UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Desarrollo de herramientas computacionales basadas en reglas para la identificación de biomarcadores de cáncer en el espacio extendido metabólico

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Martín Làzaro, Héctor

Tutor/a: Carbonell Cortés, Pablo Jorge

Cotutor/a externo: AMARA, ADAM

CURSO ACADÉMICO: 2021/2022

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Ingeniería de Sistemas y Automática

Desarrollo de herramientas computacionales basadas en reglas para la identificación de biomarcadores de cáncer en el espacio extendido metabólico

Trabajo Fin de Máster

Máster Universitario en Automática e Informática Industrial

AUTOR/A: Martín Làzaro, Héctor

Tutor/a: Carbonell Cortés, Pablo Jorge

Cotutor/a externo: AMARA, ADAM

CURSO ACADÉMICO: 2021/22

El objetivo de este proyecto es el desarrollo de una herramienta computacional predictiva para identificar biomarcadores de diferentes tipos de cáncer a partir de muestras humanas, analizadas por métodos ómicos como la metabolómica. A partir de una lista de reglas de reacción bioquímicas elementales, la herramienta generará una base de datos extensiva de reacciones y metabolitos putativos. La herramienta proporcionará para cada muestra analizada una lista de posibles reacciones y metabolitos. Se han utilizado algoritmos de aprendizaje automático para predecir los mejores candidatos a biomarcadores.

El proyecto se ha realizado en colaboración con la Agencia Internacional para la Investigación del Cáncer de la OMS (IARC/WHO) en Lyon, que también ha proporcionado las muestras.

Palabras Clave: Cáncer, metabolómica, biomarcador, aprendizaje automático, predictivo

The objective of this project is the development of a computational predictive tool to identify different types of cancer using human samples, analyzed metabolomics. Starting with a list of biochemical elemental reaction rules and metabolites, the tool will provide a list of possible reactions and metabolites for each analyzed sample. Machine learning algorithms have been used to find the best biomarker candidates.

The project has been realized in collaboration with International Agency for Research on Cancer (IARC/WHO) in Lyon, which has also provided the samples.

Key Words: Cancer, metabolomics, biomarker, machine learning, predictive

Índice

- Documento I: Memoria
- Documento II: Presupuesto

Índice de la memoria

Índice de figuras

Índice de tablas

DOCUMENTO I:

MEMORIA

1.- INTRODUCCIÓN

El cáncer es una de las enfermedades más comunes del mundo y una de las principales causas de muerte de los seres humanos [1]. Por este motivo, multitud de estudios, equipos de investigación y científicos dedican sus esfuerzos a encontrar métodos para combatir esta enfermedad. En la actualidad existen muchos proyectos enfocados a encontrar tratamientos para sanar esta enfermedad, a la par que detectar futuros casos. En nuestro caso vamos a centrarnos en la identificación del cáncer.

No obstante, gracias al desarrollo de la tecnología y del aprendizaje automático se abre todo un nuevo campo de estudio que puede aplicarse a esta identificación. Utilizando mecanismos del *machine learning* podemos obtener herramientas que nos faciliten esa identificación, pudiendo llegar a detectar posibles cánceres antes de que aparezcan. De esta forma, podrían utilizarse tratamientos menos invasivos y aumentar las posibilidades de sobrevivir a esta enfermedad.

En este proyecto vamos a hacer uso del *machine learning* para encontrar indicadores de un posible cáncer, de forma que se pueda tratar antes de que se expanda. Estos indicadores son llamados biomarcadores y gracias al estudio del metabolismo podemos utilizarlos. Mediante es estudio de las reacciones y los metabolitos que participan y se forman, podemos encontrar estos biomarcadores y trazar una ruta hasta otros metabolitos presentes en enfermos de cáncer.

Todo este trabajo se realizará en colaboración con el Centro Internacional de Investigaciones sobre el Cáncer, IARC, ubicado en Lyon, Francia. Ellos nos proporcionarán todas las muestras y se encargarán de la parte química del proyecto, además de analizar todos los datos que obtengamos para comprobar su veracidad.

2.- OBJETIVOS

Los objetivos a alcanzar en el presente Trabajo Fin de Máster son los siguientes:

- Desarrollo de una herramienta computacional para identificar biomarcadores.
- Generar una base de datos extensiva de reacciones y metabolitos.
- Uso de algoritmos de aprendizaje automático para predecir candidatos a biomarcadores.
- Identificar nuevos biomarcadores para el cáncer.

También se tiene como objetivo secundario el siguiente:

- Familiarizarse con el lenguaje de programación Python y con las diferentes librerías que ofrece, tanto para la química informática como el aprendizaje automático.

3.- ANTECEDENTES

Se ha realizado el presente trabajo teniendo como punto de partida los siguientes antecedentes:

- Primero de todo, los conocimientos de programación aprendidos durante el Grado de Ingeniería de las Tecnologías Industriales (GITI), como en el Máster de Automática e Informática Industrial (MAII). También los conocimientos sobre aprendizaje automático adquiridos en dicho máster.
- En segundo lugar, los análisis realizados por el IARC para la obtención de los datos sobre los metabolitos y los diferentes pacientes.
- Los trabajos realizados con anterioridad por el tutor con el módulo de RDKit.

4.- CONCEPTOS PREVIOS

4.1.- Machine Learning

El aprendizaje automático o en inglés, *machine learning* es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan [2]. Este aprendizaje se logra a través de algoritmos que permiten a los ordenadores identificar patrones en grandes volúmenes de datos y elaborar predicciones.

Estos algoritmos se pueden clasificar de diferentes formas, para el trabajo que nos interesa hablaremos de dos de ellas, el aprendizaje supervisado y el no supervisado.

4.1.1.- Aprendizaje Supervisado

El objetivo de este aprendizaje es obtener una función a partir de nuestra muestra de datos. Con esta, tendremos relacionadas las entradas y salidas del sistema, de modo que, a partir de una nueva serie de datos podremos predecir que valores de salida tendremos (figura 1). Esta salida podrá ser de dos tipos: un valor numérico, en cuyo caso estaremos ante un problema de regresión, o una etiqueta, un problema de clasificación. Una etiqueta podría ser por ejemplo qué tipo de planta tenemos, a que clase pertenece el objeto, etc [3].

En el ámbito de este trabajo utilizaremos el aprendizaje supervisado para un problema de clasificación.

4.1.2.- Aprendizaje No Supervisado

La diferencia principal con el supervisado es que aquí no existe ningún etiquetado de los datos. En el supervisado nuestra máquina podía saber qué era cada uno de los objetos que recibía. En cambio, en el no supervisado no es capaz de etiquetarlos, solamente de agruparlos de acuerdo a una serie de patrones que encuentra (figura 1). No hay una respuesta buena o una mala, simplemente el resultado que nos ha producido el algoritmo [4]. Esto es porque el algoritmo nunca tiene en cuenta la salida, solamente la entrada. Principalmente se utiliza para resolver problemas de clustering o para comprimir información.

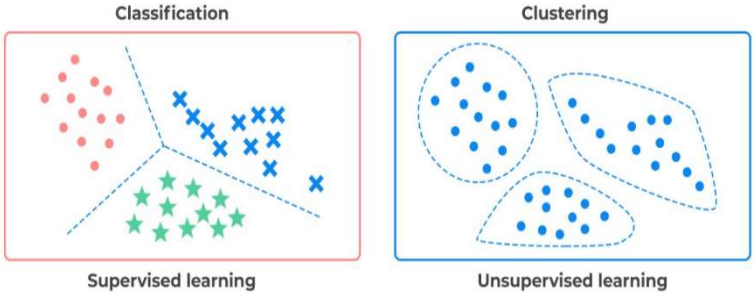

Figura 1: Aprendizaje supervisado y No Supervisado

4.1.3.-Técnicas de clasificación

Como hemos comentado, en nuestro proyecto trataremos un problema de clasificación. El objetivo de este será determinar si un paciente padece o no cáncer. A partir de las entradas que serán las diferentes intensidades de los metabolitos etiquetaremos al paciente entre enfermos o sanos. Para resolver estos problemas existen muchas técnicas diferentes, en nuestro caso nos centraremos en cuatro de ellas:

Árboles de decisiones

Un árbol de decisión (figura 2) es un modelo de predicción basado en la construcción de diagramas que sirven para representar y categorizar una serie de reglas de forma sucesiva [5]. El objetivo es poder crear un modelo que prediga el valor final, en nuestro caso la etiqueta, utilizando reglas simples que aprende de los datos de entrenamiento [6].

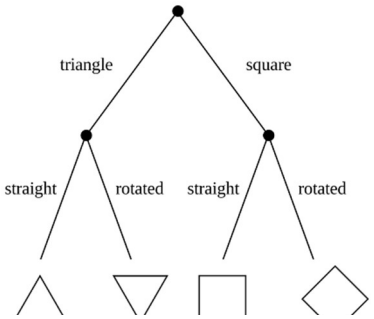

Figura 2. Ejemplo árbol de decisión

Redes neuronales

Las redes neuronales son modelos que imitan el funcionamiento del cerebro humano. Tenemos las neuronas, que se organizan en capas, que van transmitiendo la información desde la entrada hasta la salida. En cada una de estas neuronas los valores de la entrada sufren una transformación, en forma de una suma ponderada, y este resultado es el que se envía a la siguiente capa. Estos valores de ponderación son los que obtenemos a partir del entrenamiento de la red, llamados coeficientes. En nuestro caso utilizaremos una MLP que viene del inglés *Multi-layer Perceptron* [7]. El funcionamiento de este modelo esta explicado en la figura 3. Nuestros datos llegan a la capa de entrada desde donde son enviados a la siguiente capa. En ella se realizan las transformaciones correspondientes y se envían a la siguiente capa y así sucesivamente, hasta llegar a la capa de salida, donde obtenemos el valor final. Si el valor supera un umbral se le asignará una etiqueta, si no lo supera, otra.

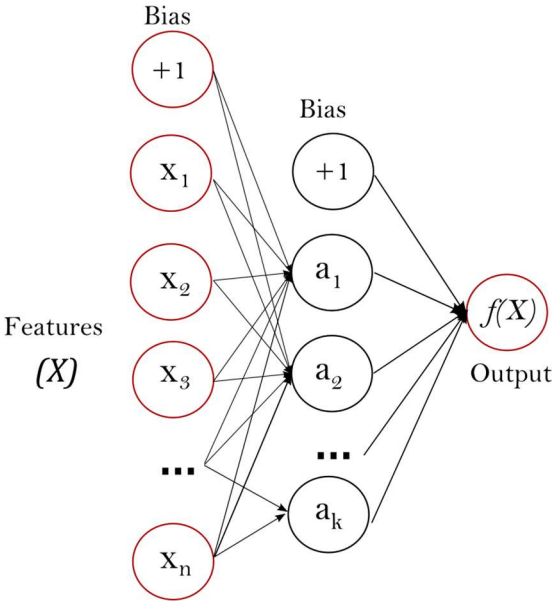

Figura 3. Esquema de una MLP

Máquinas de vectores de soporte

Las máquinas de vectores de soporte o SVM son un conjunto de algoritmos de aprendizaje supervisado. La idea de estos es a partir de los datos de entrenamiento, representar estos en el espacio, separándolos en dos espacios del plano. Como hay infinitos planos que separan el espacio, el algoritmo busca el que tenga una mayor separación a los conjuntos de datos. Este plano estará definido por un vector que llamamos el vector soporte. Para predecir las nuevas muestras se representan en el espacio y según la zona a la que pertenezcan son etiquetadas de una u otra manera. [8]

Clasificador Naive Bayes

Este clasificador está basado en una serie de algoritmos que aplican el teorema de Bayes con la suposición de que las diferentes características de las muestras son independientes. Es decir, asume que la presencia o ausencia de una característica particular no está relacionada con la presencia o ausencia de otra [9]. Una importante ventaja de estos clasificadores es que requieren una cantidad de datos de entrenamiento reducida. Los parámetros a estimar serán las medias y varianzas de las variables y posteriormente aplicar los algoritmos. En nuestro caso utilizaremos el algoritmo Gaussiano de Naive Bayes.

4.2.- Metabolómica y cáncer

La metabolómica es el estudio de los procesos químicos que involucran metabolitos, es decir, el estudio de cualquier molécula utilizada o producida durante el metabolismo. Por ejemplo, gracias a estos estudios sabemos que un metabolito A puede convertirse en C. Mediante una serie de reacciones, este metabolito A se convertirá en B y luego en C, dando lugar a una ruta metabólica. Estas rutas son estudiadas y descritas, de forma que obtenemos un mapa de todo el metabolismo de un ser vivo.

En el caso del cáncer, las células cancerosas también experimentan cambios metabólicos dejando un rastro que podemos detectar. Este rastro es muy relevante en la investigación del cáncer, permitiendo comprender mejor el comportamiento de las células tumorales y ayudando a encontrar biomarcadores. Esto consiste en localizar un metabolito que aparezca en estas rutas durante la aparición del cáncer. Entonces, gracias a este biomarcador, podría detectarse el cáncer con antelación o podrían realizarse fármacos que atacarán ese biomarcador, previniendo la formación del cáncer [10]

Para este análisis de metabolómica se utilizan técnicas como la espectrometría de masas, métodos estadísticos y de aprendizaje automático.

4.3.- RetroRules, una base de datos de reacciones

RetroRules es una base de datos libre de reglas de reacciones para el descubrimiento de rutas metabólicas e ingeniería metabólica. Estas reglas son descripciones genéricas de reacciones químicas que pueden ser utilizadas para enumerar todas las posibles rutas que conectan a una molécula con su predecesora [11].

Un ejemplo de estos datos sería el siguiente:

RR-02-5594c358cc56f547-02-F,"([#8&v2:1](-[#6&v4:2])-[#1&v1:3])>>([#1&+&v0:3].[#8&v2](-[#8&v2:1]-[#6&v4:2])
[#1&v1])","NOEC","1","2","0.0","MNXR94690_MNXM9689","0","0","both","1.0"

De aquí podemos extraer diferente información del dato, pero nos centraremos en la relevante para el trabajo. Primero de todo, tenemos la regla de la reacción propiamente dicha: *"([#8&v2:1](-[#6&v4:2])-[#1&v1:3])>>([#1&+&v0:3].[#8&v2](-[#8&v2:1]-[#6&v4:2]) [#1&v1])"*

Esta regla está expresada en la notación SMARTS, todo lo que está delante del “>>” es el reactivo y lo posterior es el producto. En nuestro ejemplo tenemos dos productos, separados por un punto. Todas las reglas están expresadas de manera que siempre haya solo un reactivo.

En segundo lugar, tenemos el número EC que es básicamente un identificador para la enzima que interviene en la reacción. En nuestro ejemplo, al no intervenir ninguna aparece NOEC, pero otro ejemplo podría ser 1.5.1.21

A continuación, tenemos el diámetro de la regla, esto es, la cantidad de átomos alrededor del centro de la reacción que se están teniendo en cuenta. Conforme más elevado sea este valor, más específica será la regla. El diámetro puede variar desde 2 hasta 18.

Por último, tenemos la referencia a otra base de datos que es MetaNetX. Aquí tenemos dos identificadores, el primero para la reacción (MNXR94690) y el segundo para el metabolito que está reaccionando (MNXM9689).

4.4.- Base de datos inicial: Recon3D

Recon3D es una base de datos de metabolitos y proteínas que representa la mayor red metabólica del ser humano con más de 13000 reacciones que involucran 4000 metabolitos [12]. Los autores la utilizan principalmente para caracterizar mutaciones asociadas con enfermedades e identificar respuestas metabólicas causadas por ciertas drogas. Para nuestro trabajo solo utilizaremos los metabolitos de Recon3D y serán nuestro punto de partida para obtener los nuevos.

Existen ya otras líneas de investigación que han utilizado Recon3D como fuente de datos. Por ejemplo, en un proyecto cuyo objetivo es crear un modelo a escala genómica que ayude a mejorar el tratamiento a pacientes con cánceres resistentes [13].

4.5.- RDKit, una herramienta para la bioinformática

RDKit es una colección de programas informáticos y de machine learning aplicados a la química escritos en C++ y Python [14]. Esta herramienta resulta muy práctica para trabajar con moléculas ya que permite primero de todo, la visualización de estas (figura 4), además de muchas otras aplicaciones. Estas aplicaciones pueden ser desde calcular la masa de la molécula a pasarla a formato SMILE o SMART con una sola función. También nos permite hacer pasar la molécula por una reacción y obtener productos, una función que nos será muy útil como veremos más adelante.

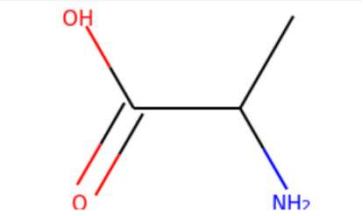

Figura 4: Ejemplo de molécula en RDKit

Esta herramienta ya esta siendo utilizado en otros proyectos de investigación contra el cáncer. Por ejemplo, se ha utilizado RDKit como herramienta química e informática para el diseño de una droga y sus derivados. Esta droga permite inhibir actividades metábolicas y ayuda al tratamiento del cáncer [15]. También se ha utilizado en otra investigación cuyo objetivo era predecir el comportamiento que tendrían diferentes tratamientos para el cáncer combinados. En este caso, el uso que le han dado ha sido como herramienta para obtener las muestras en forma digital y apta para trabajar con machine learning [16].

5.- Solución propuesta

En este apartado vamos a desarrollar la solución que proponemos en nuestro proyecto, pero antes de todo necesitamos saber los procesos previos. Nuestro trabajo parte de base de datos iniciales de metabolómica, obtenidos por el IARC. Estos datos provienen de muestras tomadas a pacientes enfermos de cáncer y a gente sana. Utilizando la espectrometría de masas detectan picos de intensidad, que asocian a una masa concreta. Esto quiere decir que han encontrado alguna molécula con esa masa. Ahora la dificultad reside en identificar el metabolito detectado. En la figura 5 tenemos esquematizado el proceso de obtención de estas bases de datos sobre el cáncer.

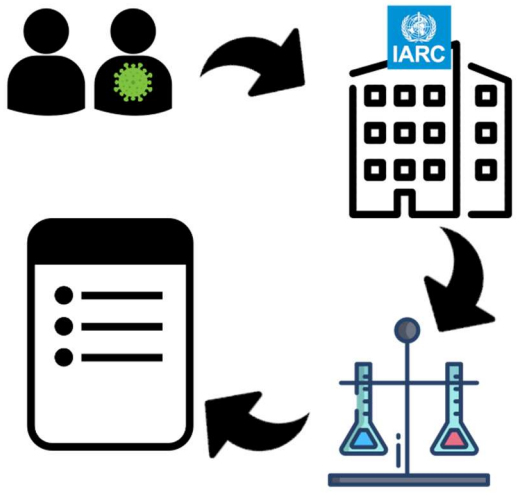

Figura 5: Proceso de obtención de datos de cáncer

Para ello, en nuestro proyecto vamos a generar una lista con multitud de metabolitos, todos ellos generados a partir de reacciones químicas, obtenidas de RetroRules. Con unos metabolitos iniciales, que serán los conocidos en el cuerpo humano, generaremos una red metabólica nueva y muy extensa (figura 6). En esta red podremos buscar si aparece la masa que se ha obtenido en el espectrómetro de masas. Si esto ocurre, sería un primer paso para la identificación del metabolito. Además, como hemos creado una red podemos viajar en sentido contrario, partiendo desde nuestro metabolito hacia los causantes de su formación. Si dichos metabolitos también se encuentran en el espectrómetro, y las reacciones son posibles en el cuerpo humano tendremos muchas más pruebas para afirmar que este metabolito es el correcto.

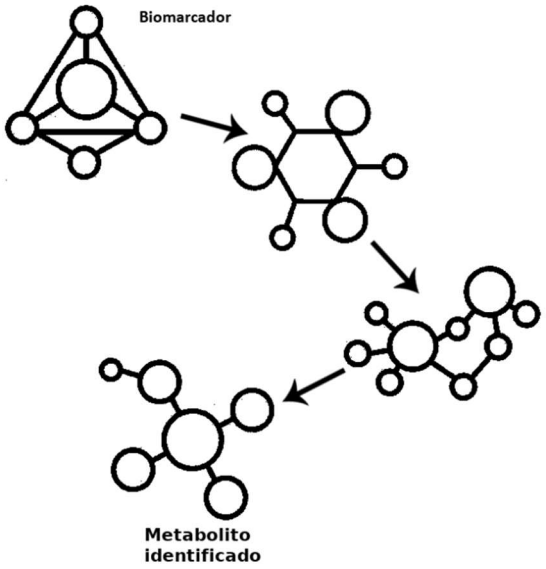

Figura 6: Ejemplo de ruta metabólica

No obstante, encontrar el metabolito no es nuestro objetivo final. Como ya hemos mencionado, queremos encontrar biomarcadores, es decir, moléculas que nos permitan afirmar si una persona padecerá cáncer o no. Para ello realizaremos dos tareas. La primera es ajena a nosotros, el IARC se ocupará de comprobar si los metabolitos encontrados pueden ser o no biomarcadores. En nuestro proyecto, utilizaremos técnicas de aprendizaje automático para examinar si utilizando los metabolitos detectados mejoramos las predicciones realizadas de enfermo o sano.

6.- Implementación

La implementación de la solución se ha desarrollado en diferentes fases, que serían equivalentes a los diferentes *scripts* que se han realizado. Estas serían:

- En primer lugar, se ha desarrollado un generador de nuevos metabolitos, partiendo de las reglas de RetroRules y de los metabolitos de Recon3D.
- En segundo lugar, se han obtenido las masas de estos nuevos metabolitos y se han emparejado con los datos proporcionados por el IARC.
- En tercer lugar, con los resultados anteriores se han preparado los datos para aplicarles el *machine learning*.
- En último lugar, se han realizado las diferentes pruebas de *machine learning* y evaluado los resultados obtenidos.

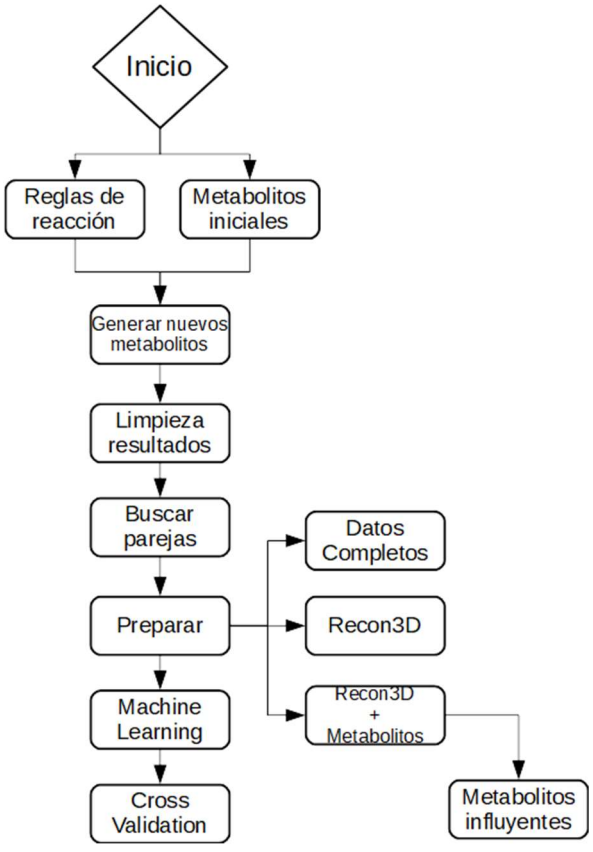

Figura 7: Esquema Workflow del desarrollo

En la figura 7 podemos ver un esquema *Workflow* explicando el desarrollo seguido de forma más detallada. Estas fases explicarían el orden general del trabajo, pero en cualquier momento se han podido realizar modificaciones. Algún error ha podido ser pasado por alto en un principio para ser solucionado en un momento posterior del desarrollo. Ahora procederemos a explicar en detalle el trabajo realizado.

6.1.- Generador de nuevos metabolitos

6.1.1 Obtención de las reglas de reacción

Como hemos explicado anteriormente, el primer paso es desarrollar un generador de nuevos metabolitos. Primero de todo partimos de RetroRules. De la base de datos general se extraen las reacciones y se clasifican según el valor del diámetro. De esta forma, tenemos diez archivos *.csv*, cada uno de ellos con las reacciones de un solo diámetro. Además, en estos archivos hemos eliminado la información que no era necesaria para nosotros.

Ahora que ya tenemos las reacciones listas se procede a leer los datos con Python. Para ello se utiliza la librería Pandas, que nos permite leer los archivos *.csv* y la librería RDKit, que nos es muy útil para trabajar con todo lo relacionado con la química. Solo utilizamos un diámetro a la vez, pudiendo elegir cual queremos al principio del programa. Una vez leídas, se convierte la reacción del formato SMARTS al tipo de variable *ChemicalReaction*, gracias a RDKit. La función que nos permite hacer esto se llama *Chem.rdChemReactions.ReactionFromSmarts().* Aquí tenemos una muestra del código de este apartado (figura 8):

```
df= pd.read_csv('data/Reactions/reactions_d8.csv',usecols=['Reaction','EC Number','MNX ID'])
ec_num=df["EC Number"]
leg_id=df["MNX ID"]
rule=df["Reaction"]

"Obtenemos las reacciones y sus componentes"
r=[0]*len(rule)

for i in range(len(rule)):
    r[i]=Chem.rdChemReactions.ReactionFromSmarts(rule[i])

del(rule)
```

Figura 8: Código para la obtención de reglas de reacción

A lo largo de todo el código vamos eliminando de la memoria variables que ya no vamos a utilizar, para aligerar la carga de computación. Todo se ha realizado en un ordenador personal, por lo que necesitábamos reducirla al máximo para poder ejecutarlo sin problemas.

6.1.2 Obtención de los metabolitos iniciales

Una vez obtenidas las reacciones necesitamos los metabolitos que reaccionarán con ellas. Estos son los que encontramos en Recon3D, es decir, aquellos presentes en el metabolismo humano, pero solo utilizaremos aquellos que tengan su formulación escrita en formato SMILES. Al igual que en las reacciones, mediante Pandas leemos el archivo con los metabolitos.

Usando RDKit convertimos los metabolitos de SMILES a variables tipo *Mol*. Hay 5 metabolitos de Recon3D que dan problemas al convertirlos, así que los hemos eliminado también. Por último, para evitar problemas de formato, todos los metabolitos son pasados por un filtro que los estandariza.

Este filtro no está desarrollado por nosotros, sino que lo hemos obtenido de un repositorio de GitHub [17]. Este filtro utiliza las herramientas que tiene RDKit para normalizar al máximo las moléculas.

```
"Obtenemos los metabolitos de la base de Adam"
df= pd.read_csv('data/original_metabolites.csv', usecols=['smile'])
smile=df['smile']
smile=smile.dropna()

sm=smile.reset_index()
smile=sm['smile']
mol=[]
a=Standardizer()

for i in range(len(smile)):
    m=Chem.MolFromSmiles(smile[i])
    m=a.sequence_rr_legacy(m)
    mol.append(m)
```

Figura 9: Código para la obtención de metabolitos iniciales

Aquí tenemos el fragmento de código correspondiente a este apartado (figura 9). Las funciones *dropna()* y *reset_index()* son dos funciones de pandas que hemos utilizado para eliminar los espacios en blanco de los datos. Estos espacios vienen por los metabolitos que hay en Recon3D pero no tienen su forma SMILE escrita. En el bucle *for* tenemos la función *Chem.MolFromSmiles()* para leer las SMILES y la función siguiente es la correspondiente al filtro ya comentado.

6.1.3 Generación de los nuevos metabolitos

Una vez ya tenemos las reacciones y los metabolitos es el momento de empezar a generar nuevos metabolitos. Para ello, haremos pasar a todos los metabolitos por cada una de las reacciones y ejecutaremos la función de RDKit, *RunReactants*. En el caso de que la reacción y el metabolito no den ningún resultado el algoritmo simplemente seguirá calculando.

Por otro lado, si hemos obtenido un resultado, comprobaremos si la masa de este nuevo metabolito es mayor o menor que 1000 Da. Si es mayor el metabolito es descartado debido a dos motivos. El primero es para aliviar carga computacional, ya que metabolitos grandes es probable que sean polímeros que pueden reaccionar infinitamente, por lo que estaríamos calculando sin beneficio. El segundo motivo, es que no tenemos casi metabolitos con masa mayor de 1000 en nuestros datos ya que, salvo en casos concretos como las proteínas que cuentan con mecanismo de digestión propios en los tejidos, estas moléculas de alto peso molecular son poco metabolizables por el organismo, por lo que es poco probable emparejarlos, así que hemos decidido ahorrar de esta forma.

En el caso de que su masa molecular sea menor de este límite, el nuevo metabolito será anotado en una nueva base de datos, junto a su masa, el metabolito padre, la reacción de la que procede y también el número EC. Además, anotaremos en qué iteración ha sido generado.

Este proceso puede repetirse en nuevas iteraciones, que utilizarán los metabolitos generados en la anterior para generar nuevos.

```
for j in range(ite):
    for mm in mol:
        for i in range(len(r)):
            try:
                pr=r[i].RunReactants( (mm ,) )
                if len(pr) > 0:
                    m= pr[0][0]
                    Chem.SanitizeMol(m)
                    if Descriptors.MolWt(m) < 1000:
                        masa = Descriptors.MolWt(m)
                        productos.append(Chem.CanonSmiles(Chem.MolToSmiles(m)))
                        padre = Chem.MolToSmiles(Chem.RemoveHs(mm))
                        resultado.append(Chem.CanonSmiles(Chem.MolToSmiles(m))+"$"+str(masa)+"$"+padre
            except:
                continue
    productos = list(dict.fromkeys(productos))
    resultado = list(dict.fromkeys(resultado))
    mol= []
    for k in productos:
        m1=Chem.MolFromSmiles(k)
        m1=a.sequence_rr_legacy(m1)
        mol.append(m1)
    productos = []
    print("Empezamos la siguiente iteracion, longitud: ",len(resultado))
```

Figura 10: Código para la generación de nuevos metabolitos

Aquí podemos ver el bucle qué se ejecuta en este apartado (figura 10). Como funciones a destacar, utilizamos la función de RDKit *Chem.SanitizeMol()* para limpiar la molécula y evitar problemas al calcular la masa, y la función *Descriptors.MolWt()* para obtener dicha masa.

Después de toda la obtención de los nuevos metabolitos realizamos dos limpiezas. Con la función *list(dict.fromkeys())* lo que hacemos es eliminar los valores duplicados de nuestras listas. Al crear un diccionario se eliminan los valores repetidos y luego simplemente lo volvemos a convertir en una lista. Los motivos para realizar esta limpieza son dos. Primeramente, para la siguiente iteración eliminamos todos los valores repetidos, así evitamos que calcule lo mismo varias veces. El otro motivo es que a la hora de almacenar estos datos no nos interesan los valores repetidos.

El bucle acaba con la creación de la nueva lista de metabolitos que se utilizará en la siguiente iteración.

6.1.4 Resultados de la generación

Para comprobar si nuestro generador funciona correctamente realizamos una seria de pruebas y analizamos los resultados obtenidos. Estas pruebas consistirán en modificar los valores de diámetro, las iteraciones realizadas o la masa máxima.

Resultados según diámetro

La primera prueba realizada consiste en comprobar cuántos metabolitos obtenemos en cada uno de los diámetros (figura 11). A mayor diámetro, más específicas son las reacciones por lo que esperamos que los resultados sean menores conforme aumentamos el valor. Todas las simulaciones han sido realizadas con una sola iteración y con el límite de masa del metabolito en 1000 Da.

Con un diámetro de 2 obtenemos cerca de 600000 resultados diferentes y este número baja de manera significada hasta llegar al diámetro 18 donde no obtenemos ningún resultado. Como habíamos supuesto, un diámetro 2 es muy general y prácticamente todo reacciona, mientras que uno de 18 es demasiado específico. Cabe destacar que los resultados no son equivalentes a nuevos metabolitos. Para que un resultado sea considerado único basta con que el producto, el metabolito original, la reacción o el número EC sean diferentes del resto de resultados. Esto lleva a que dos metabolitos puedan ser originados por el mismo reactivo, pero en diferentes reacciones o con diferente número EC y los contabilicemos como resultados diferentes. Es decir, de estos 600000 resultados del diámetro 2 habrá metabolitos repetidos.

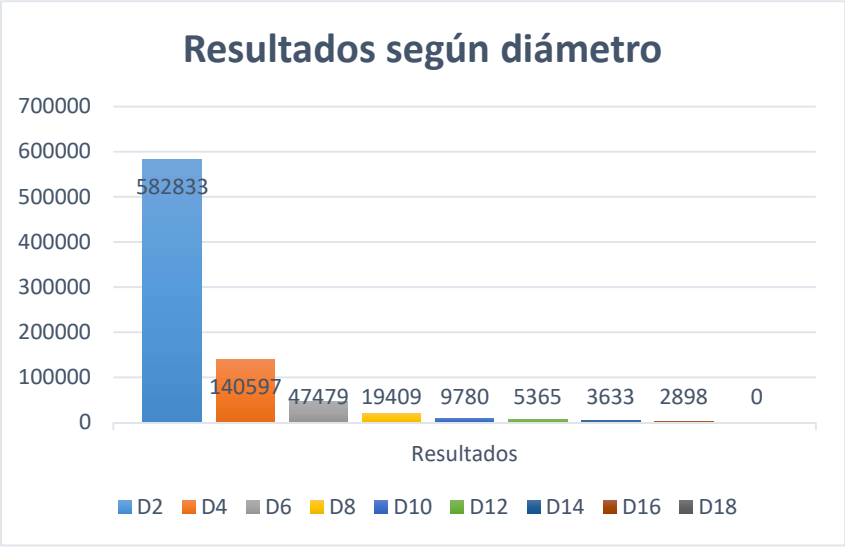

Figura 11: Resultados según diámetro

Resultados por iteración

Para la segunda prueba elegimos el diámetro 8 como valor fijo y mantenemos la masa limitada. Ahora vamos a realizar tres iteraciones con el objetivo de ver si va generando nuevos o se queda estancada y no obtenemos nada. Partimos de los metabolitos de Recon3D, que hemos llamado iteración 0, que son 3227 y los hacemos pasar tres veces por el algoritmo. En la primera iteración obtenemos 19409 resultados, en la segunda 139328 y en la última obtenemos 582098. Estos valores son el total acumulado, por lo que habría que restar el valor anterior para saber exactamente el número obtenido en cada una. Como podemos observar en la figura (figura 12), nuestro algoritmo funciona correctamente, ya que nos está generando cada vez más resultados nuevos.

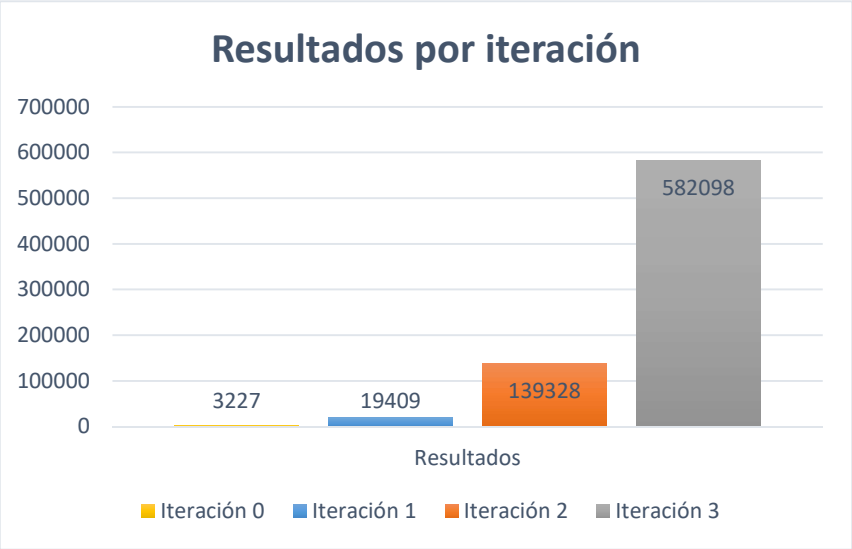

Figura 12: Resultados por iteración

Resultados según masa máxima

Por último, para comprobar si nuestra idea de limitar los nuevos metabolitos a aquellos con masa menor de 1000 Da es necesaria, realizamos otra simulación, pero esta vez quitando este limitador. En este caso, el diámetro sigue siendo de 8 y solo hemos realizado una iteración. Vemos como sí que están apareciendo una buena proporción de metabolitos (figura 13), así que nuestro filtro es

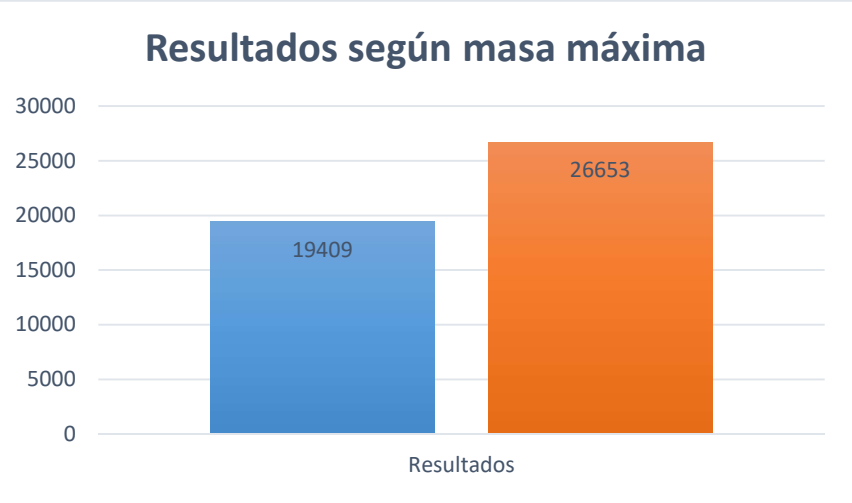

Figura 13: Resultados según masa máxima

necesario. Como ya se ha comentado, masas mayores de 1000 Da es muy probable que correspondan a polímeros que pueden reaccionar de forma infinita saturando nuestro programa.

6.1.5.- Limpieza de los resultados

Antes de seguir con el proceso, hay que modificar los metabolitos que hemos obtenido. Muchas de las reglas de reacción producen más de un metabolito. Por lo tanto, tenemos muchos metabolitos en el formato de *MetabolitoA.MetabolitoB.* Esto implica que, al calcular la masa de esta molécula, suma las masas de los dos metabolitos. Para evitar esto, se ha desarrollado un sencillo algoritmo que busca el punto que separa los metabolitos y devuelve los dos metabolitos separados. Esto tiene que hacerse varias veces, ya que hay algunos que tienen más de dos metabolitos unidos. Muchos de estos son simplemente moléculas de agua que podemos ignorar, pero alguno de ellos aporta información útil.

En cuanto a los resultados, teníamos 582098 de ellos, después de separar hemos contado cuantos metabolitos diferentes tenemos, siendo estos 114011. Esto equivaldría a alrededor del 20% de los resultados totales.

6.2.- Búsqueda de parejas mediante la masa

Con los nuevos metabolitos generados, nuestro objetivo ahora es encontrar cuales de estos pueden aparecer en los pacientes enfermos. Primero de todo explicaremos cómo se presentan los datos:

	GK	GL	GM	
	LivCan_395	LivCan_396	adduct	p
	Non-case	Incident	nan	n
	nan	HCC/	nan	n
3	16953.2003	18156.3009		
3	605123.757	551415.806		
	5922.21347			
7	17960.8208	16394.5752		
9	2330104.36	2091275.23		
5	44145.7872	52985.1077		
7	5960.04132	7330.19452		
5	9018.4714	10532.8784		
3	25549.5591	43315.3611		
2	34374.5225	26468.6809		
7	6946.44927	6429.71351		
5	97082.6168	160734.016	[M+H]+ 75.0684	
		3930.50682		
9	110248.38	106223.957		
L	13029.6151	12102.217		
			[M+H]+ 82.0782	
2	52883.7837	30521.5858		

Figura 14: Muestra de los datos de pacientes

En la figura 14 podemos ver el ejemplo de dos pacientes. Cada una de las filas corresponde a un metabolito encontrado en la persona, siendo el número la intensidad con que aparece el pico en el espectrómetro de masas. El problema es que muchos de estos picos no han podido ser identificados en el análisis de la muestra, son aquellos para los que la columna *adduct* aparece en blanco. Nosotros de momento nos centraremos en los que ha podido ser medida la masa, que aparecen en el formato [M+X] + Y, siendo X un elemento ya conocido, por ejemplo, hidrogeno, e Y la masa que queremos encontrar. Nuestro objetivo será averiguar si alguno de nuestros metabolitos generados tiene la misma masa que alguno de los detectados en los pacientes.

Para ello hemos utilizado los resultados obtenidos del generador de metabolitos utilizando un diámetro de 8 y realizando 3 iteraciones. Estos datos los exportamos a *.csv* para tenerlos guardados también y poder consultarlos. Nuestro programa leerá primero las masas que queremos localizar y los metabolitos generados. De estos solo nos interesará la masa que tienen. Ahora aplicaremos un algoritmo de búsqueda secuencial para encontrar los metabolitos con la masa deseada. Para nuestra búsqueda hemos puesto una tolerancia de ±1e-2. Esta tolerancia es con la que trabajan los equipos del IARC y también la tolerancia utilizada en Recon3D, motivo por el que se ha seleccionado.

Aquí podemos ver el código del propio algoritmo (figura 15). Este pasa por todas las masas que

```
for j in range(len(mass)):
    for k in range(len(masa_met)):
        if abs(mass[j] - masa_met[k]) < tol:
            pair.append([lines[j], k, met[k]])
            mm = 1
    if mm == 1:
        e=lines[j].replace("\n","")
        mass_matched.append(e)
    mm = 0
```

Figura 15: Código para la búsqueda de masas

queremos encontrar y las que tenemos, y vamos restándolas. Si el valor absoluto de esta resta es menor que una tolerancia ya definida asumimos que la masa es igual. En caso de que sea igual almacenamos en una variable la masa encontrada, el metabolito que tiene esa masa y su posición en nuestra lista de metabolitos totales. También tenemos otra variable que solamente almacena esta masa encontrada, en formato [M+X] + Y, que utilizaremos en las siguientes partes del programa. El motivo de tener las dos variables es que más de un metabolito puede emparejarse con la misma masa. Así que, de esta forma evitamos tener la misma masa repetida muchas veces para el programa. Por otro lado, la primera variable nos guarda toda la información sin ninguna pérdida.

También con el mismo algoritmo buscamos las parejas entre los datos de los pacientes y Recon3D. Para ello, igual que en el anterior, obtendremos las masas de Recon3D a partir de sus SMILES y las comparamos mediante búsqueda secuencial con las que queremos localizar. En el caso de Recon3D, encontramos 95 masas.

En la siguiente tabla podemos ver los resultados que hemos obtenido probando con diferentes tolerancias. Como hemos dicho, en nuestro caso utilizaremos la tolerancia de 1e-2, con la que obtenemos un 40% de los metabolitos. Una tolerancia menor encuentra bastantes más, pero al ser tan pequeña no podemos asegurar que sean correctos. En cambio, una tolerancia más alta reduce en gran medida los resultados, por lo que tampoco nos interesa para poder trabajar con el *machine learning*.

Tolerancia	Metabolitos a buscar	Masas encontradas
1e-1	910	760
1e-2	910	395
1e-3	910	70

Tabla 1: Metabolitos emparejados según la tolerancia usada

6.3.- Preparación de los datos para el machine learning

Hasta ahora hemos obtenido nuevos metabolitos y los hemos emparejado con otros desconocidos encontrados en pacientes con cáncer. Ahora nuestro objetivo será ver si estos pueden sernos útiles como herramientas para detectar si un paciente padece la enfermedad o no. Para ello se realizarán una serie de experimentos, descritos más adelante, para los que necesitaremos diferentes datos.

El primero de estos grupos de datos será el grupo completo de los metabolitos proporcionado por el IARC. Estos no necesitan ningún tipo de preparación, así que los utilizaremos tal cual nos los han enviado.

El segundo será el grupo de los emparejados con Recon3D. Primero eliminamos todos aquellos metabolitos cuya masa no ha podido ser encontrada previamente por el IARC. De estos 900 nos quedaremos solo con los 95 que se han emparejado con Recon3D.

El tercer grupo consistirá en Recon3D más los emparejados con los generados. Igual que en el grupo dos, primero eliminamos los que no tienen una masa definida. Ahora vamos a comprobar si hay metabolitos repetidos en el grupo de Recon3D y en los generados, para así evitar contarlos dos veces. De esta forma nos quedamos con 335 metabolitos en este grupo.

El código para generar estos dos grupos consiste simplemente en pasar por la lista que nos ha creado el apartado anterior e ir buscando cada [M+X]. Así vamos añadiendo toda la información del metabolito del IARC y obtenemos los datos como nos interesan.

Los últimos dos grupos consistirán en muestras aleatorias de datos, que nos servirán de control, siendo ambos de 335 metabolitos cada uno. Uno será elegido entre los metabolitos con masa de manera aleatoria, y el otro entre todos los metabolitos de los datos originales. Estos grupos se generarán unas cien veces cada uno, siendo cada vez diferentes.

```
df=pd.read_csv('data2.csv',usecols=lines)
df=df.to_numpy()
df=np.delete(df,186,1)
data=df[1:]
labels=df[0]
a=list(range(0,len(data)))
rand=random.sample(set(a),335)
rand.sort()
result=np.zeros((335,186))
j=0
for i in rand:
    result[j]=(data[i])
    j=j+1
```

Figura 16: Preparación para el machine learning

Para crear estos últimos grupos hemos utilizado este programa (figura 16). Consiste en generar una serie de números aleatorios entre 0 y el número de datos totales. Luego estos números son utilizados para elegir qué datos se usará en el *machine learning*. Podemos ver como también extraemos una variable que son las etiquetas, y también eliminamos alguna fila que no nos interesaba.

6.4.- Métodos de machine learning

Con los datos ya preparados llega el momento de realizar el *machine learning*. Como ya se ha mencionado, vamos a realizar cuatro métodos diferentes de aprendizaje supervisado. Cada uno de ellos se ha implementado en una función diferente que procederemos a explicar. Sin embargo, antes de esto se ha creado otra función, llamada *get_data* (figura 17).

```
def get_data():
    with open('data/headers.txt') as f:
        lines = f.readlines()

    for i in range(187):
        a=lines[i].find('\n')
        lines[i]=lines[i][0:a]

    df=pd.read_csv('data2.csv',usecols=lines)
    df=df.to_numpy()
    df=np.delete(df,186,1)
    scale=StandardScaler()
    data=df[1:]
    labels=df[0]

    imp = IterativeImputer(max_iter=5, random_state=0)
    imp.fit(data)
    x=imp.transform(data)
    x= scale.fit_transform(x)
    y = labels
    x = np.transpose(x)

    return ([x,y])
```

Figura 17: Código de la función get_data()

La tarea de esta función es leer los datos que se han preparado y añadirles unas modificaciones. La primera de estas será estimar los valores en blanco. En nuestros datos existen metabolitos que no se han encontrado en algún paciente por lo que aparecen como espacios en blanco. Esto es un problema para la librería que vamos a utilizar, *scikit-learn*, ya que no está preparada para trabajar con estos valores. Una solución sería descartar estos metabolitos, pero perderíamos información, por lo que optamos por estimarlos. Para ello utilizamos una función que nos proporciona *scikit* llamada *IterativeImputer*.

IterativeImputer modela cada característica con valores en blanco como una función de las otras, y utiliza esa estimación para calcular su valor. Lo hace de forma reiterada, de manera que cada iteración es una característica la que se estima y todas las demás son las entradas [18]. Cuando todas están estimadas nos devuelve los datos con los valores rellenados.

La segunda de estas modificaciones es normalizar los valores. De la misma manera que con los valores en blanco, los modelos de *scikit* funcionan mejor si los datos se parecen a una distribución normal. De hecho, la SVM asume que trabajamos con valores así. Para normalizarlos utilizamos la

función *StandardScaler*, a la que simplemente le damos nuestros valores y nos los devuelve ya normalizados.

Por último, separaremos nuestros datos según sean entradas o etiquetas. Es decir, los valores de los metabolitos serán las entradas, que llamaremos x, y las etiquetas serán si el paciente padecía o no cáncer, que será nuestra y.

Las funciones para cada método son similares, solamente cambiamos el clasificador (tabla 2). Utilizamos cuatro clasificadores con su respectiva función, aquí podemos ver las diferentes funciones (figuras 18, 19, 20, 21).

```
def gaussian():
    from sklearn.naive_bayes import GaussianNB

    matrix=np.zeros((1,2))
    x,y = get_data()

    clf = GaussianNB()
    scores = cross_val_score(clf,x,y,cv=5)
    matrix[0][0]=scores.mean()
    matrix[0][1]=scores.std()
    return matrix
```

Figura 18: Código del método Naive Bayes

```
def svm():
    from sklearn import svm

    matrix=np.zeros((1,2))
    x,y = get_data()

    clf = svm.SVC()
    scores = cross_val_score(clf,x,y,cv=5)
    matrix[0][0]=scores.mean()
    matrix[0][1]=scores.std()
    return matrix
```

Figura 19: Código del método SVM

```
def decision_tree():
    from sklearn import tree

    matrix=np.zeros((1,2))
    x,y = get_data()

    clf = tree.DecisionTreeClassifier()
    scores = cross_val_score(clf,x,y,cv=5)
    matrix[0][0]=scores.mean()
    matrix[0][1]=scores.std()
    return matrix
```

Figura 20: Código del método árbol de decisión

```
def neural_network():
    from sklearn.neural_network import MLPClassifier

    matrix=np.zeros((1,2))
    x,y = get_data()

    clf = MLPClassifier(solver='lbfgs',alpha=1e-5,hidden_layer_sizes=(5,),random_state=1,activation='logistic', max_iter=600)
    scores = cross_val_score(clf,x,y,cv=5)
    matrix[0][0]=scores.mean()
    matrix[0][1]=scores.std()
    return matrix
```

Figura 21: Código del método red neuronal

Modelo	Función
Árbol de decisiones	tree.DecisionTreeClassifier()
Redes neuronales	MLPClassifier()
SVM	svm.SVC()
Gaussiano de Naive Bayes	GaussianNB()

Tabla 2: Función usada en cada modelo de machine learning

Para la red neuronal estamos utilizando una red con una función de activación logística, f(x) = 1 / (1 + exp(-x)), y el *solver*, solucionador en castellano, *lbgfs*. Este es un optimizador de la familia de los métodos cuasi newtonianos.

6.5.- Cross-validation

Para evitar el problema del sobreajuste los datos son divididos en dos grupos, los datos de entrenamiento y los datos de test (figura 22). Para ello utilizaremos el procedimiento de la *cross-validation*, en castellano, validación cruzada. Este consiste en dividir los datos de entrenamiento de forma que en cada iteración una de estas divisiones se utiliza como test. El modelo resultante de estas iteraciones es validado con los datos de test. El resultado entonces será la media de las diferentes iteraciones. De esta manera evitaremos perder datos cuando tenemos un tamaño de muestras reducido.

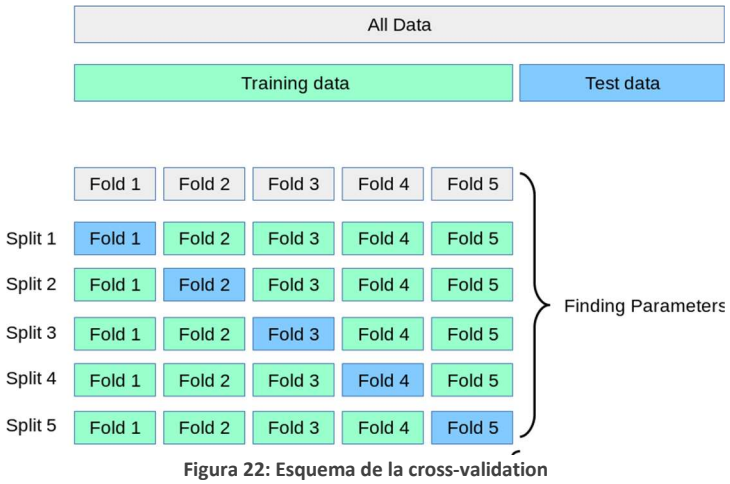

Figura 22: Esquema de la cross-validation

En nuestro experimento dividimos los datos en 5 grupos y luego obtenemos la *accuracy* de nuestro modelo. La *accuracy* es la cantidad de predicciones acertadas, dividido el número total de predicciones.

6.6.- Ajuste de parámetros

Una vez tenemos nuestros modelos preparados vamos a realizar un ajuste de parámetros para encontrar los que nos proporcionan el mejor resultado. En este apartado explicaremos cómo se ha procedido y los parámetros que hemos estado modificando, mientras que en el apartado de experimentos expondremos los resultados y analizaremos con cual nos quedamos. Primero de todo, hemos ajustado los valores del *IterativeImputer*, el algoritmo que nos completa los valores faltantes de nuestros datos, y de los modelos de Gaussiano de Naive Bayes, SVM y red neuronal.

Ajuste de parámetros IterativeImputer

El *IterativeImputer* es, como ya hemos comentado antes, el algoritmo que nos va a rellenar los valores en blanco que aparecen en nuestros datos de entrada. Este algoritmo pasa por todas las entradas y va obteniendo el valor de las faltantes en función del resto de entradas. En este caso vamos a modificar 3 parámetros: la estrategia, el orden de imputación y el número de iteraciones máximas.

La estrategia es cómo va el algoritmo a inicializar estos valores faltantes. Las estrategias que puede utilizar son: la media, la mediana, la moda o una constante. Esta estrategia sirve para obtener el punto de partida, a partir del cual las siguientes iteraciones ajustarán el valor.

El orden de imputación indica qué datos van a ser los primeros en obtenerse. Tenemos 5 formas diferentes de ordenarlos: ascendente, descendente, romana, árabe o aleatoria. El orden ascendente y descendente consisten en empezar por la entrada que tenga menos o más valores en blanco, respectivamente. El orden romano consiste en hacerlo de izquierda a derecha y el orden árabe, al contrario, de derecha a izquierda. Por último, la forma aleatoria sigue un orden aleatorio para cada iteración.

Las iteraciones máximas son el límite de iteraciones que puede realizar antes de devolver los últimos valores calculados. En caso de que el criterio de parada se cumpla antes, el algoritmo no llegará al límite de las iteraciones. El criterio es el siguiente:

$$\frac{\max(abs(X_t - X_{t-1}))}{\max(abs(X[known_{vals}]))} < tol$$

Donde X el valor de la entrada y t la iteración en la que se encuentra.

Hay otros parámetros que pueden ser modificados como la tolerancia del criterio de parada, el estimador para ajustar el valor o el valor máximo y mínimo que puede asignar a una entrada. Estos

parámetros hemos decidido no modificarlos por diferentes motivos. Los valores máximo y mínimo no tienen sentido porque no hay ninguna referencia de cuanto tiene que valer. La tolerancia la hemos dejado en 1e-3 y el estimador está según predefinido, el cual es *BayesianRidge*.

Ajuste de parámetros Gaussiano de Naive Bayes

Para el modelo gaussiano podemos modificar dos parámetros, la prioridad o el alisado de la varianza. En nuestro caso solo hemos ajustado el alisado de la varianza. Este parámetro controla la cantidad de varianza mayor entre todas las entradas que se añade a las varianzas para la estabilidad del cálculo. La prioridad ajusta la importancia que tienen las diferentes clases, pero dado que tenemos solo 2 clases, sano o enfermo, y ambas tienen la misma importancia, la dejamos desactivada.

Ajuste de parámetros SVM

En el caso del modelo SVM hemos modificado el tipo de *kernel* que utiliza el algoritmo. Los diferentes tipos de *kernel* son el lineal, el polinómico, el *rbf* o *kernel* en base radial y el sigmoide.

Ajuste de parámetros de redes neuronales

Para las redes neuronales vamos a modificar dos parámetros la función de activación y el solucionador para la optimización de los pesos. En cuanto a la función de activación tenemos 4 tipos diferentes: identidad, logística, tanh y relu. La función de identidad nos devuelve la función

$$f(x) = x$$

La función logística nos devuelve:

$$f(x) = \frac{1}{1 + e^{-x}}$$

La función tanh:

$$f(x) = \tanh(x)$$

Y la función relu devuelve:

$$f(x) = \max(0, x)$$

El otro parámetro, el solucionador, tiene 3 opciones diferentes: lbfgs, sgd y Adam.

- Lbfgs es un optimizador de la familia de los métodos cuasi newtonianos.
- Sgd se refiere a un descenso de gradiente aleatorio.
- Adam se refiere a un optimizador basado en el gradiente.

6.7.- Metabolitos más influyentes

Otro de los análisis que realizamos sobre los datos es comprobar cuáles de estos son más influyentes. Para ello partimos de los emparejados con Recon3D y los generados. Nuestra función calculará la media de las intensidades con las que aparecen estos metabolitos, una media para los pacientes con cáncer y otra para los sanos. Además de las medias también calculamos las varianzas de cada uno. Con estos dos valores procedemos a realizar a un t-test para cada uno de los metabolitos, concretamente un test de hipótesis nula. De esta manera demostramos si las medias entre pacientes enfermos o sanos son diferentes. En nuestra prueba obtenemos que 78 de estos metabolitos tienen medias diferentes, por lo que pueden ser candidatos a biomarcadores. Una vez tenemos estos metabolitos buscamos la masa correspondiente en nuestra base de datos generada. Anotaremos qué metabolito es el encontrado, el reactivo del que proviene, la reacción que lo ha producido y el número EC de esa reacción.

7.- Experimentos y resultados

7.1.- Resultados del generador

	A	B	C	D	E	F
1	**Index**	**Metabolito Generado**	**Masa**	**Padre**	**Reacción**	**Número EC**
2	0	N=C(O)CCC(NC(=O)c1cc	570.456	Nc1nc(O)c2c	MNXR10002	6.3.1.2
3	1	N=C(O)CCC(NC(=O)c1cc	490.477	Nc1nc(O)c2c	MNXR10003	1.4.1.13
4	2	Nc1nc(O)c2c(n1)NCC(C	674.586	Nc1nc(O)c2c	MNXR10007	6.3.2.2
5	3	Nc1nc(O)c2c(n1)NCC(C	576.567	Nc1nc(O)c2c	MNXR10011	3.5.1.94
6	4	Nc1nc(O)c2c(n1)NCC(C	651.659	Nc1nc(O)c2c	MNXR10013	3.4.19
7	5	Nc1nc(O)c2c(n1)NCC(C	980.628	Nc1nc(O)c2c	MNXR10027	2.7.2.11
8	6	N=C(O)CCC(NC(=O)c1cc	490.477	Nc1nc(O)c2c	MNXR10028	2.4.2.14
9	7	N=C(O)CCC(NC(=O)c1cc	836.669	Nc1nc(O)c2c	MNXR10038	6.3.5.2
10	8	Nc1nc(O)c2c(n1)NCC(C	645.519	Nc1nc(O)c2c	MNXR10114	3.1.1.5
11	9	NCCOP(=O)(O)OCC(O)C	688.588	Nc1nc(O)c2c	MNXR10115	NOEC
12	10	Nc1nc(O)c2c(n1)NCC(C	719.598	Nc1nc(O)c2c	MNXR10115	NOEC
13	11	Nc1nc(O)c2c(n1)NCC(C	719.598	Nc1nc(O)c2c	MNXR10116	NOEC
14	12	Nc1nc(O)c2c(n1)NCC(C	719.598	Nc1nc(O)c2c	MNXR10119	NOEC
15	13	Nc1nc(O)c2c(n1)NCC(C	719.598	Nc1nc(O)c2c	MNXR10119	NOEC
16	14	N=C(O)CCC(NC(=O)c1cc	702.571	Nc1nc(O)c2c	MNXR10301	NOEC
17	15	Nc1nc(O)c2c(n1)NCC(C	475.462	Nc1nc(O)c2c	MNXR10335	1.2.1.88
18	16	Nc1nc(O)c2c(n1)NCC(C	475.462	Nc1nc(O)c2c	MNXR10337	1.5.1.12
19	17	N=c1nc(O)c2c([nH]1)N(	991.922	N=c1nc(O)c2	MNXR10335	1.2.1.88
20	18	N=c1nc(O)c2c([nH]1)N(	991.922	N=c1nc(O)c2	MNXR10337	1.5.1.12
21	19	CCCCCCCCCCCC(=O)OC(	554.702	CCCCCCCCCC	MNXR10012	3.1.3.69
22	20	CCCCCCCCCCCCCCCC(=(	713.138	CCCCCCCCCC	MNXR10297	3.1.1.32
23	21	CCCCCCCCCCCC(=O)O	474.723	CCCCCCCCCC	MNXR10297	3.1.1.32
24	22	CCCCCCCCCCCCCCCC(=(	713.138	CCCCCCCCCC	MNXR10298	NOEC
25	23	CCCCCCCCCCCC(=O)O	474.723	CCCCCCCCCC	MNXR10298	NOEC
26	24	CCCCCCCCCCCCCCCCCC	741.192	CCCCCCCCCC	MNXR10298	NOEC
27	25	CCCCCCCCCCCC(=O)O	474.723	CCCCCCCCCC	MNXR10298	NOEC
28	26	CCCCCCCCCCCC(=O)OC(	554.702	CCCCCCCCCC	MNXR10360	NOEC
29	27	CCCCCCCCCCCC(=O)OC(	490.722	CCCCCCCCCC	MNXR10377	NOEC
30	28	CCCCCC=CCC(O)C=CC=C	827.655	CCCCCC=CCC	MNXR10027	2.7.2.11
31	29	CCCCCC=CCC(O)C=CC=C	492.546	CCCCCC=CCC	MNXR10114	3.1.1.5
32	30	CCCCCC=CCC(O)C=CC=C	535.615	CCCCCC=CCC	MNXR10115	NOEC
33	31	CCCCCC=CCC(O)C=CC=C	566.625	CCCCCC=CCC	MNXR10115	NOEC
34	32	CCCCCC=CCC(O)C=CC=C	566.625	CCCCCC=CCC	MNXR10116	NOEC
35	33	CCCCCC=CCC(O)C=CC=C	566.625	CCCCCC=CCC	MNXR10119	NOEC
36	34	CCCCCC=CCC(O)C=CC=C	566.625	CCCCCC=CCC	MNXR10119	NOEC
37	35	CCCCCC=CCC(O)C=CC=C	322.489	CCCCCC=CCC	MNXR10224	1.2.1.20
38	36	CCCCCC=CCC(O)C=CC=C	774.03	CCCCCC=CCC	MNXR10297	3.1.1.32
39	37	CCCCCC=CCC(O)C=CC=C	805.04	CCCCCC=CCC	MNXR10298	NOEC
40	38	CCCCCC=CCC(O)C=CC=C	833.094	CCCCCC=CCC	MNXR10298	NOEC
41	39	CCCCCC=CCC(O)C=CC=C	774.03	CCCCCC=CCC	MNXR10299	NOEC
42	40	CCCCCC=CCC(O)C=CC=C	805.04	CCCCCC=CCC	MNXR10300	NOEC
43	41	CCCCCC=CCC(O)C=CC=C	833.094	CCCCCC=CCC	MNXR10300	NOEC
44	42	O	354.487	CCCCCC=CCC	MNXR10377	NOEC
45	43	CCCCCC=CCC(O)C=CC=C	322.489	CCCCCC=CCC	MNXR10395	NOEC
46	44	CCCCCC=CCC(C=CC=CC(	843.654	CCCCCC=CCC	MNXR10027	2.7.2.11

Figura 23: Tabla de resultados del generador de metabolitos

En la imagen anterior (figura 23) tenemos una muestra de los resultados que obtenemos del generador. Como ya hemos dicho, este nos proporciona cinco tipos de información: el metabolito generado, la masa de este metabolito, el metabolito del que proviene, que nosotros llamamos padre, la reacción que lo ha producido y el número EC de esta reacción. También tenemos una columna de índice para tenerlos ordenados y saber a qué iteración pertenecen y demás.

Hay varias cosas que comentar de la imagen. Primero de todo, la tabla resultante no es exactamente así, la hemos recortado un poco para que se ajuste a las dimensiones del papel y no ocupe demasiado en la memoria. Esto lleva a algunas incoherencias, como en el metabolito 42 que es el oxígeno y nos dice que su masa son 354 Da. Obviamente esto es un error, la masa corresponde al otro metabolito generado en la reacción, pero como hemos recortado en la foto, éste no aparece. También, muchos metabolitos tienen una expresión larga, por lo que tampoco aparece al completo en la imagen.

Pasando ya a analizar los resultados propiamente, vemos como confirma diversas cosas que hemos ido comentando. Hay metabolitos generados que se repiten, su expresión y masa es igual, pero al proceder de una reacción distinta o con un número EC distinto los contamos como resultado nuevo. También podemos ver que un metabolito puede reaccionar en muchas reacciones diferentes, como en el caso del primero que reacciona en 18. Y, además, vemos como este mismo metabolito con la misma reacción puede producir diferentes resultados o utilizar otro número EC para realizarla. Esto nos da una idea de la enorme cantidad de datos que produce este generador y de lo compleja que puede llegar a ser la red metabólica creada a partir de este. Un trabajo futuro podría ser realizar la red propiamente, no solo tener la información como ahora mismo.

7.2.- Experimento 1: Rude data vs Recon3D vs nuestra data

El primero de los experimentos que se ha realizado ha sido comprobar el efecto de nuestra selección de metabolitos en las predicciones. Para ello, comparamos las predicciones hechas con los datos tal cual los tenemos, *rude data*, solo los metabolitos encontrados en Recon3D y los metabolitos de Recon3D más los nuestros. Estas predicciones las realizamos con los 4 modelos de aprendizaje que hemos explicado anteriormente.

Primero de todo, indicar que trabajamos con 186 pacientes diferentes, de los cuales la mitad padecen cáncer y la otra mitad no. Este número de pacientes se mantendrá constante a lo largo de todos los experimentos. Lo que varía, en este caso, es el número de metabolitos que se tienen en cuenta para la predicción, es decir, el número de entradas del modelo. En este caso, la *rude data* contiene 2388

metabolitos, Recon3D contiene 58 y Recon3D más los generados por nuestro algoritmo incluye 335. Los resultados obtenidos han sido los siguientes (tabla 3):

	Naive Bayes	*SVM*	*Red Neuronal*	*Árbol*
Rude data	67,8 %	60,2 %	61,8 %	60,8 %
Recon3D	50,5 %	54,8 %	58,1 %	53,2 %
Recon3D + Generados	66,6 %	68,3 %	64 %	63,5 %

Tabla 3: Resultados obtenidos en cada grupo de datos según el modelo

En esta tabla mostramos los resultados de *accuracy* que obtiene cada uno de los grupos de datos en cada uno de los modelos. A primera vista, vemos cómo utilizar solo Recon3D empeora notablemente las predicciones obtenidas. Es muy probable que esto sea debido a la gran reducción de datos de entrada entre Recon3D y los datos originales. Una base de datos insuficiente daría lugar a más predicciones erróneas. No obstante, podemos apreciar como utilizando Recon3D más los generados no ocurre esta reducción. De hecho, en 3 de los 4 casos mejora la *accuracy* y en dos de ellos de forma significativa. Teniendo en cuenta que las entradas siguen siendo inferiores en número respecto original, es un gran resultado esta mejora. Utilizando muchos menos metabolitos podemos tener una mejor predicción, lo que significa que estos metabolitos son interesantes para ser utilizados como biomarcadores.

7.3.- Experimento 2: Datos aleatorio contra nuestros datos

En el segundo experimento queremos comprobar si nuestra selección mejora los resultados respecto a una selección aleatoria. El experimento consistió en ir generando una selección de datos aleatoria, a partir de los datos originales y pasarla por los cuatro modelos anteriores. La selección consta de 335 metabolitos diferentes, elegidos sin importar si hay una masa indicada o no. Esta selección la repetimos durante 500 iteraciones, anotando los resultados de *accuracy* que obtenemos en cada una de ellas. Con todos estos datos podemos obtener la siguiente tabla (tabla 4) con los valores medios y la desviación típica de cada uno de los modelos:

Modelo	Media	Des. Típica
Naive Bayes	61,69%	0,038
SVM	58,01%	0,059
Red Neuronal	63,05%	0,042
Árbol de decisión	61,17%	0,037

Tabla 4: Valores medios de accuracy para cada modelo en las 500 iteraciones de los datos originales

Podemos comprobar como nuestra selección obtiene mejores resultados que la media de las selecciones aleatorias. No obstante, para una mayor exactitud vamos a comprobar estadísticamente la aleatoriedad de estos resultados. Para esto, primero tenemos estas gráficas de los resultados de todas las selecciones aleatorias (figuras 24, 25, 26 y 27).

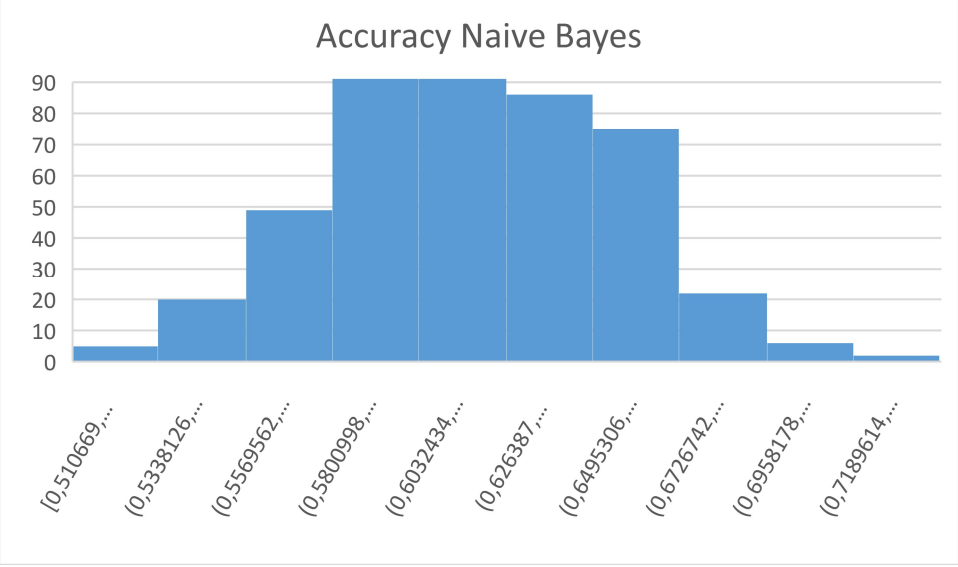

Figura 24: Distribución de la accuracy en Naive Bayes de la selección aleatoria

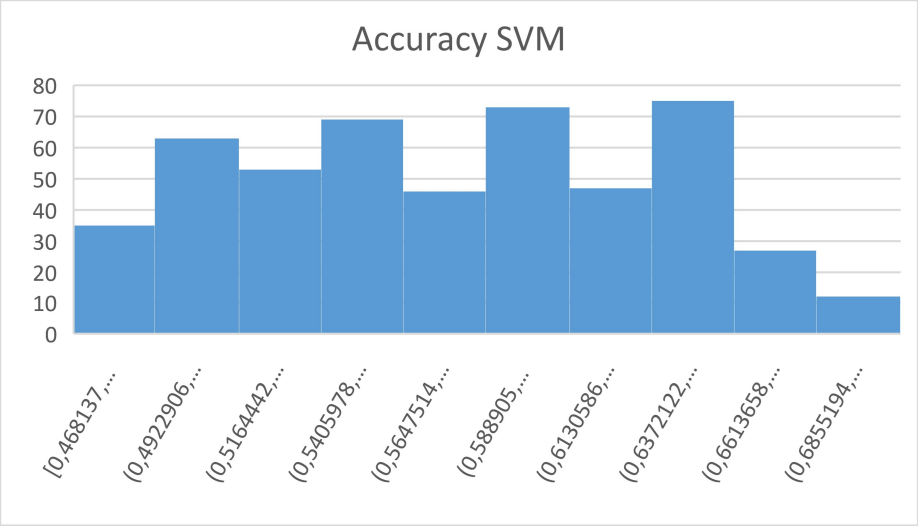

Figura 25: Distribución de la accuracy en SVM de la selección aleatoria

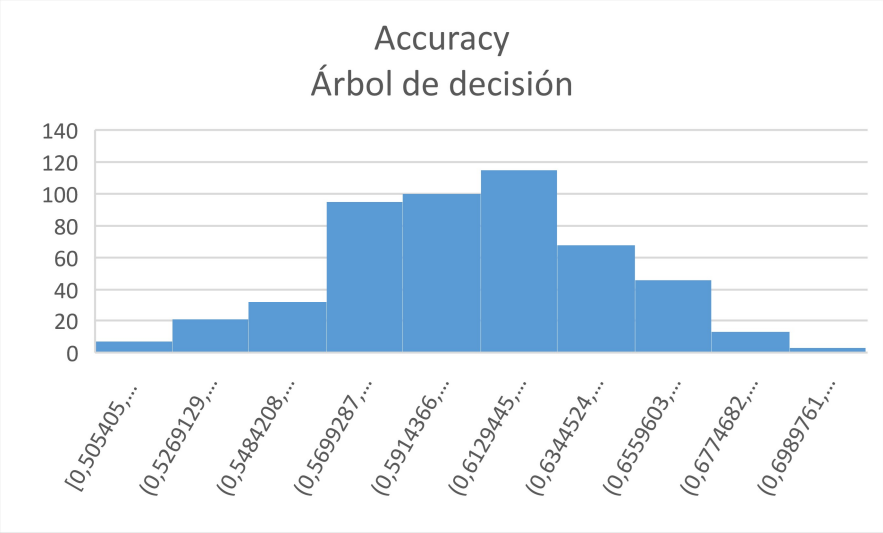

Figura 26: Distribución de la accuracy en árbol de decisión de la selección aleatoria

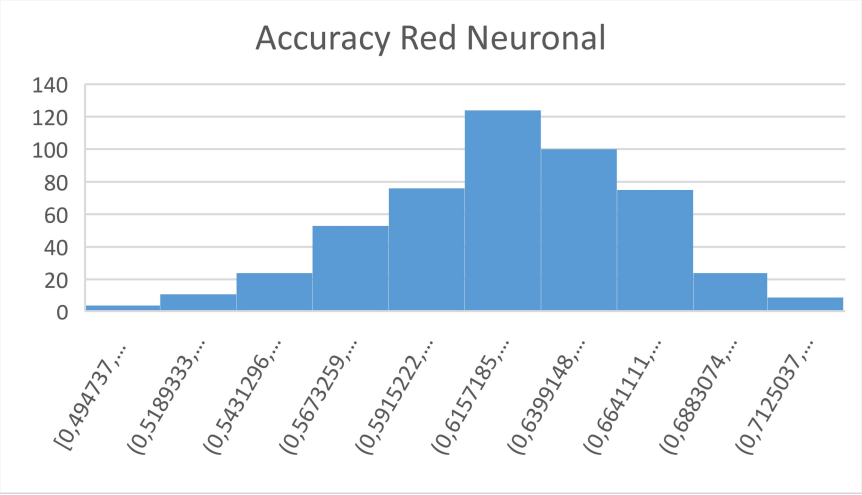

Figura 27: Distribución de la accuracy en red neuronal de la selección aleatoria

Vemos como el modelo de Naive Bayes, la red neuronal y el árbol de decisión se asemejan gráficamente a una distribución normal. No obstante, el modelo SVM no parece ser una normal. Para poder verificar si nuestras distribuciones son normales o no, se realizó un test de Kolmogorov-Smirnov o test K-S. Este test se utiliza normalmente para comprobar la bondad de ajuste de una distribución con otra. Aplicando este test obtenemos que ninguna de las cuatro distribuciones puede ser considerada una normal.

Por consiguiente, vamos a realizar un test K-S entre la distribución aleatoria y la distribución de nuestros datos. Como nuestros datos siempre dan el mismo resultado, necesitamos crear una distribución. Para ello, elegimos aleatoriamente un 80% de nuestros datos y obtenemos una predicción. Este proceso lo repetimos 500 veces como con los datos aleatorios, de modo que obtenemos una distribución con la que comparar. Los resultados que obtenemos con nuestros datos son (tabla 5):

Modelo	Media	Des. Típica
Naive Bayes	65,68%	0,025
SVM	66,97%	0,024
Red Neuronal	66,07%	0,028
Árbol de decisión	63,29%	0,037

Tabla 5: Valores medios de accuracy para cada modelo en las 500 iteraciones de nuestros datos

Con estos datos procedemos a realizar los test K-S para comprobar si las distribuciones son idénticas o no. Para nuestro test, la hipótesis nula es que las distribuciones son iguales. Estos son los resultados obtenidos (tabla 6):

Modelo	Valor P	Conclusión
Naive Bayes	< 1%	Diferentes
SVM	< 1%	Diferentes
Red Neuronal	< 1%	Diferentes
Árbol de decisión	< 1%	Diferentes

Tabla 6: Resultados de los test K-S para cada modelo

Como todos los valores obtenidos son menores que el 1%, podemos rechazar la hipótesis en todos los casos. Así tenemos un argumento para afirmar que nuestra selección de datos influye en el resultado de la predicción de forma positiva y que esta mejora no ha sido por puro azar, sino que tiene una base.

7.4.- Experimento 2.5: Datos aleatorios solo con masa contra nuestra selección

El tercer experimento consiste en una variación del anterior. Se va a realizar el mismo experimento, pero esta vez, la selección de datos aleatorios se realiza solo entre los metabolitos con masa identificada. De igual forma, aquí tenemos la tabla con los resultados obtenidos (tabla 7):

Modelo	Media	Des. Típica
Naive Bayes	64,75%	0,029
SVM	60,31%	0,042
Red Neuronal	63,35%	0,036
Árbol de decisión	62,74%	0,038

Tabla 7: Valores medios de accuracy para cada modelo en las 500 iteraciones de los datos originales

Al igual que en el experimento anterior, obtenemos de media mejores resultados con nuestra selección que con la selección aleatoria. Para una mayor exactitud vamos a comprobar estadísticamente la aleatoriedad de estos resultados. Primero, tenemos estas gráficas de los resultados de las selecciones aleatorias (figuras 28, 29, 30 y 31):

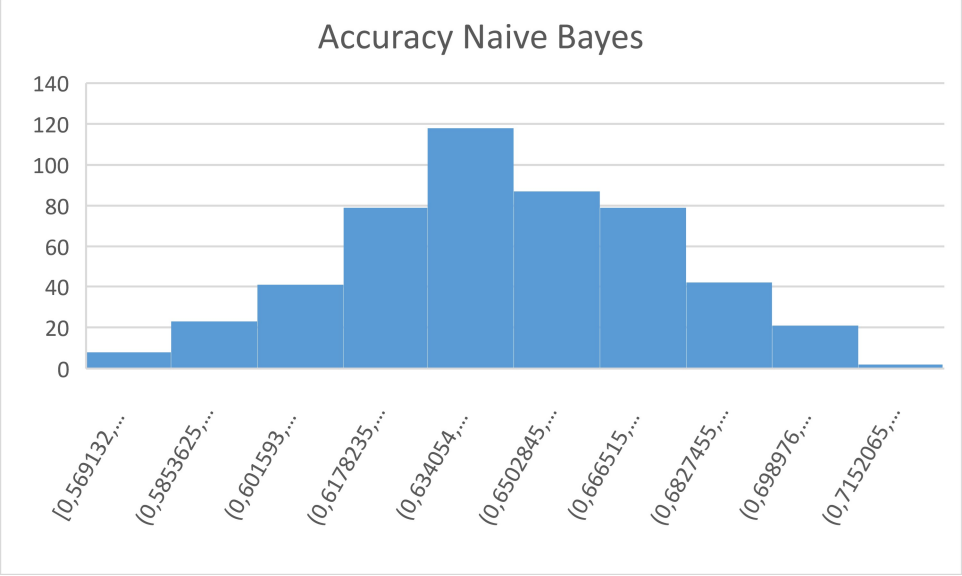

Figura 28: Distribución de la accuracy en Naive Bayes de la selección aleatoria solo masa

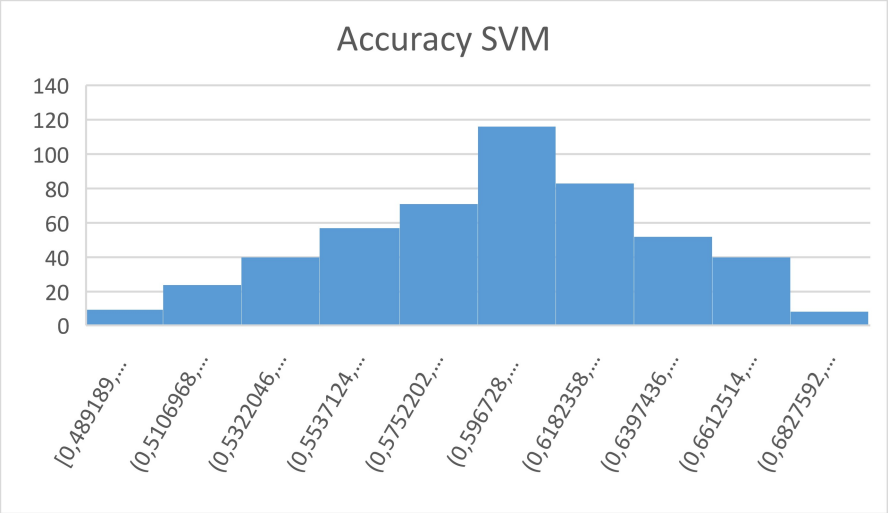

Figura 29: Distribución de la accuracy en SVM de la selección aleatoria solo masa

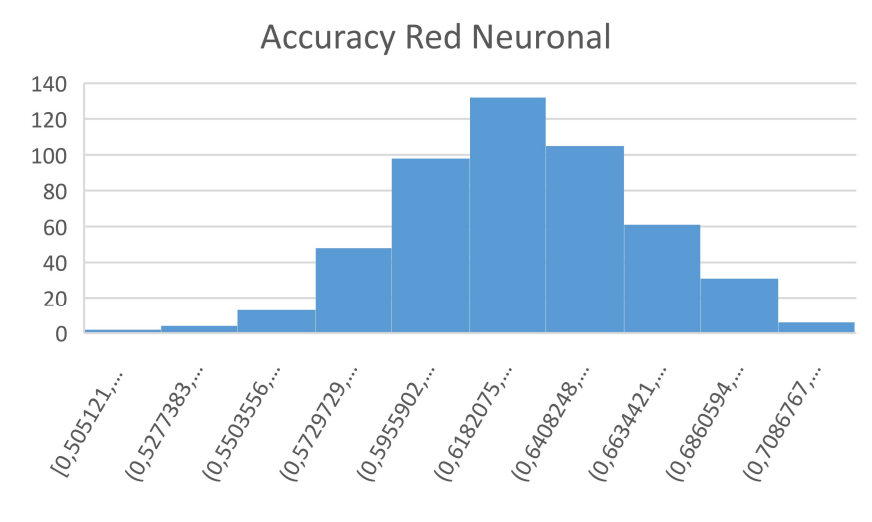

Figura 30: Distribución de la accuracy en red neuronal de la selección aleatoria solo masa

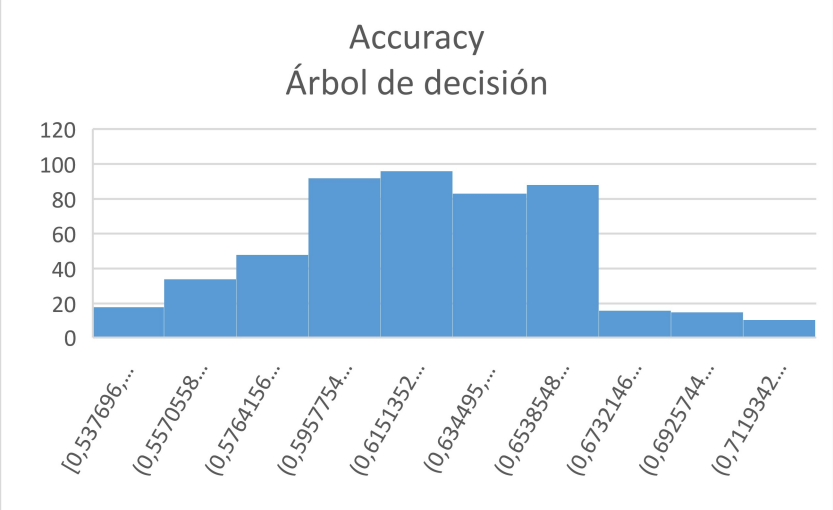

Figura 31: Distribución de la accuracy en árbol de decisión de la selección aleatoria solo masa

Al igual que en el anterior gráfico las distribuciones parecen seguir una distribución normal. Sin embargo, realizando un test K-S para comprobar su normalidad obtenemos que ninguna de ellas la sigue. De forma que, volvemos a repetir el proceso anterior. No hace falta que calculemos una nueva distribución, utilizamos la calculada anteriormente.

En este caso también obtenemos que los 4 modelos tienen un valor p menor del 1%, por lo que podemos afirmar lo mismo. Nuestra selección de datos influye en el resultado de la predicción de forma positiva y que esta mejora no ha sido por puro azar, sino que tiene una base.

7.5.- Experimento 3: Metabolitos más influyentes

Como se ha comentado antes, uno de los análisis realizados es buscar qué metabolitos son más influyentes a la hora de predecir si se padecerá cáncer o no. Para ello tras realizar el t-test a los datos obtenemos los siguientes resultados: de 355 metabolitos encontramos que 78 tienen una media estadísticamente diferentes, mientras que los 257 restantes no. Esto supone un 30% de los metabolitos como posibles influyentes. No obstante, recordar que en nuestra base de datos de pacientes teníamos los metabolitos en forma [M+X] + Y, siendo X algún elemento químico y la Y la masa buscada. Por lo tanto, estos 78 metabolitos no son realmente metabolitos, si no masas de diferentes metabolitos. De modo que diferentes metabolitos pueden tener la misma masa, así que procedemos a obtener estos metabolitos.

	Cantidad
Metabolitos	178
Reactivos	1029
Reacciones	229
Número EC	92

Tabla 8: Metabolitos influyentes encontrados, con los reactivos, reacciones y número EC únicos

Como muestra la tabla (tabla 8), hemos encontrado 178 metabolitos diferentes que pueden ser influyentes. Estos han sido producidos en 229 reacciones diferentes y provienen de 1029 reactivos diferentes. También se han contado las veces que aparecen cada uno estos. De forma que los metabolitos más frecuentes con diferencia son:

- Ácido succínico (O=C(O)CCC(=O)O): aparece en 495 ocasiones, siendo este el 15,98% de las apariciones totales.

- Adenosín difosfato (Nc1ncnc2c1ncn2C1OC(COP(=O)(O)OP(=O)(O)OCC(O)CO)C(O)C1O): aparece en 1559 ocasiones, siendo este el 50,34% de las apariciones totales.

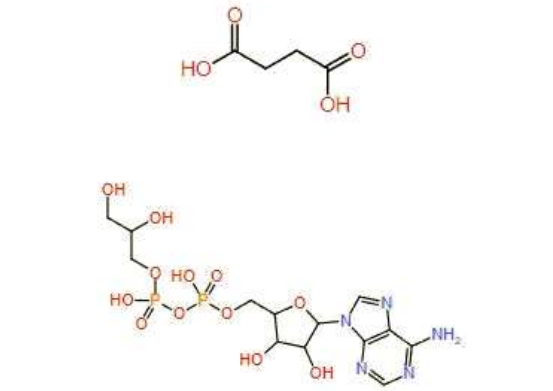

Figura 32: Estructura de los metabolitos más frecuentes

Estos dos metabolitos (figura 32) son bastante comunes en nuestro cuerpo, ya que participan en el ciclo de Krebs, por lo tanto, es esperable que aparezcan tantas veces. De hecho, el siguiente metabolito más frecuente aparece 45 veces, lo que sería más de 10 veces menos, remarcando lo comunes que son. Como pequeña muestra de los resultados (tabla 9), estos serían los 8 metabolitos con más apariciones, excluyendo los ya comentados (figuras 33-40):

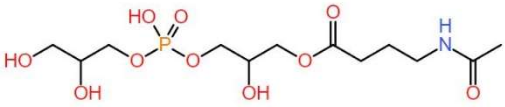

Figura 33: Estructura del metabolito 1

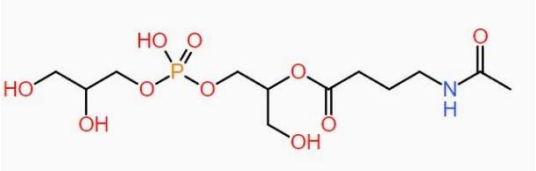

Figura 34: Estructura del metabolito 2

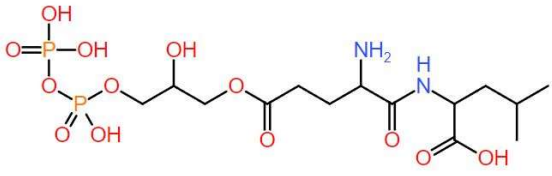

Figura 35: Estructura del metabolito 3

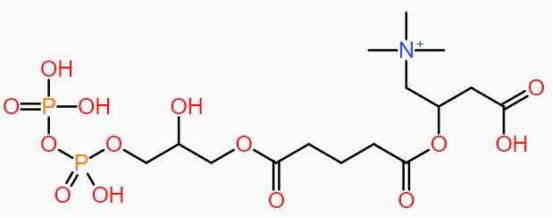

Figura 36: Estructura del metabolito 4

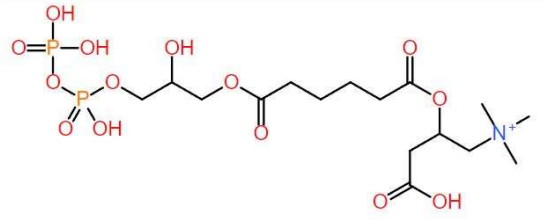

Figura 37: Estructura del metabolito 5

Figura 38: Estructura del metabolito 6

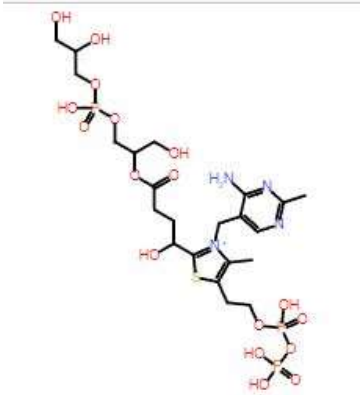

Figura 39: Estructura del metabolito 7

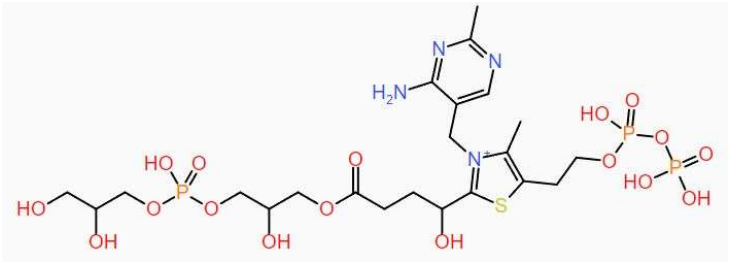

Figura 40: Estructura del metabolito 8

Metabolito	Apariciones	% Aparición
Metabolito 1	45	1,45 %
Metabolito 2	45	1,45 %
Metabolito 3	42	1,36 %
Metabolito 4	41	1,32 %
Metabolito 5	40	1,29 %
Metabolito 6	40	1,29 %
Metabolito 7	37	1,19 %
Metabolito 8	35	1,13 %

Tabla 9: Metabolitos influyentes con más apariciones

Respecto a los reactivos encontramos una distribución mucho más uniforme. No hay ninguno que destaque sobre el resto (tabla 10). Los 4 más repetidos son (figuras 41, 42, 43 y 44):

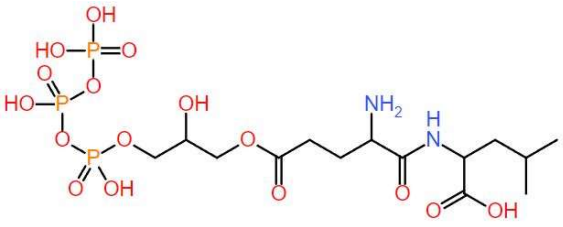

Figura 41: Estructura del metabolito padre 1

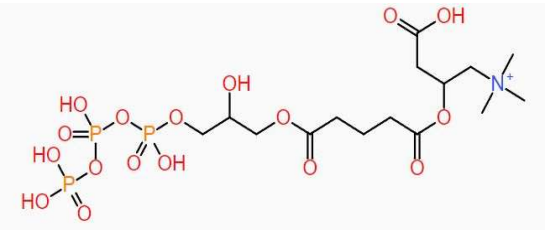

Figura 42: Estructura del metabolito padre 2

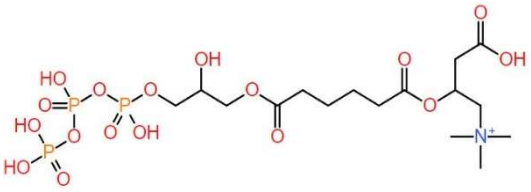

Figura 43: Estructura del metabolito padre 3

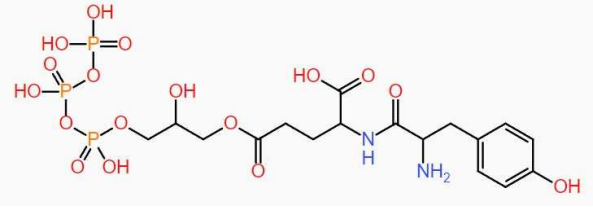

Figura 44: Estructura del metabolito padre 4

Reactivo	Apariciones	% Aparición
Padre 1	34	1,10 %
Padre 2	33	1,07 %
Padre 3	32	1,03 %
Padre 4	32	1,03 %

Tabla 10: Reactivos influyentes con más apariciones

Por lo que se refiere a las reacciones si podemos encontrar algunas que destaquen. Concretamente tenemos 5 de ellas que aparecen sobre las 300 o más veces, dejando las siguiente más frecuentes en torno a las 100 apariciones. Estas reacciones, utilizando el identificador de MetaNetx, son:

- MNXR104300_MNXM1488: aparece 397 veces. Esta reacción produce el ácido succínico por lo que es de esperar que sea común. Su porcentaje de aparición es del 12,81%.
- MNXR101146_MNXM32950: aparece 302 veces, siendo este un 9,75%
- MNXR101151_MNXM32511: aparece 302 veces
- MNXR101158_MNXM6605: aparece 302 veces
- MNXR101160_MNXM6607: aparece 302 veces

Por último, recordamos que el número EC es la identificación de la enzima que interviene en la reacción. Encontramos que lo más frecuente es que no intervenga ninguna enzima, ya que hay bastantes reacciones que no necesitan de su intervención. Esta falta de enzima aparece en 1705 ocasiones, siendo este el 55,04% de las veces. Entre las enzimas propiamente dichas encontramos dos con una frecuencia bastante significativa. Son:

- 3.5.1.18: aparece 400 veces, es la enzima de la reacción del ácido succínico, por lo que su alta aparición era esperable. Esta aparición es el 12,91% de las veces.
- 3.1.1.5: aparece 314 veces, siendo esta el 10,14%. Es la enzima que interviene en la segunda reacción más común, por lo que también era esperable su alta frecuencia.

7.6.- Experimento 4: Pruebas con las masas emparejadas

En este experimento hemos querido comprobar la cantidad de masas emparejadas fuera de los metabolitos ya encontrados. En los datos proporcionado por el IARC, teníamos algunos con la masa indicada en formato [M+X] + Y. No obstante, todos los metabolitos tenían indicada una masa. El objetivo es ver cuantas de estos se emparejan con los nuestros. Para ello, vamos a restar a esa masa los 4 elementos que ya han restado en el IARC: H, Na, NH_4 y K. Luego los pasaremos por el programa para ver que parejas obtenemos. Los resultados obtenidos han sido los siguientes (tabla 11):

Tolerancia	H	Na	NH_4	K
Tol-1	1988	1983	1858	1594
Tol-2	862	878	785	547
Tol-3	106	122	118	74

Tabla 11: Masas emparejadas en el completo de los datos

El total de metabolitos es de 2389, por lo que la misma tabla en porcentajes sería la siguiente (tabla 12):

Tolerancia	H	Na	NH_4	K
Tol-1	83,21%	83%	77,77%	66,72%
Tol-2	36,08%	36,75%	32,86%	22,89%
Tol-3	4,44%	5,1%	4,94%	3,1%

Tabla 12: Porcentaje de masas emparejadas en el completo de los datos

Podemos ver que las 4 pruebas obtienen resultados similares variando la tolerancia.

7.7.- Experimento 5: Ajuste de parámetros

Este experimento tiene como objetivo encontrar los mejores parámetros para nuestras predicciones. De esta forma, vamos variando estos parámetros hasta dar con el que mejor se ajuste. Primero de todo vamos a ajustar los parámetros del *IterativeImputer*. Recordamos que trabajamos con 3 parámetros, iteraciones máximas, la estrategia de inicialización y el orden de imputación. Veamos primero los resultados de variar las iteraciones máximas (tabla 13):

Iteraciones máximas	Naive Bayes	SVM	Red Neuronal	Árbol de decisión
1	0,672	0,699	0,635	0,617
2	0,677	0,677	0,656	0,591
5	0,666	0,683	0,640	0,635
10	0,667	0,672	0,651	0,607

Tabla 13: Accuracy para cada modelo variando las iteraciones máximas de IterativeImputer

Hasta ahora hemos trabajo siempre con un máximo de iteraciones de 5, por lo que utilizaremos estos valores como punto de partida. Además de la precisión de la predicción, hemos observado los valores que daba el IterativeImputer a los huecos en blanco. Primeramente, en ninguno de los casos llega a cumplir el criterio de parada, pero a partir de las 5 iteraciones los valores resultantes son prácticamente los mismos. Realizar solo una iteración es la que mejor resultados nos da en general, pero aquí solamente estaríamos asignándole el valor de la media. Por estos motivos decidimos seguir utilizando 5 como iteraciones máximas.

Una vez decididas las iteraciones vamos a comprobar los otros dos parámetros. Para ello hemos ido emparejando las diferentes opciones. En la siguiente vamos a ir variando el orden de imputación o la estrategia de inicialización (tabla 14):

Estrategia y orden	Naive Bayes	SVM	Red Neuronal	Árbol de decisión
Media Ascendente	0,666	0,683	0,640	0,635
Mediana Ascendente	0,678	0,704	0,696	0,580
Más frecuente Ascendente	0,619	0,683	0,581	0,624
Media Descendente	0,694	0,699	0,661	0,597

Estrategia y orden	Naive Bayes	SVM	Red Neuronal	Árbol de decisión
Mediana Descendente	0,651	0,693	0,666	0,608
Más frecuente Descendente	0,625	0,693	0,682	0,635
Constante Descendente	0,646	0,688	0,640	0,629
Media Romano	0,650	0,694	0,650	0,671
Mediana Romano	0,667	0,699	0,651	0,585
Más frecuente Romano	0,640	0,688	0,630	0,645
Constante Ascendente	0,619	0,683	0,624	0,543
Media Árabe	0,651	0,693	0,661	0,620
Constante Romano	0,662	0,688	0,634	0,597
Mediana Árabe	0,656	0,71	0,694	0,586
Más frecuente Árabe	0,651	0,677	0,672	0,661
Constante Árabe	0,651	0,672	0,678	0,672
Media Aleatoria	0,645	0,683	0,640	0,672
Mediana Aleatoria	0,651	0,677	0,704	0,677
Más frecuente Aleatoria	0,656	0,666	0,672	0,602
Constante Aleatoria	0,656	0,666	0,666	0,619

Tabla 14: Accuracy para cada modelo variando la estrategia y orden del IterativeImputer

De la tabla podemos sacar varias conclusiones. La primera es que todos los valores rondan el 60% - 70% de precisión, algo esperado ya que son los valores que suelen obtener en los experimentos en el IARC. En segundo lugar, cada uno de los diferentes modelos tiene unos parámetros que le vienen mejor lo cual, también es algo esperable. De forma que como conclusión obtenemos que hay que adaptar estos parámetros al modelo. La mejor combinación para cada modelo sería la siguiente:

- Naive Bayes: la combinación de media y descendente nos da una precisión del 69,4%.
- SVM: la combinación de mediana y árabe nos da una precisión del 71%.
- Red Neuronal: la combinación de mediana y aleatoria nos da una precisión del 70,4%
- Árbol de decisiones: la combinación de mediana y aleatoria nos da una precisión del 67,7%.

Ahora vamos a pasar ya a ajustar los parámetros para cada uno de los modelos. Para este ajuste partiremos de los datos anteriores, de modo que cada uno de ellos tendrá su mejor combinación en el *IterativeImputer*.

Por lo que se refiere al modelo Gaussiano de Naive Bayes, el parámetro que vamos a ajustar es el alisado de la varianza. De modo que hemos experimentado con valores desde 1e-1 hasta 1e-9, con los resultados siguientes (tabla 15):

Alisado de la varianza								
1e-1	1e-2	1e-3	1e-4	1e-5	1e-6	1e-7	1e-8	1e-9
0,6612	0,6826	0,6558	0,6989	0,6886	0,6937	0,6937	0,6937	0,6937

Tabla 15: Accuracy para el Gaussiano modificando el parámetro de la varianza

Vemos como los valores de precisión se quedan estancados a partir de un alisado de valor 1e-6. Como son valores ya tan reducidos es difícil que afecten al resultado final. En este caso nuestro mejor resultado lo obtenemos utilizando un alisado de valor 1e-4, con el que nos da un 69,89%

Ahora pasamos a analizar el modelo SVM. En este caso ajustamos el parámetro del tipo de *kernel* entre 4 tipos posibles. Los resultados son (tabla 16):

Tipo de Kernel	Precisión
Lineal	0,7421
Polinomial	0,6989
Base radial	0,7097
Sigmoide	0,6179

Tabla 16: Accuracy para el SVM modificando el parámetro del kernel

Podemos ver como el *kernel* tipo lineal es el que mejor resultado obtiene, con un 74,21% de precisión.

Por último, vamos a ajustar el modelo de red neuronal. En este caso tenemos dos parámetros que ajustar, la función de activación y el solucionador. Para ello, al igual que antes, hemos ido combinándolos entre ellos de todas las formas posibles y este es el resultado (tabla 17):

		Solucionador			
		Identidad	Logístico	Tanh	Relu
Función de activación	Lbfgs	0,6932	0,7037	0,6457	0,4892
	Sgd	0,7370	0,5108	0,7099	0,5054
	Adam	0,5216	0,6832	0,6612	0,5109

Tabla 17: Accuracy para la red neuronal modificando el solucionar y la función de activación

De aquí podemos ver que el solucionador relu es claramente el peor de los 4 y que la combinación de función de activación sgd y la identidad es la mejor con un 73,7% de precisión.

8.- CONCLUSIONES

Para empezar, vamos a comprobar si hemos cumplido los objetivos que nos habíamos propuesto al principio. El primero de todos es generar una base de datos extensiva de reacciones y metabolitos. Podemos afirmar que esta base de datos ha sido lograda. Gracias a RetroRules y a Recon3D hemos creado una base de datos de aproximadamente 600000 datos. Esta base de datos es expandible usando nuevas bases de datos iniciales y la podemos modificar con nuevas iteraciones, o variando el diámetro de la reacción. Los siguientes dos objetivos pueden ir de la mano, usar algoritmos de aprendizaje automático para predecir candidatos a biomarcadores e identificar nuevos biomarcadores. También hemos logrado estos objetivos. Con los diferentes algoritmos hemos probado como mejoraban las predicciones de cáncer en los pacientes, y también hemos encontrado metabolitos concretos que mejoraban estas predicciones. Estos metabolitos son los que podemos indicar como candidatos a biomarcadores. Además, para añadir más posibilidades hemos analizado qué metabolitos influyen más para decidir si un paciente padece o no cáncer, obteniendo una lista con metabolitos, reacciones y enzimas que influyentes. Estos datos se han enviado al IARC, para que ellos se encarguen de comprobar estos metabolitos, si hay posibilidades de que aparezcan, reaccionen, etc. En resumen, todo el tema químico que no es de interés para este trabajo.

Aparte de los resultados obtenidos, este proyecto nos ha servido para obtener conocimientos del lenguaje de programación Python, con las diferentes librerías que hemos utilizado. Algunas de estas son de uso muy extendido, que nos podrían servir para muchos otros proyectos como Pandas o Numpy. Otras como RDKit son mucho más específicas, pero igualmente útiles si continuamos trabajando en proyectos similares a este. También hemos trabajado con conceptos estadísticos aplicables a muchos ámbitos. Asimismo, hemos ampliado los conceptos de *machine learning* obtenidos durante el máster, pudiendo aplicarlos a un proyecto concreto y analizando su comportamiento.

Respecto al futuro, queda pendiente el análisis por parte del IARC de nuestros resultados. Así podremos saber de forma definitiva si nuestros candidatos a biomarcadores lo son realmente. También como proyecto futuro quedaría implementar nuestros códigos al sistema del IARC, para que ellos mismos puedan realizar las modificaciones que necesiten y puedan encontrar biomarcadores para otros tipos de cáncer u otras enfermedades. Por último, nos quedaría publicar el código como una herramienta de código abierto, para que otros investigadores puedan usarla en sus proyectos e investigaciones. Como posible mejora para el código, nos gustaría que generara las rutas metabólicas para obtener cada metabolito generador, de forma que pudiéramos crear todo un mapa con todas las reacciones. Así, cuando encontráramos un biomarcador podríamos recorrer la ruta inversamente y encontrar a los causantes de que aparezca este biomarcador.

9.- Bibliografía

[1] Página web de la OMS, principales causas de defunción. https://www.who.int/es/news-room/fact-sheets/detail/the-top-10-causes-of-death

[2] Página web de la Wikipedia, definición de 'Machine Learning'. https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico

[3] Página web de la Wikipedia, definición de 'Aprendizaje supervisado'. https://es.wikipedia.org/wiki/Aprendizaje_supervisado#Enfoques_y_algoritmos

[4] Página web de TIBCO, definición de 'Aprendizaje no supervisado'. https://www.tibco.com/es/reference-center/what-is-unsupervised-learning

[5] Página web de la Wikipedia, definición de 'Árbol de decisión' https://es.wikipedia.org/wiki/%C3%81rbol_de_decisi%C3%B3n#Acciones_simult%C3%A1neas_en_%C3%A1rboles_de_decisi%C3%B3n

[6] Página web de Scikit-Learn, definición de 'Decision Trees' https://scikit-learn.org/stable/modules/tree.html#classification

[7] Página web de Scikit-Learn, 'Neural Network models' https://scikit-learn.org/stable/modules/neural_networks_supervised.html

[8] Página web de la Wikipedia, definición de 'SVM' https://es.wikipedia.org/wiki/M%C3%A1quinas_de_vectores_de_soporte

[9] Página web de Scikit-Learn, 'Naive Bayes' https://scikit-learn.org/stable/modules/naive_bayes.html

[10] Cardoso, S., Baptista, D., Santos, R., Rocha, M. (2019). A Review on Metabolomics Data Analysis for Cancer Applications. In: Fdez-Riverola, F., Mohamad, M., Rocha, M., De Paz, J., González, P. (eds) Practical Applications of Computational Biology and Bioinformatics, 12th International Conference. PACBB2018 2018. Advances in Intelligent Systems and Computing, vol 803. Springer, Cham. https://doi.org/10.1007/978-3-319-98702-6_19

[11] Duigou T, du Lac M, Carbonell P, Faulon JL. RetroRules: a database of reaction rules for engineering biology. *Nucleic Acids Research*, 2019. | doi: 10.1093/nar/gky940 | PMID: 30321422

[12] Brunk, Elizabeth, Sahoo, Swagatika, Zielinski, Daniel C., Altunkaya, Ali, Dräger, Andreas, Mih, Nathan, Gatto, Francesco, Nilsson, Avlant, Preciat Gonzalez, German Andres, Aurich, Maike Kathrin, Prlić, Andreas, Sastry, Anand, Danielsdottir, Anna D., Heinken, Almut, Noronha, Alberto, Rose, Peter W., Burley, Stephen K., Fleming, Ronan M. T., Nielsen, Jens, Thiele, Ines, and Palsson, Bernhard

O. *Recon3D enables a three-dimensional view of gene variation in human metabolism*. United States: N. p., 2018. doi:10.1038/nbt.4072

[13] Joshua E. Lewis, Tom E. Forshaw, David A. Boothman, Cristina M. Furdui, Melissa L. Kemp, Personalized Genome-Scale Metabolic Models Identify Targets of Redox Metabolism in Radiation-Resistant Tumors, Cell Systems, Volume 12, Issue 1, 2021, Pages 68-81.e11, ISSN 2405-4712, https://doi.org/10.1016/j.cels.2020.12.001

[14] Página web de RDKit https://github.com/rdkit/rdkit

[15] Linxiao Wang, Qian Zhang, Zhe Wang, Wufu Zhu, Ninghua Tan, Design, synthesis, docking, molecular dynamics and bioevaluation studies on novel N-methylpicolinamide and thienopyrimidine derivatives with inhibiting NF-κB and TAK1 activities: Cheminformatics tools RDKit applied in drug design, European Journal of Medicinal Chemistry, Volume 223, 2021, 113576, ISSN 0223-5234, https://doi.org/10.1016/j.ejmech.2021.113576

[16] AUTHOR=Sidorov Pavel, Naulaerts Stefan, Ariey-Bonnet Jérémy, Pasquier Eddy, Ballester Pedro J. TITLE=Predicting Synergism of Cancer Drug Combinations Using NCI-ALMANAC Data JOURNAL=Frontiers in Chemistry, VOLUME=7, YEAR=2019, ISSN=2296-2646 DOI= https://doi.org/10.3389/fchem.2019.00509

[17] Página web de MolVS: Molecule Validaton and Standarization https://molvs.readthedocs.io/en/latest/index.html

[18] Página web de Scikit-learn, 'Imputation of missing values' https://scikit-learn.org/stable/modules/impute.html#iterative-imputer

DOCUMENTO II:

PRESUPUESTO

Índice del presupuesto

1.- Precios materiales y mano de obra

A continuación, se muestra una lista con los costes de materiales, hardware y software necesarios para realizar este proyecto:

	Precio
Licencia de 1 año Microsoft 365	70 €
Determinación de masa con alta resolución por espectrometría de masas con ionización por electrospray	10 €
Análisis de elementos en una disolución	75 €

El precio por hora de un graduado en Ingeniería en Tecnologías Industriales es de 30 €/h, incluyendo la cotización a la seguridad social.

2.- Precios descompuestos

Nº		Descripción de las unidades de obra	Medida	Precio	Importe
01		**Obtención de las muestras**			
	u.	Determinación de masa con alta resolución por espectrometría de masas con ionización por electrospray	100	10,00 €	1000,00 €
	u.	Análisis de elementos en una disolución	1	75,00€	75,00 €
	%	Costos indirectos	2	1.075,00 €	21,50 €
	%	Gastes complementarios	3	1.096,50 €	32,90 €
		TOTAL			1.129,40 €
02		**Estudio y familiarización de RetroRules, Recon3D y RDKit**			
	h	Graduado en Ing. en Tecnologías Industriales	35	30,00 €	1.050,00 €
	%	Costos indirectos	2	1.050,00 €	21,00 €
	%	Gastos complementarios	3	1.071,00 €	32,13 €
		TOTAL			1.103,13 €
03		**Programación del generador de nuevos metabolitos**			
	h	Graduado en Ing. en Tecnologías Industriales	75	30,00 €	2.250,00 €
	%	Costos indirectos	2	2.250,00 €	45,00 €
	%	Gastos complementarios	3	2.295,00 €	68,85 €
		TOTAL			2.363,85 €
04		**Programación del emparejamiento según la masa**			
	h	Graduado en Ing. en Tecnologías Industriales	40	30,00 €	1.200,00 €
	%	Costos indirectos	2	1.200,00 €	24,00 €
	%	Gastos complementarios	3	1.224,00 €	36,72 €
		TOTAL			1.260,72 €
05		**Programación del machine learning**			
	h	Graduado en Ing. en Tecnologías Industriales	60	30,00 €	1.800,00 €
	%	Costos indirectos	2	1.800,00 €	36,00 €
	%	Gastos complementarios	3	1.836,00 €	55,08 €
		TOTAL			1.891,08 €

Nº		Descripción de las unidades de obra	Medida	Precio	Importe
06		**Ajuste de los parámetros del *machine learning***			
	h	Graduado en Ing. en Tecnologías Industriales	55	30,00 €	1.650,00 €
	%	Costos indirectos	2	1.650,00 €	33,00 €
	%	Gastos complementarios	3	1.683,00 €	50,49 €
		TOTAL			1.733,49 €
07		**Análisis de los resultados**			
	h	Graduado en Ing. en Tecnologías Industriales	50	30,00 €	1.500,00 €
	%	Costos indirectos	2	1.500,00 €	30,00 €
	%	Gastos complementarios	3	1.530,00 €	45,90 €
		TOTAL			1.575,90 €
08		**Redacción de los documentos**			
	h	Graduado en Ing. en Tecnologías Industriales	60	30,00 €	1.800,00 €
	%	Costos indirectos	2	1.800,00 €	36,00 €
	%	Gastos complementarios	3	1.836,00 €	55,08 €
		TOTAL			1.891,08 €

3.- Precios unitarios

Nº	Descripción de las unidades de obra	Precio unitario
01	Obtención de las muestras	1.129,40 €
02	Estudio y familiarización de RetroRules, Recon3D y RDKit	1.103,13 €
03	Programación del generador de nuevos metabolitos	2.363,85 €
04	Programación del emparejamiento según la masa	1.260,72 €
05	Programación del *machine learning*	1.891,08 €
06	Ajuste de los parámetros del *machine learning*	1.733,49 €
07	Análisis de los resultados	1.575,90 €
08	Redacción de los documentos	1.891,08 €

4.- Presupuesto base de licitación

UNIDADES DE OBRA:

Obtención de las muestras	1.129,40 €
Estudio y familiarización de RetroRules, Recon3D y RDKit	1.103,13 €
Programación del generador de nuevos metabolitos	2.363,85 €
Programación del emparejamiento según la masa	1.260,72 €
Programación del *machine learning*	1.891,08 €
Ajuste de los parámetros del *machine learning*	1.733,49 €
Análisis de los resultados	1.575,90 €
Redacción de los documentos	1.891,08 €

Presupuesto de ejecución material	**12.948,65 €**
Gastos generales (13%)	1.683,32 €
Beneficio Industrial (6%)	776,92 €
Presupuesto de ejecución por contrata	**15.408,89 €**
IVA (21%)	3.235,87 €
Presupuesto base de licitación	**18.644,76 €**

El presupuesto total asciende a la cantidad de:

DIECIOCHO MIL SEISCIENTOS CUARENTA Y CUATRO EUROS Y SETENTA Y SEIS CÉNTIMOS