelante de esas particularidades que contienen, que les ha hecho encajar mejor o peor con los datos presentados.

Si hablamos de *naive bayes* y su no tan alta precisión en la predicción con nuestros datos, cabe mencionar una explicación al respecto. A pesar de que se necesitan pocos ejemplos para el desarrollo con éxito de este algoritmo, presenta una serie de inconvenientes que le hace errar con respecto al resto. Uno de ellos es la realización de supuestos que realiza este modelo, supuestos defectuosos normalmente y con demasiada frecuencia. Esto se agrava si además se cuenta con que el algoritmo no es ideal para conjuntos de datos con muchas características numéricas como es el caso de la base de datos actual. En el caso opuesto, el algoritmo *artificial neuronal network* realiza unas suposiciones sobre las relaciones de los datos más acertadas *que naive bayes*, convirtiéndose en uno de sus puntos fuertes. Si además le añadimos una mayor capacidad de modelar patrones más complejos que casi cualquier algoritmo nos encontramos con un modelo que supera a *naive bayes* para estos datos, pero no ofrece mejores resultados debido a que presenta cierta tendencia a cometer sobreajuste. Este sobreajuste merece su debida explicación:

Al realizar un modelo lo que se genera sería una función a partir de los datos de entrenamiento de los que se dispone. Lo ideal sería encontrar una función similar al de la [Figura 3] donde cada nuevo dato introducido en el eje horizontal podría ser predicho por esta función dando la componente en el eje vertical de acuerdo a la función.

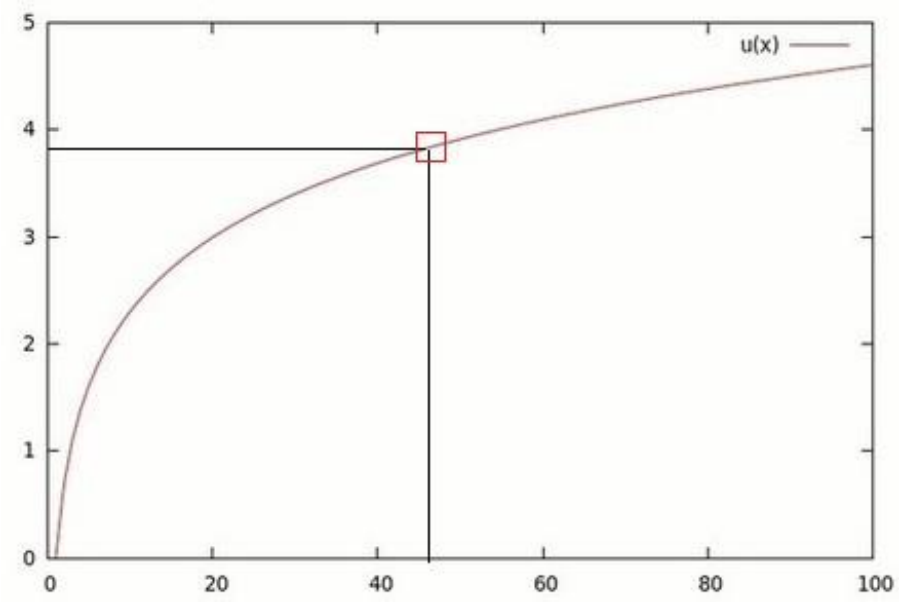

Figura 3. Ejemplo de función típica

Sin embargo una función sobre ajustada presentaría el siguiente aspecto [Figura 4], en ella se puede observar como solo los datos que presentasen las mismas características numéricas tendrían una buena predicción, ya que se situarían en el mismo vértice de la función, por el contrario, cualquier nuevo dato que varíe lo suficiente cometerá un error al predecirlo, en resumen, solo predice con acierto los mismos datos, para nuevos datos cometerá graves errores.

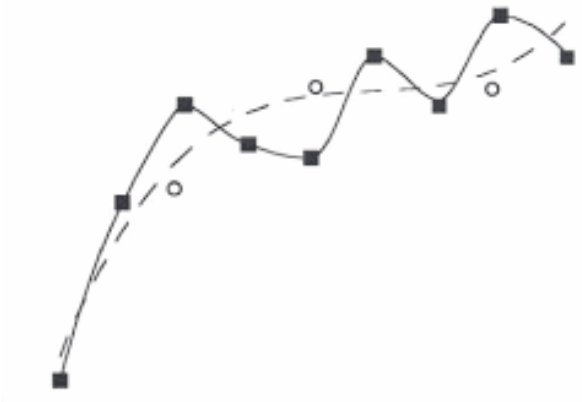

Figura 4. Ejemplo de función sobre ajustada.

Por lo que tampoco sería el más idóneo para este tipo de datos.

Un modelo que se opone a hacer suposiciones sobre la distribución de los datos es el algoritmo *knn*. Parece hallar una gran precisión aún respecto a esto, por lo que no se llegaría a considerar una característica negativa, sino

más bien al contrario. Como tal, no llega a producir un modelo propiamente dicho, lo que limita la capacidad de comprender como se relacionan las características con la clase. Otra dificultad que arrastra es la necesidad de selección de un K apropiado. Pese a estos rasgos, consiguen la misma predicción que el modelo de *artificial neuronal networks*. Al igual que *knn,* el modelo de *support vector machines* requiere probar diferentes kernels para la realización del modelo predictivo, pese a ello consigue una precisión tan vistosa como los dos modelos anteriormente citados.

El modelo de *ramdon forest* es capaz de elegir de entre todas las características presentes en la base de datos aquellas más importantes, pero conlleva un trabajo extra el ajustar el modelos a los datos, por lo que se sitúa justo por encima de los anteriores modelos en la predicción para estos datos, ya que requiere de un esfuerzo extra para obtener mejores resultados.

Por último, el algoritmo *decision tree* que se antepuso al resto de algoritmos presenta las mejores características que hacían destacar al resto de modelos para estos datos. De forma similar a *ramdon forest*, excluye características sin importancia de esta serie de datos, que se obtuvieron a partir de una librería, rcdk, que podría ser usada para la predicción de multitud de moléculas distintas con sus propias particularidades, por lo que seguramente habríamos generado características irrelevantes para nuestros datos que no harían más que introducir ruido ellos. Siendo indispensable está característica selectiva.

Como nuestra base de datos no es lo suficientemente grande, un algoritmo que pudiese operar con volúmenes pequeños de datos también sería una particularidad indispensable, en este caso el algoritmo *desicion tree* es especialista en ello.

Existen algoritmos como *decision tree* que se empequeñecen cuando necesitan clasificar los datos en un gran número de niveles, en el caso de los datos de toxicidad solo presentarían dos niveles, por lo que se beneficiaria de esta debilidad.

Estas tres características hacen que *decision tree* se eleve respecto al resto de algoritmos para este conjunto de datos.

3.3. Mejor modelo para los distintos tipos de datos

Todos los modelos estudiados fueron de *aprendizaje supervisado*, donde se predice una variable objetivo a partir de un conjunto de datos. Este a su vez se divide en algoritmos de *variables categóricas* y *variables continuas*. Como se trataba de predecir la toxicidad o la no toxicidad (clases) de las moléculas, se utilizaron los algoritmos de clasificación. Si por el contrario se hubiese deseado predecir valores continuos habría sido necesario realizar algún modelo de regresión. También existe el *aprendizaje no supervisado* que predice estructuras o patrones puesto que no existe ninguna variable objetivo a predecir. Un ejemplo de ellos serían los *análisis de componentes principales*

(PCA). He de mencionar que existen otros tipos de machine learning pero no se profundizará en ello [16].

- *Knn*: Puede ser usado tanto para problemas de clasificación como de regresión. Sin embargo, es más ampliamente utilizado en problemas de clasificación. Se debe tener en cuenta que las variables deben normalizarse, de lo contrario, las variables de rango superior pueden sesgar al resto. Además de una minuciosa etapa de pre-procesamiento de los datos para la eliminación de ruido. *Knn* funciona bien con un pequeño número de variables de entrada, pero tiene problemas cuando el número de entradas es muy grande.

- *SVM*: Como se dijo anteriormente, las máquinas de vectores de soporte se dirige tanto a la resolución de problemas de clasificación como de regresión, indistintamente. Al contrario que *knn,* funciona perfectamente con datos ruidosos. Útil para cuando los distintos niveles de clasificación se encuentran muy diferenciados entre ellos.

- *Naive bayes*: Empleado en modelos de clasificación. En concreto es el más común en clasificaciones de texto. Realmente efectivo con datos ruidosos y faltantes. Funciona bien tanto con bases de datos pequeñas como grandes, pero no es bueno con características numéricas. Tiene en cuenta que todas las variables son independientes entre ellas. Destaca con bases de datos enormes.

- *Ann*: Se puede adaptar a problemas de clasificación o predicción numérica. Al contrario que *naive bayes* hace suposiciones de relaciones entre variables, por lo que es muy indicado para tipos de datos relacionados.

- *Ramdon forest*: Es más eficiente que *decision tree* cuando los datos presentan mucha variabilidad ya que realiza promedios. Puede manejar bases de datos extremadamente grandes, porque selecciona características aleatorias, más grandes que cualquier otro algoritmo. No supone un impedimento los datos faltantes o ruidosos. Se emplea tanto para modelos de clasificación como para variables continuas.

- *Decision tree*: Al igual que *ramdon forest* se emplea tanto para modelos de clasificación como para variables continuas. Es capaz de eliminar variables poco relevantes, siendo muy útil cuando existe una base de datos con demasiadas características de dudosa relación con la variable objetivo. También se puede emplear con grandes bases de datos, pero en menor medida que su hermano. Se vuelve inútil cuando la variable a predecir contiene numerosos niveles.

3.4. Ventajas y desventajas de estos modelos

Se definirán los distintos modelos además de sus fortalezas y debilidades [7].

Algoritmo *knn*

El enfoque de clasificación de los vecinos más cercanos se ejemplifica mediante el algoritmo de vecinos más cercanos k (*Knn*).Este algoritmo se encargará de clasificar todo caso en un grupo determinado, esto se realizará mediante un conteo de k vecinos que se encontrarán más cerca de un grupo o de cualquier otro. Por lo tanto, se va a necesitar hallar la distancia de cada una de las entidades al resto, para luego ordenarlas de menor a mayor e identificar a que grupo pertenece la nueva entidad. Pertenecerá al grupo con la menor distancia [7]. Aunque este es quizás uno de los algoritmos de aprendizaje de máquina más simples, todavía se usa ampliamente. Las fortalezas y debilidades de este algoritmo son las siguientes [Tabla 3]:

Fortalezas	**Debilidades**
- Simple y eficaz	- No produce un modelo, lo que limita la capacidad de comprender cómo se relacionan las características con la clase
- No hace suposiciones sobre la distribución de datos subyacente	- Requiere selección de un k apropiado
- Fase de entrenamiento rápido	- Fase de clasificación lenta
	- Las características nominales y los datos faltantes requieren un procesamiento adicional

Tabla 3. Fortalezas y debilidades del algoritmo Knn.

Algoritmo Support Vector Machine (SVM)

Las máquinas de vectores de soporte (*Support Vector Machines, svm)* son un conjunto de algoritmos de aprendizaje supervisado, dirigido tanto a la resolución de problemas de clasificación como de regresión, indistintamente.

Los algoritmos de *svm* se basan en buscar el hiperplano [17] que tenga mayor margen posible y de forma homogénea entre las clases. Formalmente, un *svm* construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) para crear particiones bastante homogéneas a cada lado.

Algunas de las aplicaciones son:

- Clasificar genes diferencialmente expresados a partir de datos de microarrays.

- Clasificación de texto en distintas categorías temáticas.

- Detección de eventos críticos de escasa frecuencia, como terremotos.

Los kernels más populares son el lineal y el gausiano, aunque hay otros como el polinomial, string kernel, chi-square kernel, etc.

En cuanto a sus fortalezas y debilidades [Tabla 4]:

Fortalezas	**Debilidades**
- Se puede usar para problemas de clasificación o predicción numérica	- Encontrar el mejor modelo requiere probar diferentes kernels y parámetros del modelo (prueba y error)
- Funciona bastante bien con datos ruidosos y no es muy propenso al overfitting	- Lento de entrenar, sobre todo a medida que aumenta el número de características
- Puede ser más fácil de usar que las redes neuronales, en particular debido a la existencia de varios algoritmos SVM bien soportados	- Los resultados del modelo son difícil, si no imposible, de interpretar (caja negra)
- Gana popularidad debido a su alta precisión y ganancias de alto perfil en competiciones de minería de datos	

Tabla 4. Fortalezas y debilidades del algoritmo SVM.

Algoritmo *Naive bayes*

El algoritmo *Naive bayes* describe un método simple para aplicar el teorema de Bayes a los problemas de clasificación. Aunque no es el único método de aprendizaje automático que utiliza métodos bayesianos [1], es el más común. Esto es particularmente cierto para la clasificación de texto, donde se ha convertido en el estándar de facto. Las fortalezas y debilidades de este algoritmo son las siguientes [Tabla 5]:

Fortalezas	Debilidades
- Sencillo, rápido y muy efectivo.	- Se basa en un supuesto a menudo defectuoso de características igualmente importantes e independientes
- Lo hace bien con datos ruidosos y faltantes.	- No es ideal para conjuntos de datos con muchas características numéricas
- Requiere relativamente pocos ejemplos para la capacitación, pero también funciona bien con un gran número de ejemplos.	- Las probabilidades estimadas son menos confiables que las clases predichas
- Fácil de obtener la probabilidad estimada de una predicción.	

Tabla 5. Fortalezas y debilidades del algoritmo Naive Bayes.

Algoritmo Artificial neuronal network

Las *redes neuronales artificiales* (*Ann*) son sistemas informáticos vagamente inspirados en las redes neuronales biológicas que constituyen los cerebros de los animales. Dichos sistemas “aprenden” (es decir, mejoran progresivamente el desempeño en las tareas al considerar ejemplos), generalmente sin programación específica de la tarea. Por ejemplo, en el reconocimiento de imágenes, pueden aprender a identificar imágenes que contienen gatos analizando imágenes de ejemplo que se han etiquetado manualmente como “gato” o “no gato” y usar los resultados para identificar gatos en otras imágenes. Lo hacen sin ningún conocimiento a priori sobre los gatos, por ejemplo, que tienen pelaje, colas, bigotes y caras de gato. En su lugar, evolucionan su propio conjunto de características relevantes a partir del material de aprendizaje que procesan.

Una *Ann* principalmente se basa en una colección de unidades conectadas o nodos llamados neuronas artificiales (una versión simplificada de neuronas biológicas en un cerebro animal) [Figura 5]. Cada conexión (una versión simplificada de una sinapsis) entre neuronas artificiales puede transmitir una señal de una a otra. La neurona artificial que recibe la señal puede procesarla y luego señalar las neuronas artificiales conectadas a ella.

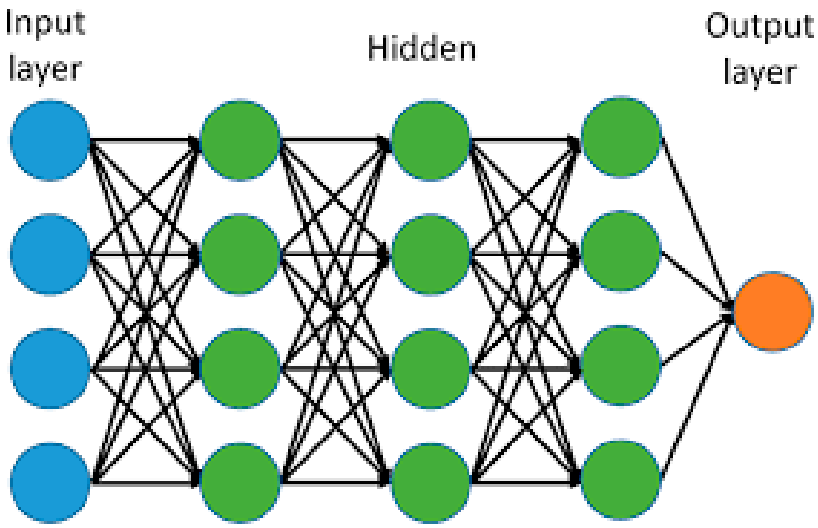

Figura 5. Capas ocultas (nodos) de una red neuronal

En implementaciones de *Ann* comunes, la señal en una conexión entre neuronas artificiales es un número real, y la salida de cada neurona artificial se calcula mediante una función no lineal de la suma de sus entradas. Las neuronas y conexiones artificiales suelen tener un peso que se ajusta a medida que avanza el aprendizaje. El peso aumenta o disminuye la intensidad de la señal en una conexión. Las neuronas artificiales pueden tener un umbral tal que solo si la señal agregada cruza ese umbral se envía la señal. Típicamente, las neuronas artificiales se organizan en capas. Diferentes capas pueden realizar diferentes tipos de transformaciones en sus entradas. Las señales viajan desde la primera (entrada) a la última capa (salida), posiblemente después de atravesar las capas varias veces [5].

Entrenar una red neuronal mediante el ajuste de los pesos de conexión es muy computacional. En consecuencia, aunque se habían estudiado durante décadas anteriores, las *Ann* rara vez se aplicaban a tareas de aprendizaje del mundo real hasta mediados de la década de 1980, cuando se descubrió un método eficiente de capacitación de una *Ann*. El algoritmo, que utiliza una estrategia de errores de propagación hacia atrás, ahora se conoce simplemente como propagación hacia atrás.

Aunque todavía es notoriamente lento en relación con muchos otros algoritmos de aprendizaje automático, el método de propagación hacia atrás llevó a un resurgimiento del interés en las *Ann*. Como resultado, las redes de avance de múltiples capas que utilizan el algoritmo de propagación hacia atrás ahora son comunes en el campo de la minería de datos. Tales modelos ofrecen las siguientes fortalezas y debilidades [Tabla 6]:

Fortalezas	Debilidades
- Se puede adaptar a problemas de clasificación o predicción numérica.	- Extremadamente intensivo en computación y lento en el entrenamiento, particularmente si la topología de la red es compleja.
- Capaz de modelar patrones más complejos que casi cualquier algoritmo.	- Muy propensos a sobreajustar los datos de entrenamiento.
- Hace algunas suposiciones sobre las relaciones subyacentes de los datos.	- Resultados en un modelo complejo de caja negra que es difícil, si no imposible, de interpretar.

Tabla 6. Fortalezas y debilidades del algoritmo Ann.

Las redes neuronales usan neuronas definidas de esta manera como bloques de construcción para construir modelos complejos de datos. Aunque existen numerosas variantes de redes neuronales, cada una puede definirse en términos de las siguientes características:

- Una función de activación, que transforma las señales de entrada combinadas de una neurona en una única señal de salida para ser transmitida en la red.

- Una topología de red (o arquitectura), que describe el número de neuronas en el modelo, así como el número de capas y la forma en que están conectadas.

- El algoritmo de entrenamiento que especifica cómo se configuran los pesos de conexión para inhibir o excitar las neuronas en proporción a la señal de entrada.

Algoritmo *Ramdon forest*

Ramdon forest construye una larga colección de árboles no correlacionados y luego los promedia para reducir la variación. Combina versatilidad y potencia en una sola máquina de aprendizaje. Como utiliza solo una pequeña porción aleatoria del conjunto completo de características, pueden manejar conjuntos de datos extremadamente grandes, cuando el problema de la dimensionalidad puede causar que otros modelos fallen. Al mismo tiempo, sus tasas de error para la mayoría de las tareas de aprendizaje están a la par con casi cualquier otro método. Sus fortalezas y debilidades se pueden resumir en [Tabla 7]:

Fortalezas	Debilidades
- Un modelo de uso múltiple que funciona bien en la mayoría de los problemas.	- A diferencia de un árbol de decisión, el modelo no es fácil de interpretar
- Puede manejar datos faltantes o ruidosos, así como características categóricas o continuas.	- Puede requerir algún trabajo para ajustar el modelo a los datos
- Selecciona solo las características más importantes.	
- Se puede usar en datos con una gran cantidad de funciones o ejemplos.	

Tabla 7. Fortalezas y debilidades del algoritmo Ramdon forest.

Algoritmo *Decision tree*

Los árboles de decisión son clasificadores muy potentes. Utilizan una estructura en árbol para modelar las relaciones entre las características y los posibles resultados [Figura 6]. Esta estructura obtuvo su nombre debido al hecho de que se representa como un árbol, comienza con un tronco ancho, que si se sigue hacia arriba, se vuelve más estrecho. De la misma manera, un clasificador de árbol de decisión utiliza una estructura de decisiones con bifurcaciones (nodos), que canalizan ejemplos para predecir una clase final.

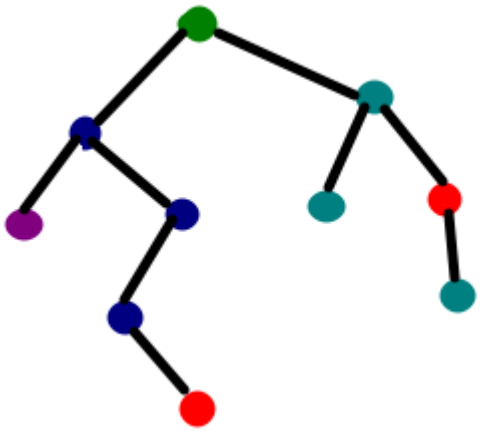

Figura 6. Esquema de un árbol de decisión con las distintas elecciones.

Los algoritmos de árbol de decisión generan una estructura resultante en un formato legible por el usuario. Esto proporciona una visión tremenda de cómo y por qué el modelo funciona o no funciona para una tarea particular. Al contrario que otros algoritmos de caja negra como pueden ser *svm* y *Ann*.

Existen numerosas implementaciones de árboles de decisión, pero una de las más conocidas es el algoritmo C5.0. Además, como se muestra en la

siguiente tabla [Tabla 8], las debilidades del algoritmo son relativamente menores y pueden evitarse en gran medida:

Fortalezas	**Debilidades**
- Clasificador de uso múltiple que funciona bien en la mayoría de los problemas	- Suelen errar cuando los datos presentan un gran número de niveles.
- Proceso de aprendizaje altamente automático que puede manejar tanto caracteres numéricos como nominales, así como valores perdidos.	- Es fácil cometer un sobreajuste.
- Excluyen características sin importancia.	- Se pueden tener problemas para modelar algunas relaciones debido a la confianza en divisiones eje-paralelo.
- Se puede usar tanto con pequeños como grandes volúmenes de datos.	- Ocurren pequeños cambios en los datos de entrenamiento que pueden dar lugar a graves cambios en las soluciones lógicas.
- Resultados interpretables sin necesitar gran experiencia en las matemáticas.	- Los árboles grandes pueden ser difíciles de interpretar y contrarios a la solución más intuitiva.
- Más eficiente que otros modelos más complejos.	

Tabla 8. Fortalezas y debilidades del algoritmo Decision tree.

4. Conclusión

Este proyecto se ha convertido en la primera toma de contacto por parte del autor con respecto a las propiedades ADME-tox. En él, se ha querido demostrar un ejemplo práctico de machine learning dirigido a investigadores con cierto conocimiento en bioinformática o similares. El objetivo era tanto demostrar como se podía realizar paso a paso este tipo de análisis como determinar cual era el mejor método para este tipo de datos utilizando la librería rcdk. Los resultados sugieren que el árbol de decisión fue el modelo que más se benefició de esta estructura de datos. Si repasamos cuales fueron las características que le hizo destacar: Excluía atributos sin importancia del conjunto de datos generados por el paquete rcdk, era capaz de manejar bases de datos de tamaño reducido y funcionaba realmente bién con pocos niveles de predicción. Estas tres características determinaron el alto índice de predicción. Cierto es que el resto de modelos también fueron capaces de obtener grandes resultados, pero por unas características determinadas, no acabaron de aproximarse al modelo elegido en cuanto a la predicción.

Los índices kappa también mostraron que la base de datos escogida para la predicción de la toxicidad fue realmente un éxito. Un índice alto demostro como de cercanos eran esos modelos en cuanto a la realidad y no dependientes del azar.

Además se destaca que los apendices que tratan las fortalezas y debilidades de los algoritmos estudiados pueden servir como punto de partida o guia para la elección de futuros modelos para el personal investigador. Mencionar a Lantz B, 2013, pues esté punto se debe exclusivamente a sus aportaciones en “Machine learning with R”.

Con respecto a si se han logrado los objetivos finales en este proyecto, decir que sí. Se ha identificado una base de datos, realizados los distintos modelos y comparados entre ellos. Todo se ha logrado con un rotundo éxito: de la base de datos se pudieron extraer unos descriptores que predecían bien la toxicidad, los modelos dieron diferencias entre ellos, pues eso era lo que se buscaba, en caso contrario, el que todos los algoritmos diesen la misma o peor, mala predicción, se debería revisar la planificación o buscar una nueva base de datos. Pero no fue el caso. Y finalmente se supo explicar el porqué de las respectivas predicciones por sus características.

En cuanto a la metodología realizada en este TFM también se le podría llamar de exitosa, pero con algunos matices. La base de este proyecto ha sido planificada satisfactoriamente, pero con un mayor número de horas se podría haber demostrado mediante ejemplos todas las fortalezas y debilidades mostradas en las tablas. Pero evidentemente se superaría con creces el número de horas dedicado a este proyecto. Por lo que se decidió no realizarlo.

En cuanto a la planificación, se ha realizado con asiduidad la entrega de las distintas partes de este proyecto como estaba redactado en el plan docente. Por lo que ha sido llevada con esmero y solventado los problemas que se iban detectando durante la realización de estas entregas.

Para futuras líneas de trabajo, sería recomendable realizar una guía donde se muestren para el tipo de propiedad ADME-Tox y base de datos, el modelo con mayor predicción y proximidad a la realidad. No solo bioinformáticos se verían beneficiados, sino el resto del personal investigador que no tiene la necesidad de aprender todas las técnicas de machine learning cuando solo requieren una en un momento de su investigación. Con ello se aceleraría la velocidad de la investigación en la comunidad científica, pues hay un excedente de datos de los que se puede sacar un gran provecho. Además, sería interesante mostrar ejemplos de esas fortalezas y debilidades para que no sea presentado únicamente de forma teórica como se mencionó anteriormente.

5. Bibliografía

1. Clark AM, Dole K, Coulon-Spektor A, McNutt A, Grass G, Freundlich JS, et al. Open Source Bayesian Models. 1. Application to ADME/Tox and Drug Discovery Datasets. Journal of Chemical Information and Modeling. 2015 Jun 22;55(6):1231-45.
2. Ekins S, Mestres J, Testa B. In silico pharmacology for drug discovery: applications to targets and beyond. British Journal of Pharmacology. 2007 Sep;152(1):21-37.
3. Ekins S. Progress in computational toxicology. Journal of Pharmacological and Toxicological Methods. 2014 Mar;69(2):115-40.
4. Hecht D. Applications of machine learning and computational intelligence to drug discovery and development. Drug Development Research. 2011 Feb;72(1):53-65.
5. Hou T, Wang J, Li Y. ADME Evaluation in Drug Discovery. 8. The Prediction of Human Intestinal Absorption by a Support Vector Machine. Journal of Chemical Information and Modeling. 2007 Nov;47(6):2408-15.
6. Korotcov A, Tkachenko V, Russo DP, Ekins S. Comparison of Deep Learning With Multiple Machine Learning Methods and Metrics Using Diverse Drug Discovery Data Sets. Molecular Pharmaceutics. 2017 Dec 4;14(12):4462-75.
7. Lantz B. Machine learning with R: learn how to use R to apply powerful machine learning methods and gain an insight into real-world applications. Birmingham: Packt Publ; 2013.
8. Maltarollo VG, Gertrudes JC, Oliveira PR, Honorio KM. Applying machine learning techniques for ADME-Tox prediction: a review. Expert Opinion on Drug Metabolism & Toxicology. 2015 Feb;11(2):259-71.
9. Shen J, Cheng F, Xu Y, Li W, Tang Y. Estimation of ADME Properties with Substructure Pattern Recognition. Journal of Chemical Information and Modeling. 2010 Jun 28;50(6):1034-41.
10. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5826228/
11. http://lmmd.ecust.edu.cn/admetsar1/
12. https://www.wisconsinwatch.org/2013/04/minnow-reveals-wastewaters-toxic-effects/
13. https://winvector.github.io/DataPrep/EN-CNTNT-Whitepaper-Data-Prep-Using-R.pdf
14. https://rpubs.com/chzelada/275494
15. http://www.hrc.es/bioest/errores_2.html

16. https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/

17. https://towardsdatascience.com/supervised-machine-learning-classification-5e685fe18a6d

6. Anexo

ANN

Asignamos a los ficheros, que contienen variables, un nombre para trabajar más fácilmente:

```
myfile1 <- "T_FHMT_I.xls"
```

Ahora se importan los datos en formato data.frame para poder trabajar con ellos:

```
muestra <- read_xls(myfile1)
```

Finalmente, podemos generar una representación SMILES de una molécula usando

```
SMILES<-muestra$SMILES
mols <- parse.SMILES(SMILES)
get.SMILES(mols[[1]])

[1] "C1=CC(C=O)=CC(OC)=C1OCCCCCC"



a<-unlist(lapply(mols, get.SMILES))
head(a)

                C1=CC(C=O)=CC(OC)=C1OCCCCCC
              "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
    C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O
  "C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O"
  C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2
"C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2"
                      CC1=C(NC=O)C=CC=C1Cl
                    "CC1=C(NC=O)C=CC=C1Cl"
          CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1
        "CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1"
          C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C
        "C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C"
```

```
dc <- get.desc.categories()
dc
```

```
[1] "hybrid"         "constitutional" "topological"    "electronic"
[5] "geometrical"
```

```
descNames <- unique(unlist(sapply(get.desc.categories(), get.desc.name
s)))
descNames
```

```
 [1] "org.openscience.cdk.qsar.descriptors.molecular.WHIMDescriptor"
 [2] "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor"
 [3] "org.openscience.cdk.qsar.descriptors.molecular.XLogPDescriptor"
 [4] "org.openscience.cdk.qsar.descriptors.molecular.WeightDescriptor"
```

```
allDescsfinal <- eval.desc(mols, descNames)

allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
any(is.na(x)) )]
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
length(unique(x)) == 1 )]

CAS<-muestra$CAS
Labels<-muestra$Labels
datafinal<-cbind(CAS,allDescsfinal,Labels)

rownames(datafinal) <- c()

datafinal<-datafinal[-1]

matrizc<-cor(datafinal)
write.xlsx(matrizc, file="correlación.xlsx",row.names=TRUE)

Label<-as.factor(datafinal$Labels)
datafinal_fac<-cbind(datafinal[-175],Label)#Añadir columna

lab.group <- c("no","toxico")
clase <- factor(Label,labels=lab.group)

datafinal_facnclase <- datafinal_facn

datafinal_facnclase$no <- clase=="no"
datafinal_facnclase$toxico <- clase=="toxico"

set.seed(1234)
n_train <- 2/3
n <- nrow(datafinal_facnclase)
train <- sample(n,floor(n*n_train))
data_nrm.train <- datafinal_facnclase[train,]
data_nrm.test  <- datafinal_facnclase[-train,]

#step(lm(Labels~.,data=data_nrm.train),direction="backward")

xnam <- names(datafinal_facnclase[1:174])
```

```
(fmla <- as.formula(paste("no+toxico ~ ",  paste(xnam, collapse= "+"))
))
```

```
no + toxico ~ BCUTw.1l + BCUTw.1h + BCUTc.1l + BCUTc.1h + BCUTp.1l +
    BCUTp.1h + XLogP + MW + LipinskiFailures + nRotB + MLogP +
    nAtomLAC + nAtomP + nAtomLC + nB + nBase + nAtom + nAromBond +
    naAromAtom + ALogP + ALogp2 + AMR + nAcid + nSmallRings +
    nAromRings + nRingBlocks + nAromBlocks + nRings3 + nRings5 +
    nRings6 + nRings7 + tpsaEfficiency + Zagreb + WPATH + WPOL +
    WTPT.1 + WTPT.2 + WTPT.3 + WTPT.4 + WTPT.5 + VAdjMat + VABC +
    TopoPSA + topoShape + PetitjeanNumber + MDEC.11 + MDEC.12 +
    MDEC.13 + MDEC.14 + MDEC.22 + MDEC.23 + MDEC.24 + MDEC.33 +
    MDEC.34 + MDEC.44 + MDEO.11 + MDEO.12 + MDEO.22 + MDEN.11 +
    MDEN.12 + MDEN.13 + MDEN.22 + MDEN.23 + MDEN.33 + khs.sCH3 +
    khs.dCH2 + khs.ssCH2 + khs.tCH + khs.dsCH + khs.aaCH + khs.sssCH +
    khs.tsC + khs.dssC + khs.aasC + khs.aaaC + khs.ssssC + khs.sNH2 +
    khs.ssNH + khs.aaNH + khs.tN + khs.dsN + khs.aaN + khs.sssN +
    khs.ddsN + khs.aasN + khs.sOH + khs.dO + khs.ssO + khs.aaO +
    khs.sF + khs.dsssP + khs.sSH + khs.dS + khs.ssS + khs.aaS +
    khs.dssS + khs.ddssS + khs.sCl + khs.sBr + khs.sI + Kier1 +
    Kier2 + HybRatio + fragC + FMF + ECCEN + SP.0 + SP.1 + SP.2 +
    SP.3 + SP.4 + SP.5 + SP.6 + SP.7 + VP.0 + VP.1 + VP.2 + VP.3 +
    VP.4 + VP.5 + VP.6 + VP.7 + SPC.4 + SPC.5 + SPC.6 + VPC.4 +
    VPC.5 + VPC.6 + SC.3 + SC.4 + SC.5 + SC.6 + VC.3 + VC.4 +
    VC.5 + VC.6 + SCH.3 + SCH.4 + SCH.5 + SCH.6 + SCH.7 + VCH.3 +
    VCH.4 + VCH.5 + VCH.6 + VCH.7 + C1SP1 + C2SP1 + C1SP2 + C2SP2 +
    C3SP2 + C1SP3 + C2SP3 + C3SP3 + C4SP3 + ATSp1 + ATSp2 + ATSp3 +
    ATSp4 + ATSp5 + ATSm1 + ATSm2 + ATSm3 + ATSm4 + ATSm5 + ATSc1 +
    ATSc2 + ATSc3 + ATSc4 + ATSc5 + nHBDon + nHBAcc + bpol +
    apol
```

```
mydata_model <- neuralnet(fmla,
                          data = data_nrm.train,
                          hidden=1)


# obtain model results
model_results <- compute(mydata_model, data_nrm.test[1:174])$net.resul
t

# Put multiple binary output to categorical output
maxidx <- function(arr) {
  return(which(arr == max(arr)))
}
idx <- apply(model_results, 1, maxidx)
prediction <- factor(idx,levels=c(1,2),labels=lab.group )
res <- table(prediction, clase[-train])
conf_matrix<- confusionMatrix(res)
conf_matrix
```

```
Confusion Matrix and Statistics


prediction  no toxico
    no      46     14
```

```
    toxico  17    108

               Accuracy : 0.8324
                 95% CI : (0.7707, 0.8832)
    No Information Rate : 0.6595
    P-Value [Acc > NIR] : 1.172e-07

                  Kappa : 0.6226

 Mcnemar's Test P-Value : 0.7194

            Sensitivity : 0.7302
            Specificity : 0.8852
         Pos Pred Value : 0.7667
         Neg Pred Value : 0.8640
             Prevalence : 0.3405
         Detection Rate : 0.2486
   Detection Prevalence : 0.3243
      Balanced Accuracy : 0.8077

       'Positive' Class : no
```

Árbol de decisión

Asignamos a los ficheros, que contienen variables, un nombre para trabajar más facilmente:

```
myfile1 <- "T_FHMT_I.xls"
```

Ahora se importan los datos en formato data.frame para poder trabajar con ellos:

```
muestra <- read_xls(myfile1)
```

Finalmente, podemos generar una representación SMILES de una molécula usando

```
SMILES<-muestra$SMILES
mols <- parse.SMILES(SMILES)
get.SMILES(mols[[1]])

[1] "C1=CC(C=O)=CC(OC)=C1OCCCCCC"



a<-unlist(lapply(mols, get.SMILES))
head(a)

                C1=CC(C=O)=CC(OC)=C1OCCCCCC
              "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
    C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O
  "C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O"
```

```
  C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2
"C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2"
                       CC1=C(NC=O)C=CC=C1Cl
                     "CC1=C(NC=O)C=CC=C1Cl"
          CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1
        "CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1"
          C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C
        "C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C"



dc <- get.desc.categories()
dc

[1] "hybrid"         "constitutional" "topological"    "electronic"
[5] "geometrical"



descNames <- unique(unlist(sapply(get.desc.categories(), get.desc.name
s)))
descNames
```

```
 [1] "org.openscience.cdk.qsar.descriptors.molecular.WHIMDescriptor"
 [2] "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor"
 [3] "org.openscience.cdk.qsar.descriptors.molecular.XLogPDescriptor"
 [4] "org.openscience.cdk.qsar.descriptors.molecular.WeightDescriptor"
```

```
allDescsfinal <- eval.desc(mols, descNames)

allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
any(is.na(x)) )]
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
length(unique(x)) == 1 )]

CAS<-muestra$CAS
Labels<-muestra$Labels
datafinal<-cbind(CAS,allDescsfinal,Labels)

rownames(datafinal) <- c()

datafinal<-datafinal[-1]

matrizc<-cor(datafinal)
write.xlsx(matrizc, file="correlación.xlsx",row.names=TRUE)

Label<-as.factor(datafinal$Labels)
datafinal_fac<-cbind(datafinal[-175],Label)#Añadir columna

set.seed(1234)
n_train <- 2/3
n <- nrow(datafinal_fac)
train <- sample(n,floor(n*n_train))
data_nrm.train <- datafinal_fac[train,]
data_nrm.test  <- datafinal_fac[-train,]
```

```
data.model<- C5.0(data_nrm.train[-175], data_nrm.train$Label)
data.predict<- predict(data.model, data_nrm.test)
confusionMatrix<-confusionMatrix(data.predict, data_nrm.test$Label)
confusionMatrix

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  51  11
         1  12 111

               Accuracy : 0.8757
                 95% CI : (0.8193, 0.9195)
    No Information Rate : 0.6595
    P-Value [Acc > NIR] : 1.574e-11

                  Kappa : 0.7221

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.8095
            Specificity : 0.9098
         Pos Pred Value : 0.8226
         Neg Pred Value : 0.9024
             Prevalence : 0.3405
         Detection Rate : 0.2757
   Detection Prevalence : 0.3351
      Balanced Accuracy : 0.8597

       'Positive' Class : 0
```

Naive Bayes

Asignamos a los ficheros, que contienen variables, un nombre para trabajar más facilmente:

```
myfile1 <- "T_FHMT_I.xls"
```

Ahora se importan los datos en formato data.frame para poder trabajar con ellos:

```
muestra <- read_xls(myfile1)
```

Finalmente, podemos generar una representación SMILES de una molécula usando

```
SMILES<-muestra$SMILES
mols <- parse.SMILES(SMILES)
get.SMILES(mols[[1]])
```

```
[1] "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
```

```
a<-unlist(lapply(mols, get.SMILES))
head(a)
```

```
                C1=CC(C=O)=CC(OC)=C1OCCCCCC
              "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
    C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O
  "C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O"
  C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2
"C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2"
                       CC1=C(NC=O)C=CC=C1Cl
                     "CC1=C(NC=O)C=CC=C1Cl"
          CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1
        "CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1"
          C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C
        "C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C"
```

```
dc <- get.desc.categories()
dc
```

```
[1] "hybrid"         "constitutional" "topological"    "electronic"
[5] "geometrical"
```

```
descNames <- unique(unlist(sapply(get.desc.categories(), get.desc.name
s)))
descNames
```

```
 [1] "org.openscience.cdk.qsar.descriptors.molecular.WHIMDescriptor"
 [2] "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor"
 [3] "org.openscience.cdk.qsar.descriptors.molecular.XLogPDescriptor"
 [4] "org.openscience.cdk.qsar.descriptors.molecular.WeightDescriptor"
```

```
allDescsfinal <- eval.desc(mols, descNames)

allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
any(is.na(x)) )]
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
length(unique(x)) == 1 )]

CAS<-muestra$CAS
Labels<-muestra$Labels
datafinal<-cbind(CAS,allDescsfinal,Labels)

rownames(datafinal) <- c()

datafinal<-datafinal[-1]

matrizc<-cor(datafinal)
write.xlsx(matrizc, file="correlación.xlsx",row.names=TRUE)
```

```
Label<-as.factor(datafinal$Labels)
datafinal_fac<-cbind(datafinal[-175],Label)#Añadir columna

datafinal_facs <-data.frame(lapply(datafinal_fac,as.factor))
str(datafinal_facs)
```

```
'data.frame':   554 obs. of  175 variables:
 $ BCUTw.1l        : Factor w/ 194 levels "11.69","11.79",..: 4 110 3
 $ BCUTw.1h        : Factor w/ 428 levels "12.1000824042221",..: 184
 $ BCUTc.1l        : Factor w/ 553 levels "-0.43778723751488",..: 230
  [list output truncated]
```

```
datafinal_facn<-datafinal_facs[-175] #Eliminamos la variable de los da
tos.

set.seed(1234)
n_train <- 2/3
n <- nrow(datafinal_facn)
train <- sample(n,floor(n*n_train))
data_nrm.train <- datafinal_facn[train,]
data_nrm.test  <- datafinal_facn[-train,]

#step(lm(Labels~.,data=data_nrm.train),direction="backward")

datafinal_training<-Label[train]
datafinal_test<-Label[-train]

mydata_clsf <- naiveBayes(data_nrm.train, datafinal_training, laplace=
0)
test_pred <- predict(mydata_clsf, data_nrm.test)
confusionMatrix<-confusionMatrix(test_pred,datafinal_test)
confusionMatrix

Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 49 30
         1 14 92

               Accuracy : 0.7622
                 95% CI : (0.6942, 0.8216)
    No Information Rate : 0.6595
    P-Value [Acc > NIR] : 0.001614

                  Kappa : 0.5011

 Mcnemar's Test P-Value : 0.023739

            Sensitivity : 0.7778
            Specificity : 0.7541
         Pos Pred Value : 0.6203
         Neg Pred Value : 0.8679
```

```
             Prevalence : 0.3405
         Detection Rate : 0.2649
   Detection Prevalence : 0.4270
      Balanced Accuracy : 0.7659

       'Positive' Class : 0
```

Ramdon Forest

Asignamos a los ficheros, que contienen variables, un nombre para trabajar más fácilmente:

```
myfile1 <- "T_FHMT_I.xls"
```

Ahora se importan los datos en formato data.frame para poder trabajar con ellos:

```
muestra <- read_xls(myfile1)
```

Finalmente, podemos generar una representación SMILES de una molécula usando

```
SMILES<-muestra$SMILES
mols <- parse.SMILES(SMILES)
get.SMILES(mols[[1]])

[1] "C1=CC(C=O)=CC(OC)=C1OCCCCCC"



a<-unlist(lapply(mols, get.SMILES))
head(a)

                C1=CC(C=O)=CC(OC)=C1OCCCCCC
               "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
    C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O
  "C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O"
  C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2
"C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2"
                      CC1=C(NC=O)C=CC=C1Cl
                     "CC1=C(NC=O)C=CC=C1Cl"
          CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1
        "CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1"
          C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C
        "C(C1=CC=CC=C1)(C2=CC=CC=C2)(O)C#C"



dc <- get.desc.categories()
dc
```

```
[1] "hybrid"         "constitutional" "topological"    "electronic"
[5] "geometrical"
```

```
descNames <- unique(unlist(sapply(get.desc.categories(), get.desc.name
s)))
descNames
```

```
 [1] "org.openscience.cdk.qsar.descriptors.molecular.WHIMDescriptor"
 [2] "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor"
 [3] "org.openscience.cdk.qsar.descriptors.molecular.XLogPDescriptor"
 [4] "org.openscience.cdk.qsar.descriptors.molecular.WeightDescriptor"
```

```
allDescsfinal <- eval.desc(mols, descNames)

allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
any(is.na(x)) )]
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
length(unique(x)) == 1 )]

CAS<-muestra$CAS
Labels<-muestra$Labels
datafinal<-cbind(CAS,allDescsfinal,Labels)

rownames(datafinal) <- c()

datafinal<-datafinal[-1]

matrizc<-cor(datafinal)
write.xlsx(matrizc, file="correlación.xlsx",row.names=TRUE)

Label<-as.factor(datafinal$Labels)
datafinal_fac<-cbind(datafinal[-175],Label)#Añadir columna

set.seed(1234)
n_train <- 2/3
n <- nrow(datafinal_fac)
train <- sample(n,floor(n*n_train))
data_nrm.train <- datafinal_fac[train,]
data_nrm.test  <- datafinal_fac[-train,]

rf <- randomForest(Label ~ ., data  = data_nrm.train)
predict.fr<- predict(rf, data_nrm.test[-175])
confusionMatrix<-confusionMatrix(data_nrm.test$Label, predict.fr)
confusionMatrix

Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0  50  13
         1  15 107

               Accuracy : 0.8486
                 95% CI : (0.7887, 0.897)
```

```
    No Information Rate : 0.6486
    P-Value [Acc > NIR] : 9.91e-10

                  Kappa : 0.6656

 Mcnemar's Test P-Value : 0.8501

            Sensitivity : 0.7692
            Specificity : 0.8917
         Pos Pred Value : 0.7937
         Neg Pred Value : 0.8770
             Prevalence : 0.3514
         Detection Rate : 0.2703
   Detection Prevalence : 0.3405
      Balanced Accuracy : 0.8304

       'Positive' Class : 0
```

SVM

Asignamos a los ficheros, que contienen variables, un nombre para trabajar más fácilmente:

```
myfile1 <- "T_FHMT_I.xls"
```

Ahora se importan los datos en formato data.frame para poder trabajar con ellos:

```
muestra <- read_xls(myfile1)
```

Finalmente, podemos generar una representación SMILES de una molécula usando

```
SMILES<-muestra$SMILES
mols <- parse.SMILES(SMILES)
get.SMILES(mols[[1]])

[1] "C1=CC(C=O)=CC(OC)=C1OCCCCCC"



a<-unlist(lapply(mols, get.SMILES))
head(a)

                C1=CC(C=O)=CC(OC)=C1OCCCCCC
              "C1=CC(C=O)=CC(OC)=C1OCCCCCC"
    C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O
  "C1(OC)=C([N+]([O-])=O)C(C=O)=CC(Br)=C1O"
  C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2
"C1=CC(Cl)=CC=C1OC2=C([N+](=O)[O-])C=CC=C2"
                       CC1=C(NC=O)C=CC=C1Cl
                     "CC1=C(NC=O)C=CC=C1Cl"
```

```
          CCCCOC(=O)C1=CC=CC(C(=O)OCCCC)=C1


dc <- get.desc.categories()
dc
```

```
[1] "hybrid"         "constitutional" "topological"    "electronic"
[5] "geometrical"
```

```


descNames <- unique(unlist(sapply(get.desc.categories(), get.desc.name
s)))
descNames
```

```
 [1] "org.openscience.cdk.qsar.descriptors.molecular.WHIMDescriptor"
 [2] "org.openscience.cdk.qsar.descriptors.molecular.BCUTDescriptor"
 [3] "org.openscience.cdk.qsar.descriptors.molecular.XLogPDescriptor"
 [4] "org.openscience.cdk.qsar.descriptors.molecular.WeightDescriptor"
```

```


allDescsfinal <- eval.desc(mols, descNames)

allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
any(is.na(x)) )]
allDescsfinal <- allDescsfinal[, !apply(allDescsfinal, 2, function(x)
length(unique(x)) == 1 )]

CAS<-muestra$CAS
Labels<-muestra$Labels
datafinal<-cbind(CAS,allDescsfinal,Labels)

rownames(datafinal) <- c()

datafinal<-datafinal[-1]

matrizc<-cor(datafinal)
write.xlsx(matrizc, file="correlación.xlsx",row.names=TRUE)

Label<-as.factor(datafinal$Labels)
datafinal_fac<-cbind(datafinal[-175],Label)#Añadir columna

set.seed(1234)
n_train <- 2/3
n <- nrow(datafinal_fac)
train <- sample(n,floor(n*n_train))
data_nrm.train <- datafinal_fac[train,]
data_nrm.test  <- datafinal_fac[-train,]

mydata_model <- ksvm(Label ~ ., data = data_nrm.train,
                     kernel = "vanilladot")

 Setting default kernel parameters

mydata_predict <- predict(mydata_model, data_nrm.test)
res <- table(mydata_predict, data_nrm.test$Label)
(confusionMatrix <- confusionMatrix(res))
```

```
Confusion Matrix and Statistics


mydata_predict   0   1
             0  47  14
             1  16 108

               Accuracy : 0.8378
                 95% CI : (0.7767, 0.8878)
    No Information Rate : 0.6595
    P-Value [Acc > NIR] : 4.438e-08

                  Kappa : 0.6362

 Mcnemar's Test P-Value : 0.8551

            Sensitivity : 0.7460
            Specificity : 0.8852
         Pos Pred Value : 0.7705
         Neg Pred Value : 0.8710
             Prevalence : 0.3405
         Detection Rate : 0.2541
   Detection Prevalence : 0.3297
      Balanced Accuracy : 0.8156

       'Positive' Class : 0
```