

Modelos de *Machine Learning* para la predicción de nuevas moléculas activas en *Ciclooxigenasa-1*



Universitat
Oberta
de Catalunya



UNIVERSITAT DE
BARCELONA

Máster Universitario en
Bioinformática y Bioestadística

Area

Drug design and Structural Biology

Nombre Tutor

Jorge Valencia Delgadillo

Profesor/a responsable de la asignatura

Nuria Pérez Álvarez

Autor

Josep Villar Azorín

Barcelona, febrero de 2023



Esta obra está sujeta a una licencia de Reconocimiento-
NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

FICHA DEL TRABAJO FINAL

| | |
|---|---|
| Título del trabajo: | <i>Modelos de Machine Learning para la predicción de nuevas moléculas activas en Ciclooxygenasa-1</i> |
| Nombre del autor: | <i>Josep Villar Azorín</i> |
| Nombre del consultor/a: | <i>Jorge Valencia Delgadillo</i> |
| Nombre del PRA: | <i>Nuria Pérez Álvarez</i> |
| Fecha de entrega (mm/aaaa): | <i>02/2023</i> |
| Titulación o programa: | <i>Máster Universitario en Bioinformática y Bioestadística</i> |
| Área del Trabajo Final: | <i>Drug design and Structural Biology</i> |
| Idioma del trabajo: | <i>Castellano</i> |
| Palabras clave | <i>Ciclooxygenasa-1, Machine Learning, Python</i> |
| Resumen del Trabajo | |
| <p>La Ciclooxygenasa-1 (COX-1) es una enzima que juega un papel determinante en la síntesis de Prostaglandinas las cuales están relacionadas, entre otras, con el dolor y la inflamación.</p> <p>Se han descubierto una serie de moléculas que inhiben esta síntesis y, por lo tanto, se han desarrollado fármacos para la mitigación de esos síntomas. Estos descubrimientos han sido relativamente recientes en el tiempo y se está investigando constantemente para encontrar nuevas moléculas que permitan obtener fármacos más eficaces.</p> <p>Fruto de estas investigaciones hay disponibles bases de datos de las moléculas estudiadas donde con cuyos resultados se puede conocer si una molécula será activa o no, así como sus diferentes características.</p> <p>El objetivo de este trabajo es definir una serie de modelos de <i>Machine Learning</i> y <i>Deep Learning</i> para poder predecir la capacidad de actividad de nuevas moléculas sobre COX-1.</p> | |

Estos modelos estarán basados en diferentes algoritmos utilizados actualmente en estas áreas utilizando una de estas bases de datos. Como resumen se hará una comparativa entre ellos para determinar cuál es el modelo más adecuado para estas predicciones.

La creación de estos modelos se llevará a cabo con *Python* y diferentes librerías disponibles para la gestión de datos y la implementación de los algoritmos a utilizar.

Abstract

Cyclooxygenase-1 (COX-1) is an enzyme that plays a determining role in the synthesis of Prostaglandins which are related, among others, to pain and inflammation.

A series of molecules have been discovered as inhibitors of this synthesis and, therefore, drugs have been developed to alleviate these symptoms. These discoveries have been relatively recent in time and research is constantly being carried out to find new molecules allowing the obtention of more effective drugs.

As a result of these investigations, databases of the studied molecules are available where the results allow to know if it is possible to know whether a molecule will be active or not, as well as its different characteristics.

The objective of this work is to define a series of Machine Learning and Deep Learning models to be able to predict the if new molecules would be enough actives for COX-1 inhibition.

These models will be based on different algorithms commonly used in these areas where one of the mentioned databases will be used. As a summary a comparison will be made between them to determine which is the most suitable model for these predictions.

The creation of these models will be carried out with Python and different libraries available for data management and the implementation of the algorithms to be used.

Índice

| | | |
|------|---|----|
| 1. | Introducción | 1 |
| 1.1. | Contexto y justificación del Trabajo | 1 |
| 1.2. | Descripción General | 2 |
| 1.3. | Objetivos del Trabajo | 4 |
| 1.4. | Enfoque y método seguido | 4 |
| 1.5. | Planificación del Trabajo | 5 |
| 1.6. | Análisis de Riesgos..... | 7 |
| 1.7. | Breve resumen de productos obtenidos..... | 7 |
| 2. | Estado del Arte | 8 |
| 3. | Desarrollo del Proyecto | 9 |
| 3.1. | Datos iniciales..... | 9 |
| 3.2. | Preparación de los datos..... | 10 |
| 3.3. | Implementación de los Modelos..... | 14 |
| 3.4. | <i>k-Nearest Neighbors</i> | 23 |
| 3.5. | <i>Naïve-Bayes</i> | 26 |
| 3.6. | <i>Support Vector Machines</i> | 29 |
| 3.7. | <i>Random Forest</i> | 33 |
| 3.8. | <i>Deep Learning (Artificial Neural Networks)</i> | 37 |
| 3.9. | Comparación de resultados | 41 |
| 4. | Continuidad del trabajo..... | 43 |
| 5. | Glosario..... | 44 |
| 6. | Bibliografía..... | 45 |
| 7. | Anexos | 48 |
| 7.1. | Librería <i>Python</i> común | 48 |
| 7.2. | Notebook de preparación de datos | 56 |
| | <code>data_setup.ipynb</code> | 56 |
| 7.3. | Notebook de implementación <i>k-NN</i> | 58 |
| | <code>knn_implementation.ipynb</code> | 58 |
| 7.4. | Notebook de implementación <i>Naïve-Bayes</i> | 59 |
| 7.5. | Notebook de implementación <i>SVM</i> | 60 |

| | | |
|------|---|----|
| 7.6. | Notebook de implementación <i>Random Forest</i> | 61 |
| 7.7. | Notebook de implementación <i>Deep Learning</i> | 62 |

Lista de figuras

| | |
|---|----|
| Figura 1: Diagrama de Gantt de la planificación del Proyecto..... | 6 |
| Figura 2 <i>Ejemplo kNN</i> | 23 |
| Figura 3. Métricas para <i>kNN</i> | 24 |
| Figura 4 ROC Curves para <i>kNN</i> | 25 |
| Figura 5. Métricas para <i>Naïve-Bayes</i> | 27 |
| Figura 6 ROC Curves para <i>Naïve-Bayes</i> | 28 |
| Figura 7. Métricas para <i>SVM</i> | 31 |
| Figura 8 ROC Curves <i>Support Vector Machine</i> | 32 |
| Figura 9 Ejemplo de <i>árbol de decisión</i> | 33 |
| Figura 10. Métricas para <i>Random Forest</i> | 35 |
| Figura 11 ROC Curves para <i>Random Forest</i> | 36 |
| Figura 12 Métricas para <i>Deep Learning</i> | 38 |
| Figura 13 ROC Curves para <i>Deep Learning</i> | 40 |

1. Introducción

1.1. Contexto y justificación del Trabajo

Uno de los objetivos que toda sociedad moderna debería cubrir es garantizar el bienestar de aquellos que la componen. Hay varios factores que determinan este bienestar. Uno de los principales es la salud y para mejorar su calidad encontramos a la medicina y los medicamentos. La mitigación del dolor y de las inflamaciones son de los retos más importantes a cubrir.

A principios de los años 70 (Vane J.R., 1971) se descubrió que la Aspirina, un conocido analgésico, inhibía la producción de prostaglandinas, responsables de procesos de dolor e inflamación. Más adelante se descubrió que la Ciclooxygenasa-1 (COX-1 en adelante) estaba relacionada con la síntesis de prostaglandinas. A partir de aquí se empezaron a desarrollar líneas de investigación para obtener moléculas que, como el ácido acetilsalicílico y otros medicamentos antiinflamatorios no esteroides (AINE), pudieran actuar sobre COX-1 con el fin de inhibir la producción de prostaglandinas (Gómez-Luque A, 2005).

Como se puede ver, estos descubrimientos son relativamente recientes y, por lo tanto, hay un amplio campo de investigación para la obtención de nuevas moléculas inhibidoras con el objetivo de obtener nuevos fármacos más eficaces.

Para ayudar en esta investigación los modelos de *Machine Learning* y *Deep Learning* pueden jugar un papel clave. Éstos, junto con la evolución de sistemas de computación más potentes, pueden ayudar a determinar de forma rápida nuevas moléculas inhibidoras, permitiendo un desarrollo más rápido y económico de estos fármacos. Esto puede ayudar cubrir una de estas premisas de bienestar humano más equitativa y generalizadamente.

Este trabajo pretende aportar su grano de arena en esta investigación definiendo diferentes modelos de *Machine Learning* y *Deep Learning* para poder determinar cuáles podrían ser más eficientes a la hora de encontrar moléculas que potencialmente puedan inhibir la producción de prostaglandinas en COX-1.

1.2. Descripción General

Este proyecto está centrado en la búsqueda de un modelo de *Machine Learning* para la búsqueda de forma efectiva de moléculas que sean activas sobre la COX-1 en la inhibición de la producción de prostaglandinas con el fin de obtener nuevos fármacos para la mitigación del dolor y la inflamación.

Para ello se van a estudiar un número limitado de algoritmos conocidos y usados de forma extensiva de *Machine Learning* y *Deep Learning*. Dadas las características de la información de entrada puede haber algoritmos que ofrezcan mejores resultados que otros.

Este número de algoritmos a estudiar es limitado debido a la corta duración del proyecto y son los siguientes:

- *K-Nearest Neighbor* (kNN)
- *Naïve-Bayes*
- *Support Vector Machine* (SVM)
- *Random Forest*
- *Neural Network*

Para cada algoritmo se va a realizar una introducción a éste, la implementación en *Python* utilizando diferentes parametrizaciones propias del algoritmo y un estudio de los resultados para cada parametrización. El desarrollo y ejecución se van a realizar sobre la plataforma *Google Colaboratory* (*Google Colab*).

Todas las ejecuciones del algoritmo se van a realizar con el mismo conjunto de datos, proporcionado por el director del trabajo. Este conjunto de datos será tratado inicialmente con las siguientes acciones:

- Eliminación de características no necesarias para el estudio
- Eliminación de columnas constantes.
- Eliminación de columnas con una baja varianza
- Eliminación de columnas donde la comparación en pares da un nivel de correlación de $\pm 0,8$.

- Estandarización de los datos. Esta acción formará parte del proceso de análisis. Se aplicarán diferentes métodos de transformación y serán evaluados para cada algoritmo.

Se utilizará la proporción 80%-20% de los datos para los subconjuntos de *training-test*.

Finalmente se realizará un estudio comparativo entre los mejores resultados obtenidos para cada modelo.

1.3. Objetivos del Trabajo

El principal objetivo del trabajo es la obtención, dentro de la lista de algoritmos de *Machine Learning* propuestos, el modelo que mejor se ajusta (esto es, el que presenta mejores resultados sobre el conjunto de *test*) sobre los datos de entrada dados.

Como objetivos secundarios los más relevantes serían:

- Introducción de cada uno de los algoritmos a utilizar.
- Encontrar el ajuste los diferentes parámetros que puede requerir cada algoritmo para obtener el rendimiento óptimo de éste. También, como se ha comentado anteriormente, se aplicarán diferentes métodos de transformación de los datos para la obtención del mejor resultado posible.
- Comparación de los diferentes resultados obtenidos.

1.4. Enfoque y método seguido

Como se ha comentado anteriormente, el resultado de este proyecto está centrado en el desarrollo de modelos de *Machine Learning* con el fin de determinar los resultados de cada uno de los modelos propuestos y sus bondades.

Esto será llevado a cabo a partir de una serie de tareas de programación de dichos modelos.

Actualmente entre las principales tecnologías para desarrollos de este tipo, en las que además de la propia aplicación de cada algoritmo se necesita de herramientas potentes para la manipulación de datos, son Python (Welcome to Python.Org, n.d.) y R (R: The R Project for Statistical Computing, n.d.).

Ambas tecnologías proporcionan todo lo necesario para el desarrollo de un proyecto de este tipo y, desde mi punto de vista para el problema que se está tratando, son equivalentes.

Siendo agnóstico en cuanto a ambas tecnologías, la opción elegida para este proyecto ha sido *Python*.

Python es un lenguaje de programación muy eficiente, fácil de aprender y portable a cualquier plataforma existente, desde *Mainframes* a placas programables como *Arduino*. Es ampliamente utilizado en diferentes áreas de las TIC, especialmente en el campo de la Ciencia de Datos y del Aprendizaje Automático.

La principal motivación para esta elección es de carácter meramente práctico puesto que de la forma en que se va a ofrecer el producto no va a requerir de ninguna instalación de *software* y librerías puesto que se va a utilizar *Google Colaboratory* como entorno de trabajo.

Google Colaboratory (*Welcome To Colaboratory - Colaboratory*, n.d.) es una herramienta *online* a la que se puede acceder desde cualquier navegador moderno y que ofrece integración directa y fácil con otras herramientas de Google como por ejemplo *Google Drive*, que es donde puede residir el archivo de entrada de trabajo y almacenar posibles resultados.

R por supuesto es un entorno de trabajo excelente, pero requiere de la instalación de diferentes componentes de software (el *core* de *R*, sin duda el IDE *R Studio*, instalación de diferentes *packages* con las librerías...) que hace menos “portable” de forma inmediata.

Sin embargo, con *Google Colaboratory* se trabaja con *Jupyter notebooks*, que permiten añadir tanto texto formateado (*markdown*) como código *Python*. Dichos notebooks son ejecutados desde los servidores de *Google Colaboratory* y son de fácil exportación e importación.

1.5. Planificación del Trabajo

La ejecución de este trabajo transcurre en poco más de cuatro meses (de final de septiembre a inicio de febrero), teniendo los siguientes hitos principales:

- Definición y plan de trabajo: 28/09/2022 - 17/10/2022
- Fase 1 del desarrollo del trabajo: 18/10/2022 - 21/11/2022
- Fase 2 del desarrollo del trabajo: 22/11/2022 – 24/12/2022
- Cierre de la Memoria y presentación: 27/12/2022 – 15/01/2023
 - Propuesta de cierre de la memoria: 27/12/2022 – 12/01/2023
 - Propuesta de cierre de la presentación: 09/01/2022 – 15/01/2023
- Defensa publica: 23/01/2022 – 03/02/2022

El detalle de las tareas previstas por hito es:

| Hito | Tarea | Inicio | Final |
|-------------------------------------|--|------------|------------|
| Definición y Plan de Trabajo | Revisión documentación | 28/09/2022 | 05/10/2022 |
| | Definición y plan de trabajo | 06/10/2022 | 17/10/2022 |
| Fase 1 desarrollo | Preparacion entorno Google Collaboration | 18/10/2022 | 18/10/2022 |
| | Tratamiento datos de entrada | 19/10/2022 | 21/10/2022 |
| | Desarrollo modelo <i>kNN</i> | 24/10/2022 | 28/10/2022 |
| | Desarrollo modelo <i>Naïve Bayes</i> | 31/10/2022 | 07/11/2022 |
| | Desarrollo modelo <i>SVM</i> | 08/11/2022 | 15/11/2022 |
| | Preparación entregable | 16/11/2022 | 21/11/2022 |
| Fase 2 desarrollo | Desarrollo modelo <i>Random Forests</i> | 22/11/2022 | 29/11/2022 |
| | Desarrollo modelo <i>Deep Learning</i> | 30/11/2022 | 09/12/2022 |
| | Evaluación y comparación de los modelos | 12/12/2022 | 16/12/2022 |
| | Preparación entregable | 19/12/2022 | 24/12/2022 |
| Cierre de la Memoria y Presentación | Cierre de la Memoria | 27/12/2022 | 12/01/2023 |
| | Cierre de la Presentación | 09/01/2023 | 15/01/2023 |
| Defensa pública | Defensa Pública | 23/01/2023 | 03/02/2023 |

Siendo el diagrama de Gantt correspondiente el siguiente:

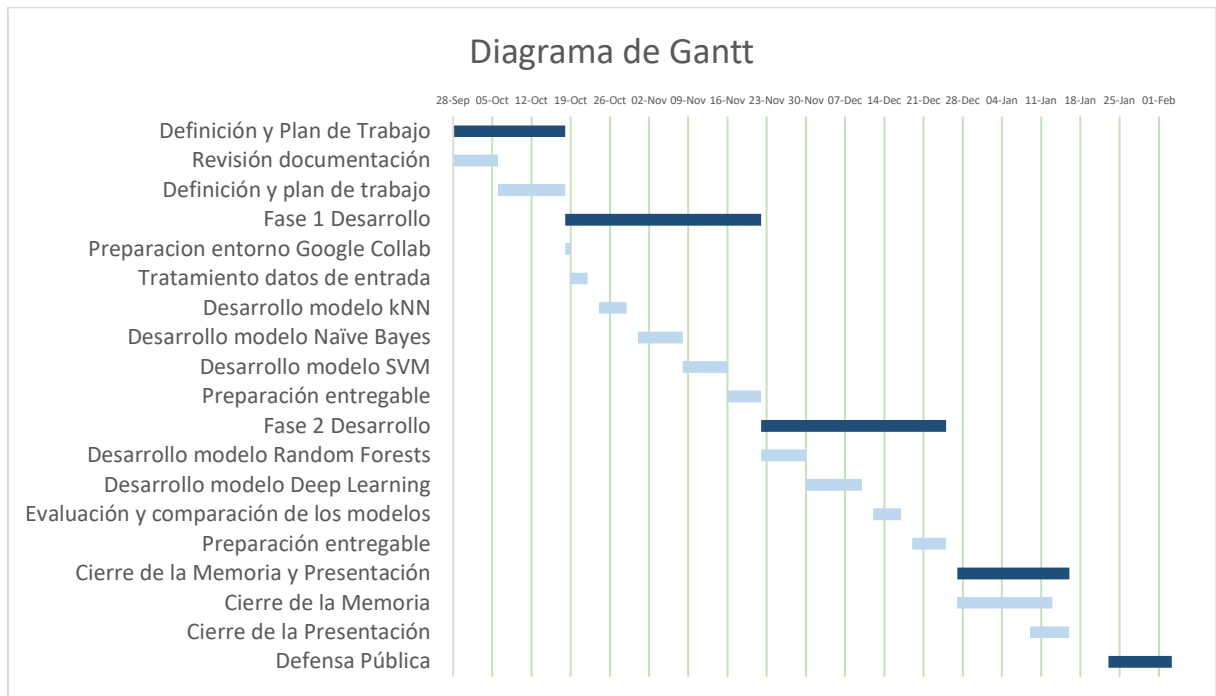


Figura 1: Diagrama de Gantt de la planificación del Proyecto

1.6. Análisis de Riesgos

Los principales riesgos a los que puede estar sujeto un trabajo de estas características son, desde mi punto de vista:

- Mala preparación de los datos iniciales
 - o Elección del *threshold* considerado óptimo para el filtraje de baja varianza.
 - o Determinar el nivel de correlación entre columnas
- Incorrecta o incompleta elección de los hiperparámetros que rigen la ejecución de los algoritmos
- Incorrecta o incompleta elección de los métodos de normalización / transformación

1.7. Breve resumen de productos obtenidos

Los productos que se esperan obtener de este trabajo son:

- La Memoria, donde se van a detallar todos los pasos del estudio realizado sobre los algoritmos propuestos. Este documento incluye la planificación del proyecto.
- Una serie de *Jupyter notebooks* en Python para ejecutar en la plataforma Google Colab. Estos *notebooks* se corresponderán al tratamiento inicial de los datos, otro para el análisis final de resultados y finalmente uno para cada uno de los algoritmos a estudiar. Se ha preferido tener esta estructura de *notebooks* en lugar de uno único por la facilidad de manejo y la independencia de ejecución de cada algoritmo.
- La presentación de la Defensa Pública del trabajo.

2. Estado del Arte

Aunque este estudio esté focalizado a nivel de datos para la determinación de moléculas activas en COX-1, se trata de un ejercicio genérico para problemas de la misma índole en el cual se ha de conseguir una clasificación lo más acurada posible a partir de un conjunto de datos dado con sus diferentes *features*. Esto es, podría extenderse a cualquier problema de búsqueda de moléculas para cualquier finalidad.

Este tipo de problemas ha sido, y es, continuamente desarrollado y, por la información que he podido recabar para la realización de este proyecto (y como se ha comentado anteriormente), son básicamente 2 lenguajes de programación, con todas sus posibles librerías asociadas, los comúnmente utilizados: *R* y *Python*.

En cuanto al uso de librerías, Scikit-Learn es la que parece ser la más ampliamente usada, aunque existen otras también muy utilizadas como *TensorFlow* y *Pytorch*, especialmente para el tema de Neural Networks. Aunque Scikit-Learn se trate de una librería *Python*, ésta puede ser utilizada de forma muy fácil desde *R*, lo que hace aún mucho más interesante su uso.

Otro tema a considerar, es la transformación o normalización de los datos. Prácticamente todo tratamiento de datos tiene que efectuar de una forma u otra esta tarea. Es importante determinar qué mecanismo utilizar para ello, aunque normalmente siempre se acaban utilizando los mismos métodos (normalización, min-max...).

Todos los proyectos, tanto comerciales como académicos que he podido observar, utilizan siempre el mismo esquema:

- Preparación de Datos. En este punto interviene el método de transformación a utilizar.
- Entrenamiento del Modelo. Es importante elegir los hiperparámetros más óptimos posibles.
- Evaluación del Modelo

Este ha sido el esquema utilizado en este proyecto y uno de los objetivos de este es intentar automatizar y ayudar a determinar qué conjunto de hiperparámetros puedan ser más eficientes así como el método de transformación más adecuado.

3. Desarrollo del Proyecto

3.1. Datos iniciales

Los datos proporcionados provienen de la base de datos ChEMBL que se corresponde con el ID CHEMBL221 (*Assay Report Card*, n.d.).

La información relativa a esta molécula se puede consultar en https://www.ebi.ac.uk/chembl/target_report_card/CHEMBL221.

Así mismo, los datos origen de este trabajo se pueden obtener desde <https://ftp.ebi.ac.uk/pub/databases/chembl/ChEMBLdb/latest/>.

Esta información ya viene con los descriptores calculados mediante RDKit (*RDKit*, n.d.), una colección de herramientas *open source* para Quimioinformática.

Se trata de un único *dataset* contenido en un archivo en formato CSV e inicialmente el archivo se ha ubicado en una carpeta de Google Drive con lo cual su lectura ha tenido que efectuarse mediante la clase `drive` del paquete `google.colab`:

```
from google.colab import drive

# Los datos originales se obtienen de Google Drive. Se establece
una conexión
drive.mount('/content/drive')

# Lectura del fichero
with open('/content/drive/My
Drive/TFM/CHEMBL221_COX1Prostaglandin_Desc.csv', 'r') as f:
    original_data = pd.read_csv(f)

# Cierre de la conexión con Google Drive
drive.flush_and_unmount()
```

El *dataset* contiene 1702 filas y 128 columnas.

3.2. Preparación de los datos

Para poder utilizar este *dataset* en los modelos de *Machine Learning* éste debe ser previamente procesado.

Los pasos de este proceso del *dataset* son los siguientes:

- Eliminación de columnas no necesarias para este estudio

Las 7 primeras columnas no proporcionan ninguna información necesaria para aplicar en los modelos, con lo que se eliminan de la siguiente forma:

```
# Eliminación de las 7 primeras columnas que no son de
utilidad para nuestro propósito
work_data = original_data.drop(original_data.iloc[:, 0:7],
axis = 1)
```

En este punto el *dataset* contiene 121 columnas.

- Separación de la columna con los resultados (*Activity*)

La columna donde se recogen los resultados para cada fila, llamada *Activity*, se tiene que eliminar de los datos a utilizar en los modelos y se tiene que almacenar en una variable diferente donde se recojan todos sus valores:

```
activity = work_data['Activity']
work_data = work_data.drop('Activity', axis = 1)
```

En este punto el *dataset* contiene 120 columnas.

- Eliminación de columnas con varianza baja

Con el fin de eliminar las columnas con varianza baja el primer paso es obtener la varianza de cada columna. A partir de esta información se decidirá cuál es el límite para considerar una varianza baja en este *dataset*. Para ello se utilizará la clase `VarianceThreshold` del paquete `sklearn.feature_selection` (*Sklearn.Feature_selection.VarianceThreshold* — *Scikit-Learn 1.2.0 Documentation*, n.d.). Esta clase permite eliminar todas las columnas con una varianza por debajo de un límite definido. Por defecto este límite es cero, pero en nuestro caso se establece primero el límite a aplicar. Para ello, primero se calculan

las diferentes varianzas para las *features* disponibles, se ordenan y se determina de forma manual cuál puede ser este límite.

```
from sklearn.feature_selection import VarianceThreshold
lvf = VarianceThreshold()

lvf.fit(work_data) # Aplicamos a nuestro conjunto potencial
de datos
lvf.variances_.sort() # Ordenamos las varianzas de menor a
mayor
print(lvf.variances_) # Mostramos el resultado
```

El resultado mostrado es el siguiente:

```
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00
0.00000000e+00 0.00000000e+00 5.87198858e-04 5.87198858e-04
1.17370730e-03 4.09589327e-03 4.69482920e-03 8.15795615e-03
2.68520066e-02 2.83955431e-02 3.30709292e-02 4.04311096e-02
4.54997991e-02 5.44976464e-02 7.62747497e-02 1.74029379e-01
2.49753867e-01 2.66760195e-01 2.99073047e-01 3.22601391e-01
3.37569611e-01 3.55799357e-01 3.75866188e-01 4.04483355e-01
4.10928734e-01 4.12761443e-01 4.15249358e-01 5.47424168e-01
6.20880460e-01 6.25475869e-01 7.39005741e-01 7.45461895e-01
7.96231751e-01 8.78183336e-01 9.70230640e-01 9.83907783e-01
9.96158870e-01 1.04664589e+00 1.10142989e+00 1.12534400e+00
1.20533008e+00 1.23360124e+00 1.27461333e+00 1.28777494e+00
1.44622152e+00 1.58759101e+00 1.65251498e+00 1.68387678e+00
1.72912114e+00 1.79258831e+00 2.23057307e+00 2.32970068e+00
2.34715811e+00 2.53018915e+00 3.29276299e+00 3.32679084e+00
3.59801301e+00 3.75960283e+00 4.17747697e+00 4.17747697e+00
4.29087314e+00 5.31502615e+00 5.76738364e+00 6.42811906e+00
6.94074711e+00 7.42930657e+00 7.42930657e+00 9.53153183e+00
1.07878324e+01 1.23867756e+01 1.47735532e+01 1.69918348e+01
1.73693217e+01 2.02337796e+01 2.11690334e+01 2.15503271e+01
2.26446685e+01 2.39484357e+01 2.60977229e+01 2.74891929e+01
2.89877817e+01 2.95682674e+01 3.11046684e+01 3.46238190e+01
3.75542981e+01 3.75542981e+01 3.95258442e+01 4.00306418e+01
4.10486786e+01 4.13138807e+01 4.21180950e+01 4.70866377e+01
5.08937086e+01 5.45128379e+01 5.54642139e+01 6.28725095e+01
6.62679197e+01 7.35631061e+01 7.40000000e+01 8.32751423e+01
1.07173713e+02 1.16000000e+02 1.18544386e+02 1.23353616e+02
1.38487127e+02 1.41069695e+02 1.48979170e+02 1.49679645e+02
1.55177481e+02 1.90883432e+02 2.39017100e+02 3.61451633e+02
3.86400000e+02 4.36805918e+02 8.19432599e+02 8.19981000e+02]
```

A partir de este resultado podría parecer razonable establecer un límite de 1 como consideración de varianza baja (aquellas marcadas en rojo). Así pues, utilizando otra vez

la clase `VarianceThreshold` con `threshold = 1` se puede proceder a la eliminación de aquellas columnas por debajo de este límite:

```
lvf = VarianceThreshold(threshold = 1)
lvf.fit(work_data)

# Recogemos las columnas donde su varianza está por debajo del
# threshold establecido

lvc = [column for column in work_data.columns if column not
in work_data.columns[lvf.get_support()]]

cr = [i.strip() for i in lvc] # Columnas a borrar
work_data = work_data.drop(cr, axis=1) # Eliminación de
# columnas con baja varianza (columnas constantes tienen varianza
# 0)
```

En este punto el *dataset* contiene 79 columnas.

- Eliminación de columnas que tengan alta correlación ($\geq \pm 0.8$)

El primer paso para esta tarea es determinar la correlación entre las diferentes columnas. Dado que a nivel Python el *dataset* en el que estamos trabajando es un objeto *dataframe* se utiliza el método `corr()` para obtener la matriz de correlaciones entre las columnas.

Del resultado obtenido se aplica la función `abs()` para cada elemento de la matriz, consiguiendo así sólo valores positivos:

```
cm = work_data.corr().abs() # Matriz positiva de correlación
```

De esta matriz sólo nos interesa el triángulo superior a la misma respecto a su diagonal.

La forma de obtener esto es la siguiente:

```
upper_triangle = cm.where(np.triu(np.ones(cm.shape), k
=1).astype(bool))
```

Las columnas donde la correlación sea mayor o igual a 0.8 son seleccionadas:

```
cr = [column for column in upper_triangle.columns if
any(upper_triangle[column] >= 0.8)]
```

Y son eliminadas del *dataset*:

```
work_data = work_data.drop(cr, axis=1)
```

En este punto el *dataset* contiene 46 columnas.

Habitualmente se incluiría un último paso de estandarización o de normalización, pero en este caso estas transformaciones se realizarán en el momento de ejecutar cada uno de los modelos propuestos como se verá más adelante, pues esta tarea será uno de los elementos para determinar un mejor rendimiento de cada modelo.

Una vez preparados los datos, éstos son divididos en 2 grupos: uno para *training* y otro para *testing*. Para ello se utiliza la función `train_test_split` de `sklearn.model_selection` ([Sklearn.Model_selection.Train_test_split — Scikit-Learn 1.2.0 Documentation](#), n.d.):

```
from sklearn.model_selection import train_test_split
```

Aplicamos una ratio 80%-20% para *training* y *testing* y un `random_state` fijo para poder obtener siempre el mismo resultado.

```
X_train, X_test, y_train, y_test = train_test_split(work_data,
activity, test_size=0.20, random_state = 19680925,
stratify=activity)
```

Finalmente se almacenan, para su uso posterior, tanto el *dataframe* obtenido así como los diferentes conjuntos de training y testing calculados utilizando, una vez más, la clase `drive` de `google.colab` ([Colabtools/Drive.Py at Master · Googlecolab/Colabtools · GitHub](#), n.d.).

```
# Escritura de los ficheros obtenidos
# Serán almacenados en Google Drive
drive.mount('/content/drive', force_remount = True)

work_data.to_csv('/content/drive/My
Drive/TFM/normalized work_data.csv', index=False)
activity.to_csv('/content/drive/My Drive/TFM/activity.csv',
index=False)
X_train.to_csv('/content/drive/My Drive/TFM/X_train.csv', index =
False)
```

```
y_train.to_csv('/content/drive/My Drive/TFM/y_train.csv', index =
False)
X_test.to_csv('/content/drive/My Drive/TFM/X_test.csv', index =
False)
y_test.to_csv('/content/drive/My Drive/TFM/y_test.csv', index =
False)
drive.flush_and_unmount()
```

3.3. Implementación de los Modelos

Después de una revisión de diferentes posibilidades, todos los modelos propuestos para este trabajo se van a implementar usando la librería *Scikit-learn* (*Scikit-Learn: Machine Learning in Python — Scikit-Learn 1.2.0 Documentation*, n.d.), una librería de aprendizaje automático que soporta tanto modelos de aprendizaje supervisado como no supervisado. También proporciona herramientas para el ajuste de modelos, así como su evaluación y una amplia serie de herramientas y utilidades.

Los motivos de esta elección han sido:

- Proporciona herramientas simples y eficientes para el análisis predictivo de datos.
- Es de fácil acceso
- Se trata de una herramienta de código abierto que se puede utilizar comercialmente, con licencia BSD.
- La compatibilidad con otras librerías utilizadas globalmente, como *NumPy*, *SciPy* y *Matplotlib*. Esto facilita la reutilización de éstas y producir un código más extensible.
- El principal uso que se hace de ésta, tanto desde el mundo académico como el empresarial. Se ha convertido en un estándar *de facto*.
- El volumen de documentación y recursos sobre la misma y la comunidad que está por detrás de ello.

Para la ejecución de los modelos se va a combinar el uso de las clases `sklearn.pipeline.Pipeline` y `sklearn.model_selection.GridSearchCV`.

`Pipeline` permite combinar la aplicación de diferentes métodos de transformación con la ejecución final de un estimador o Modelo a aplicar (*Sklearn.Pipeline.Pipeline — Scikit-Learn 1.2.0 Documentation*, n.d.).

En el caso de este trabajo no se considera como una forma de combinar diferentes métodos de transformación a la vez para un estimador dado, si no como una herramienta para facilitar la automatización de la combinación transformador-estimador.

Por su parte `GridSearchCV` permite la búsqueda sistemática de una configuración óptima de hiperparámetros para un Modelo a partir de una lista dada. El rendimiento de un Modelo depende significativamente de los hiperparámetros elegidos para el mismo, con lo que esta herramienta simplifica y facilita de forma sensible la elección óptima de los mismos (An Introduction to GridSearchCV | What Is Grid Search | Great Learning, n.d.).

Esto permite considerar un conjunto de hiperparámetros u otro de forma automatizada. Los métodos y propiedades más interesantes que proporciona esta clase son (`Sklearn.Model_selection.GridSearchCV` — *Scikit-Learn 1.2.0 Documentation*, n.d.):

- `best_params_`. Propiedad que devuelve el conjunto de hiperparámetros que proporciona mejores resultados.
- `best_score_`. Propiedad que devuelve la puntuación media con validación cruzada.
- `fit(X, y)`. Método para ejecutar el modelo (con los datos de *training*).
- `predict(X)`. Método para la obtención la predicción sobre el conjunto *X* de *testing* basado en el mejor conjunto de hiperparámetros.
- `predict_proba(X)`. Método para la obtención la predicción en términos de probabilidades sobre el conjunto *X* de *testing* basado en el mejor conjunto de hiperparámetros.

Para realizar esta búsqueda se prueban todas las combinaciones posibles de los parámetros proporcionados utilizando el método de validación cruzada (*cross validation*, de ahí el sufijo CV del nombre de la clase).

La validación cruzada es un método estadístico utilizado habitualmente para estimar el rendimiento o precisión de los modelos de *Machine Learning*. Se utiliza para proteger contra el *overfitting* en un modelo predictivo, especialmente cuando el conjunto de datos a tratar es pequeño. Este mecanismo consiste en realizar un número fijo de *folders* (particiones) de los datos de entrada, ejecutando el análisis para cada una de las particiones y devolviendo como resultado final el promedio de la estimación general de los errores (*What Is Cross Validation and Its Types in Machine Learning?* Great Learning, n.d.).

Para cada uno de los modelos propuestos se van a definir 4 *Pipelines* diferentes, donde en cada uno de ellos se va a ejecutar un método diferente de transformación con el fin de constatar que la elección de dichos métodos puede influir, dependiendo de la naturaleza de los datos y las características de cada Modelo, influye en los resultados obtenidos.

Estos métodos de transformación son los siguientes:

- Min Max

Todos los valores son convertidos al rango [0, 1]. La fórmula para esta transformación es:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

La clase a utilizar para este método de transformación es `sklearn.preprocessing.MinMaxScaler` (*Sklearn.Preprocessing.MinMaxScaler* — *Scikit-Learn 1.2.0 Documentation*, n.d.).

- Standard

Se trata de una transformación donde se estandarizan los datos mediante la supresión de la media y éstos son escalados de tal forma que su varianza toma por valor 1.

La fórmula de esta transformación es

$$z = \frac{(x - u)}{s}$$

donde x es el valor a normalizar, u es la media del conjunto de valores y s es la desviación estándar.

En este caso la clase a utilizar es `sklearn.preprocessing.StandardScaler` (*Sklearn.Preprocessing.StandardScaler* — *Scikit-Learn 1.2.0 Documentation*, n.d.).

- Normalizer

En este caso se aplica la normalización L2. Se transforman los datos de forma que si el resultado es elevado al cuadrado y sumado se obtiene como resultado 1.

La clase utilizada en este caso es `sklearn.preprocessing.Normalizer` (*Sklearn.Preprocessing.Normalizer — Scikit-Learn 1.2.0 Documentation*, n.d.).

- Robust (`sklearn.preprocessing.RobustScaler`)

Es un escalado que ofrece más fiabilidad respecto a los *outliers*. Elimina la mediana y escala los datos de acuerdo con un rango de cuartiles, en concreto el llamado intercuartiles que es el rango de valores entre el primer y tercer cuartil.

Para este método de transformación se utilizará la clase `sklearn.preprocessing.RobustScaler` (*Sklearn.Preprocessing.RobustScaler — Scikit-Learn 1.2.0 Documentation*, n.d.).

De la ejecución de cada Modelo se va a recoger una serie de métricas que van a permitir determinar cuáles son los hiperparámetros más adecuados dependiendo del método de transformación para la comparación final entre los diferentes Modelos.

Estas métricas son:

- *Accuracy* (exactitud)

Es la ratio del número predicciones correctas respecto al número total de predicciones. Se trata de cuán cerca está una medida del valor aceptado.

Las predicciones correctas es la suma de *True Positives* y *True Negatives*.

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- *Precision*

Se trata de la proporción de los *True Positives* sobre las predicciones correctas. Indica la proximidad de cada elemento del conjunto de datos respecto a los otros.

$$precision = \frac{TP}{TP + TN}$$

Hay que tener en cuenta la independencia de la *Precision* respecto a la *Accuracy*. Se puede ser muy preciso, pero no muy exacto y viceversa. Lo idóneo sería tener unos valores en ambos casos lo más alto posible.

- *Sensitivity*

Es la ratio de *True Positives*. Permite comprobar cuantas instancias positivas se han identificado correctamente. Un valor alto de esta métrica indica un valor bajo de *False Negatives*, lo que significa que se ha obviado un valor bajo de *True Positives*.

$$sensitivity = \frac{TP}{TP + FN}$$

- *Specifity*

Es la métrica equivalente a *Sensitivity* en lo que respecta a las predicciones negativas. Se trata de la ratio de *True Negatives*. Permite comprobar cuantas instancias negativas se han identificado correctamente.

$$specifity = \frac{TN}{TN + FP}$$

- *Kappa*

Coeficiente de *Cohen Kappa* que indica la utilidad del resultado. Se obtiene con la función `cohen_kappa_score` del paquete `sklearn.metrics`. Su valor está dentro del rango [0, 1]. Aunque no hay una forma estandarizada de interpretar su valor, Landis y Koch (*The Measurement of Observer Agreement for Categorical Data on JSTOR*, n.d.)establecieron en 1977 una forma de hacerlo:

- 0 – 0.20: Baja utilidad
- 0.21 – 0.40: Justo
- 0.41 – 0.60: Moderado
- 0.61 – 0.80: Sustancial
- 0.80 – 1: Perfecto

- *ROC curve / AUC*

Para cada resultado a presentar se va a mostrar un gráfico con la *ROC curve* (Receiver Operating Characteristic curve). Esta curva es descrita mediante la ratio de *True Positives* (en el eje Y) y la de *True Negatives* (eje X).

La *Area Under the Curve (AUC)* es el área que se obtiene entre el eje de coordenadas y la *ROC curve*. Su valor está en el intervalo [0, 1] y cuanto más alto sea este más fiable será el resultado.

A grandes rasgos para cada Modelo se van a ejecutar las siguientes tareas:

- Obtención y adecuación de los datos de entrada (conjuntos de *training* y *testing*) a partir de los ficheros obtenidos en el punto *Preparación de los Datos*.
- Definición del conjunto de hiperparámetros para cada Modelo
- Para cada método de transformación:
 - o Creación del *Pipeline* con este método juntamente con el Clasificador del Modelo.
 - o Definición del objeto *GridSearchCV* combinando el conjunto de hiperparámetros con el *Pipeline* creado en el punto anterior.
 - o Recogida de las métricas
 - o Creación del informe para cada Modelo a incluir en este documento.

La idea inicial de este proyecto es desarrollar un *notebook* diferente para cada uno de los algoritmos a presentar.

En una primera iteración todos los pasos anteriormente enumerados habían sido traducidos a código Python para cada uno de los algoritmos.

Esto es, evidentemente, poco práctico y elegante desde el punto de vista del mantenimiento de futuros cambios y/o el desarrollo de nuevos algoritmos.

Para ello, se ha desarrollado una librería en Python que tiene que ser importada por cada *notebook* con los métodos comunes necesarios para el desarrollo de un algoritmo. El fichero que contiene esta librería se llama `tfm_lib.py`.

Esto permite, además, poder reutilizar esta librería en un entorno diferente a *Google Collaboration*.

Esta librería contiene las siguientes funciones:

- `load_data(basePath = '/content/drive/My Drive/TFM')`

Carga los ficheros correspondientes a los conjuntos de datos de training y testing y los devuelve en forma de matrices (valores *X*) y vectores (valores *Y*).

Recibe un parámetro opcional que especifica la carpeta base donde se pueden encontrar los ficheros datos en formato CSV.

Básicamente monta el Google Drive, lee los 4 ficheros mediante el método `read_csv` de la librería *pandas* (*Pandas.Read_csv — Pandas 1.5.2 Documentation*, n.d.).

- `calculate_models(id, params, X_train, y_train, X_test, y_test, classifier, transformers = default_transformers, scoring_method = default_scoring_method)`

Calcula los diferentes modelos para cada uno de los *Transformers* que se reciben por parámetro.

Los parámetros son:

- o *id*: Identificador del algoritmo. Es importante para el siguiente parámetro porque ha de ser el sufijo de cada uno de los *settings* a aplicar en la ejecución del modelo.
- o *params*: Diccionario con los diferentes parámetros a aplicar en el modelo. Cada una de las *keys* del diccionario debe de estar prefijada con el *id* mencionado arriba juntamente con 2 *underscores* (`__`). Por ejemplo, *id* tiene valor *knn* y se quiere especificar el *setting* *n_neighbors*, entonces la key para el diccionario será *knn__n_neighbors*.
- o *X_train*, *y_train*, *X_test*, *y_test*. Conjuntos de datos de *training* y *testing*.
- o *classifier*. Instancia de clase del algoritmo a evaluar.

- `transformers`. Lista de Transformes a evaluar. Por defecto son los descritos anteriormente (`MinMaxScaler`, `StandardScaler`, `Normalizer`, `RobustScaler`).
- `scoring_method`. La clase `GridSearchCV` necesita establecer un método para poder hacer las comparaciones entre los diferentes conjuntos de parámetros. Se asume por defecto *accuracy*.

Esta función, para cada uno de los *Transformers*, crea un `Pipeline` con el *Transformer* en curso y el clasificador y una instancia de `GridSearchCV` con éste juntamente con los parámetros del clasificador. A continuación, llama al método `fit` con los datos de *training* y recoge todas las métricas y datos necesarios para devolver y que puedan ser utilizados más adelante.

El resultado se devuelve en un diccionario donde se encuentran todas las métricas calculadas.

- `show_graphs(outcome)`
Dibuja las *ROC Curves* con los resultados para cada uno de los Transformers obtenidos a partir de la ejecución de la función descrita anteriormente.
- `execute_model(id, classifier, params, transformers = default_transformers, scoring_method = default_scoring_method, basePath = '/content/drive/My Drive/TFM')`

Función de utilidad que combina `load_data` y `calculate_models` para simplificar. Devuelve la misma información que `calculate_models`.

Así pues, el notebook de la implementación para un algoritmo dado sería:

- Importación de las librerías necesarias
El conjunto mínimo es:

```
import sys
from google.colab import drive
```

Aparte se debe incluir como mínimo la referencia al algoritmo a ejecutar. Por ejemplo, para el algoritmo *kNN* se tendría que incluir:

```
from sklearn.neighbors import KNeighborsClassifier
```

- **Importación de la librería `tfm_lib`**

```
drive.mount('/content/drive', force_remount=True)
```

```
sys.path.append('/content/drive/My Drive/TFM')
```

```
import tfm_lib as tfm
```

```
drive.flush_and_unmount()
```

- **Definición del conjunto de parámetros**

```
params = {...}
```

- **Ejecución de los modelos**

```
classifier = XXX()
```

```
outcome = tfm.execute_model('xxx', classifier, params)
```

- **Mostrar gráficos de las *ROC curves***

```
tfm.show_graphs(outcome)
```

- **Mostrar el resumen de métricas**

```
outcome["report"].style
```

3.4. *k*-Nearest Neighbors

k-NN es un algoritmo de aprendizaje supervisado en el que a partir de un conjunto de datos de entrenamiento su objetivo es el de clasificar correctamente un conjunto de datos de entrada.

Este algoritmo clasifica cada elemento nuevo del conjunto de entrada en la clase que le corresponda, estableciendo las distancias respecto al número *k* de vecinos más cercanos.

En función de las distancias, se clasifica el elemento. Dependiendo del número de los más cercanos se clasifica el elemento en curso. Es por ello por lo que es importante definir el número de vecinos (*k*) como número impar.

Consideremos el siguiente ejemplo con *k*=3

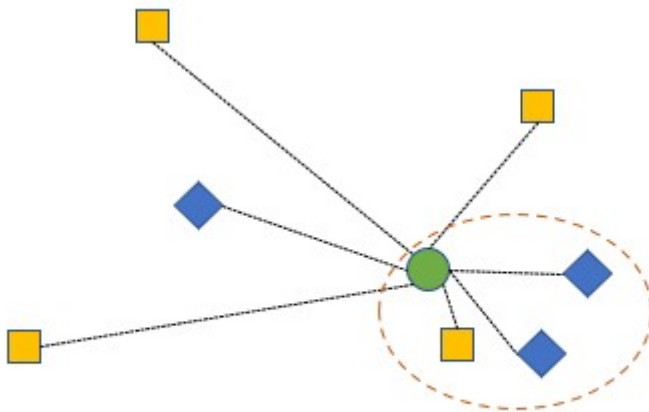


Figura 2 Ejemplo *k*NN

El elemento para clasificar (círculo verde), tiene como 3 vecinos más cercanos los delimitados por la elipse (2 rombos azules y un cuadrado amarillo). Aunque el cuadrado amarillo sea más cercano, hay 2 rombos azules con lo que el elemento se acabará clasificando como rombo azul.

Se trata de un caso de *lazy learning algorithm* donde simplemente se almacenan los datos de entrenamiento ya clasificados.

| Ventajas | Inconvenientes |
|--|--|
| Simple y efectivo | No produce un modelo, lo que limita la capacidad |
| No hace suposiciones sobre la distribución de datos subyacente | Requiere de una selección apropiada de k . |
| Fase de entrenamiento rápida | Fase de clasificación lenta |

El hiperparámetro más relevante para este algoritmo es el valor de k ($n_neighbors$), por lo que, para la ejecución de este algoritmo, se considera el siguiente conjunto de hiperparámetros:

```
params = [{'knn__n_neighbors': [3, 5, 7, 9, 11, 13]}]
```

Esto significa que se va a probar con valores para k ($n_neighbors$): 3, 5, 7, 9, 11, 13.

El resumen de mejores métricas obtenidas por *transformer* es:

| | classifier | transformer | best_parameter_set | kappa | auc | accuracy | precision | sensitivity | specificity |
|---|----------------------|----------------|--------------------|----------|----------|----------|-----------|-------------|-------------|
| 0 | KNeighborsClassifier | MinMaxScaler | 1 | 0.345106 | 0.773686 | 0.692082 | 0.617886 | 0.567164 | 0.772947 |
| 1 | KNeighborsClassifier | StandardScaler | 0 | 0.373401 | 0.766007 | 0.703812 | 0.629921 | 0.597015 | 0.772947 |
| 2 | KNeighborsClassifier | Normalizer | 2 | 0.400097 | 0.794704 | 0.724340 | 0.688679 | 0.544776 | 0.840580 |
| 3 | KNeighborsClassifier | RobustScaler | 1 | 0.346984 | 0.777165 | 0.695015 | 0.627119 | 0.552239 | 0.787440 |

Figura 3. Métricas para *kNN*

Se puede observar que, tal y como apuntan todas las métricas, el *transformer* Normalizer es el que proporciona mejores resultados.

La clase utilizada para este algoritmo es `sklearn.neighbors.KNeighborsClassifier` (*Sklearn.Neighbors.KNeighborsClassifier — Scikit-Learn 1.2.0 Documentation*, n.d.).

El `best_parameter_set` se corresponde en este caso como un valor de 7 para `n_neighbors` (k).

Los valores obtenidos no son muy buenos precisamente. Teniendo en cuenta los valores de Kappa para los 4 transformadores, el mejor de ellos está en el límite inferior de lo considerado como *justo* (0.40).

En las gráficas de ROC curve se puede apreciar también que el *transformer* Normalizer es el que proporciona mejores resultados, aunque en general como se comenta más arriba, éstos no son realmente buenos:

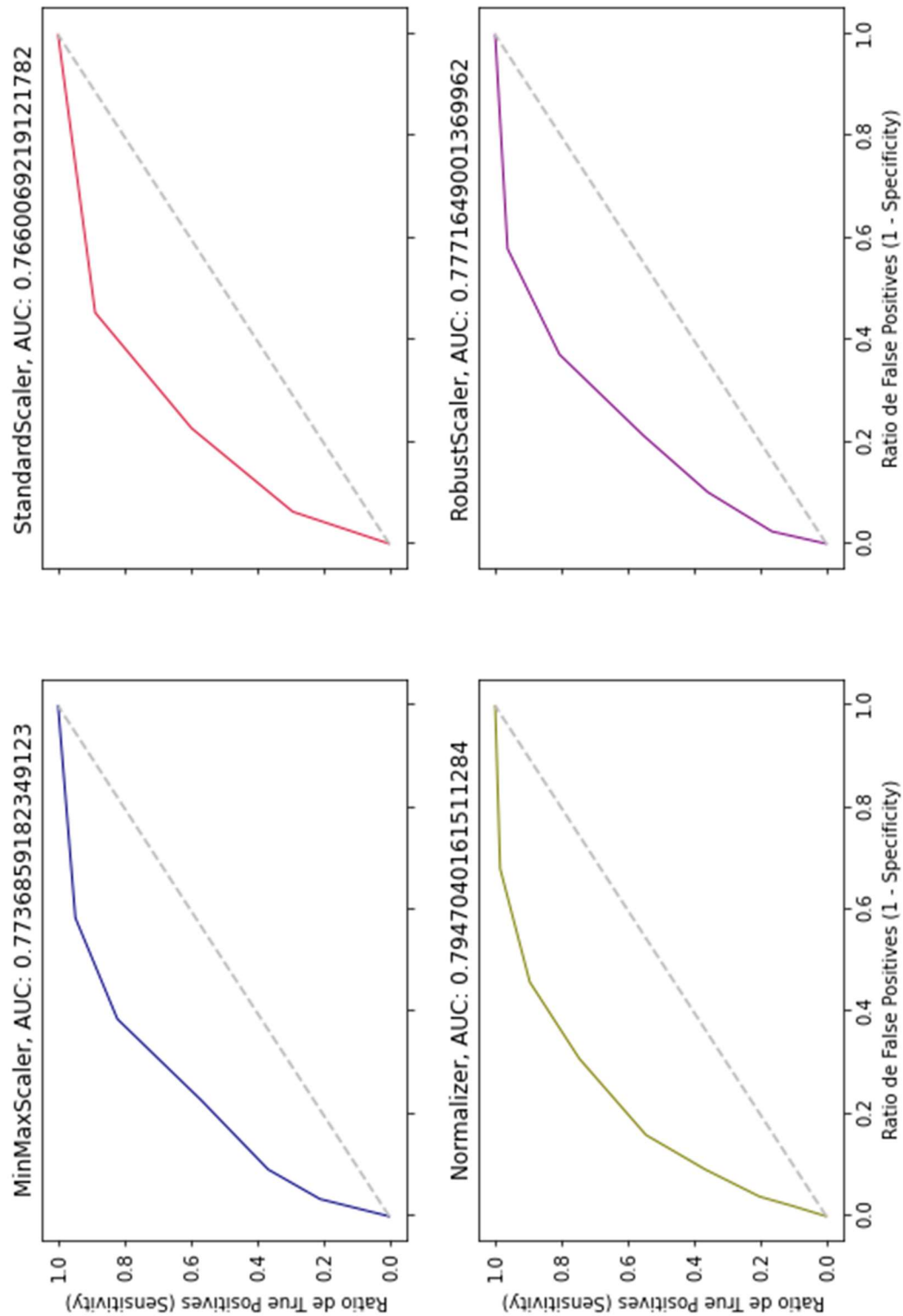


Figura 4 ROC Curves para *kNN*

3.5. Naïve-Bayes

El algoritmo *Naïve-Bayes* es un método simple de la aplicación del teorema de Bayes en problemas de clasificación. No es el único método de *Machine Learning* que aplica los métodos bayesianos, pero es el más utilizado.

Está basado en la suposición “ingenua” (*naïve*) sobre la independencia condicional entre cada par de columnas (*features*) dado el valor de la variable de clase.

A pesar de este planteamiento tan simplificado, este algoritmo ha demostrado tener una más que aceptable efectividad en aplicaciones en el mundo real, como clasificación de documentos o detección de *spam* en e-mails. Esto ya puede anticipar que probablemente este algoritmo no sea adecuado al problema de clasificación binaria que se trata en este trabajo (1.9. *Naive Bayes* — *Scikit-Learn 1.2.0 Documentation*, n.d.).

| Ventajas | Inconvenientes |
|---|---|
| Simple, rápido y efectivo | La asunción sobre la que se basa el método no es necesariamente cierta, ya que asume que todas las <i>features</i> tienen la misma importancia e independencia. |
| Su funcionamiento es aceptable con <i>missing data</i> y ruido de fondo. | No funciona de forma óptima cuando trabaja con conjuntos de datos con muchas variables numéricas. |
| No requiere de un conjunto de <i>training</i> muy grande, aunque acepta igualmente grandes conjuntos de muestras. | La fiabilidad de las probabilidades estimadas acostumbra a ser bastante inferior a las clases predichas. |

La clase utilizada en este caso es `sklearn.naive_bayes.GaussianNB` (*Sklearn.Naive_bayes.GaussianNB* — *Scikit-Learn 1.2.0 Documentation*, n.d.).

Para la ejecución de este algoritmo, el conjunto de parámetros es el siguiente:

```
params = {'nb__var_smoothing': np.logspace(0, -9, num=100)}
```

Se trata de la porción de la varianza más grande de todas las *features* que se agrega a las varianzas para la estabilidad del cálculo, este es el principal hiperparámetro que contempla este algoritmo. En este caso, se establece un vector de 100 posiciones cuyo contenido son números espaciados uniformemente en una escala logarítmica (entre -9 y 0).

El resumen de mejores métricas obtenidas por *transformer* es:

| | classifier | transformer | best_parameter_set | kappa | auc | accuracy | precision | sensitivity | specificity |
|---|------------|----------------|--------------------|----------|----------|----------|-----------|-------------|-------------|
| 0 | GaussianNB | MinMaxScaler | 34 | 0.184157 | 0.675211 | 0.571848 | 0.471429 | 0.738806 | 0.463768 |
| 1 | GaussianNB | StandardScaler | 28 | 0.184157 | 0.675175 | 0.571848 | 0.471429 | 0.738806 | 0.463768 |
| 2 | GaussianNB | Normalizer | 12 | 0.046886 | 0.596150 | 0.583578 | 0.442857 | 0.231343 | 0.811594 |
| 3 | GaussianNB | RobustScaler | 2 | 0.090302 | 0.640962 | 0.609971 | 0.508772 | 0.216418 | 0.864734 |

Figura 5. Métricas para *Naïve-Bayes*

Tal y como se había comentado anteriormente los resultados obtenidos con este modelo demuestran que no es el más idóneo para el tipo de problema que trata este trabajo. Incluso los mejores valores de *kappa* obtenidos pertenecen a la clasificación de *baja utilidad*.

Se puede observar que, tal y como apuntan todas las métricas, el *transformer* *MinMaxScaler* es el que proporciona mejores resultados, seguido muy de cerca del *StandardScaler* (las diferencias son nimias).

El *best_parameter_set* se corresponde a

```
{
    'nb__var_smoothing': 0.0008111308307896872
}
```

para el caso del *MinMaxScaler* y

```
{
    'nb__var_smoothing': 0.002848035868435802
}
```

para *StandardScaler*.

En las gráficas de ROC curve se puede apreciar también que los *transformers* *MinMaxScaler* y *StandardScaler* son los que proporcionan mejores resultados:

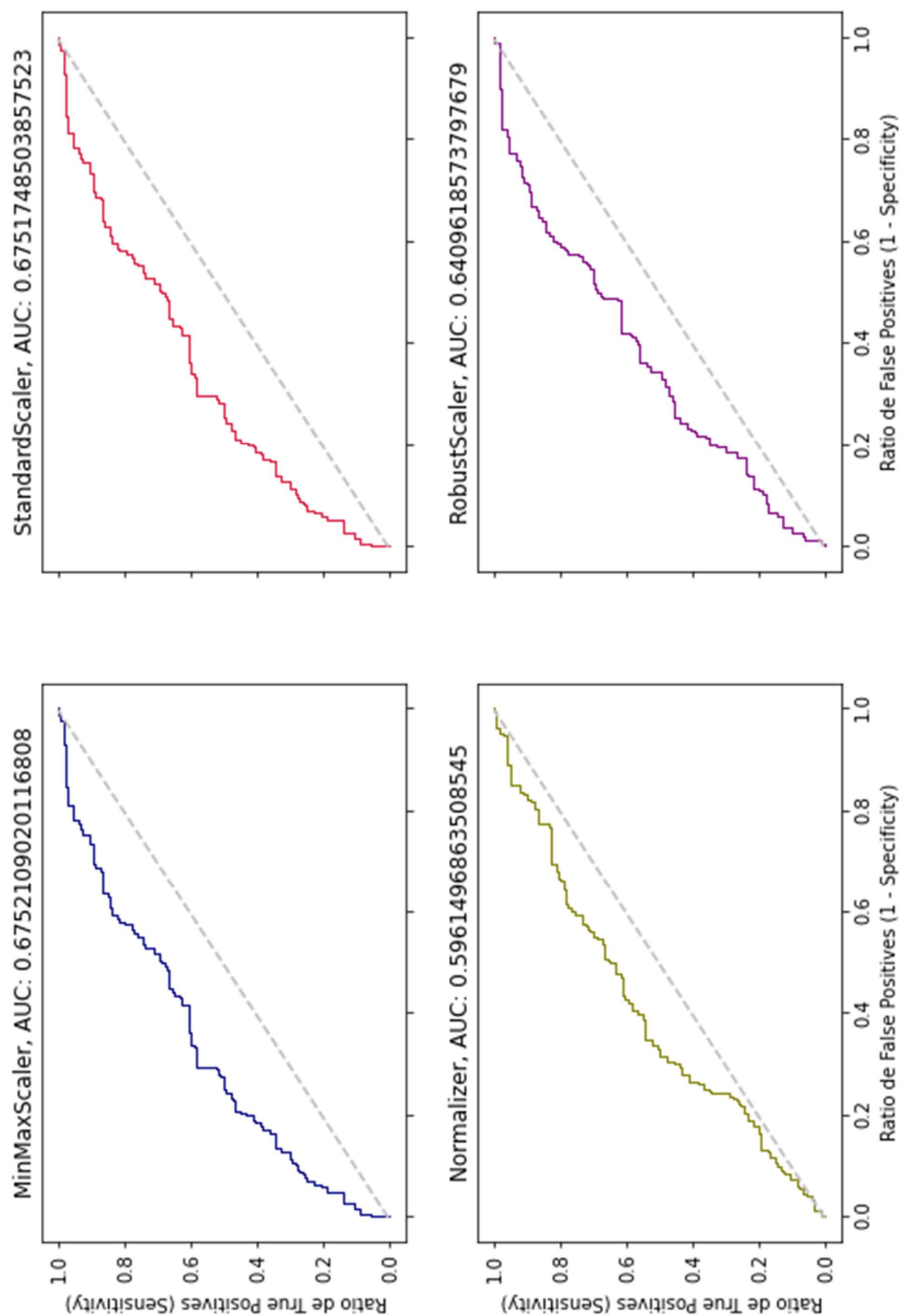


Figura 6 ROC Curves para *Naïve-Bayes*

3.6. Support Vector Machines

Dentro de los algoritmos de aprendizaje supervisado se encuentran los *Support Vector Machines*. Estos algoritmos están estrechamente relacionados con problemas de clasificación y regresión, aunque inicialmente se concibió como un método de clasificación binaria. También pueden ser usados para la detección de *outliers*.

Los datos son distribuidos de forma homogénea separados con el máximo margen posible entre ellos mediante un hiperplano de separación establecido como un vector entre 2 valores de 2 clases diferentes situados de forma más cercana al llamado vector de soporte.

En caso de que la separación entre datos no sea lineal, se tiene que recurrir a funciones de similitud parametrizadas en función de un valor (hiperparámetro C) para minimizar el coste. Este hiperparámetro permite regular el número y severidad de las violaciones del margen y, por tanto, del hiperplano que se aceptan en el proceso de ajuste. Si C tuviera un valor infinito, no sería permitida ninguna violación del margen (obteniendo así el *Maximal Margin Classifier*, sólo posible en el caso de que las clases sean totalmente separables), mientras que cuando el valor de C se va aproximando más a cero se penalizan menos los errores lo que puede llevar a una peor clasificación de las observaciones. Esto da lugar a tener que encontrar el valor óptimo de este parámetro en cada caso (*Máquinas de Vector Soporte (Support Vector Machines, SVMs)*, n.d.).

Éstos son los principales inconvenientes y ventajas:

| Ventajas | Inconvenientes |
|---|---|
| Eficaz en un espacio de altas dimensiones | Si el número de características es bastante superior al de muestras, es crucial evitar el over-fitting al elegir las funciones de <i>Kernel</i> . Implica un entrenamiento lento. |
| Sigue siendo efectivo en el caso de que el número de dimensiones sea mayor que el de muestras | No proporcionan directamente estimaciones de probabilidad, estas se calculan mediante una costosa validación cruzada (<i>five-fold cross validation</i>). |

| | |
|---|--|
| Utiliza un subconjunto de puntos de entrenamiento en la función de decisión (los vectores de soporte), por lo que también es eficiente en memoria. | Requiere especificar función de <i>Kernel</i> y ajustar el valor de C mediante prueba y error. |
| Versátil: se pueden especificar diferentes funciones del Kernel para la función de decisión. Se proporcionan kernels comunes, pero también es posible especificar kernels personalizados. | |
| Muy eficaz en predicción y clasificación. | |
| Más fácil de usar que los algoritmos de Redes Neuronales. | |

La clase utilizada para implementar este modelo ha sido `sklearn.svm.SVC` (*Sklearn.Svm.SVC — Scikit-Learn 1.2.0 Documentation*, n.d.).

Para la ejecución de este algoritmo, el conjunto de parámetros es el siguiente:

```
params = {'svm__C': [0.1, 1, 10],
          'svm__gamma': [1, 0.1, 0.01],
          'svm__kernel': ['rbf', 'linear', 'sigmoid']}
```

Donde:

- *C*: Valor de la función de similitud (regularización)
- *gamma*: Coeficiente para el *Kernel* elegido (sólo se aplicará para *rbf* y *sigmoid*).
- *kernel*: Especifica el tipo de *Kernel* a utilizar en el algoritmo.

El resumen de mejores métricas obtenidas por *transformer* es:

| | classifier | transformer | best_parameter_set | kappa | auc | accuracy | precision | sensitivity | specificity |
|---|------------|----------------|--------------------|----------|----------|----------|-----------|-------------|-------------|
| 0 | SVC | MinMaxScaler | 10 | 0.421496 | 0.789693 | 0.727273 | 0.664000 | 0.619403 | 0.797101 |
| 1 | SVC | StandardScaler | 12 | 0.433367 | 0.787908 | 0.736070 | 0.689655 | 0.597015 | 0.826087 |
| 2 | SVC | Normalizer | 20 | 0.422332 | 0.791027 | 0.730205 | 0.677966 | 0.597015 | 0.816425 |
| 3 | SVC | RobustScaler | 17 | 0.429378 | 0.791549 | 0.733138 | 0.680672 | 0.604478 | 0.816425 |

Figura 7. Métricas para *SVM*

Se puede observar que, tal y como apuntan todas las métricas, el *transformer* `StandardScaler` es el que proporciona mejores resultados, aunque el resto basados en los valores de *kappa* y *auc* están muy cercanos.

Sin ser unos resultados excepcionales, basándonos en el valor de *kappa*, éstos pueden ser considerados como *moderados*, bastante mejores que los obtenidos con *Naïve-Bayes* y un poco superiores a los de *k-NN*.

El `best_parameter_set` se corresponde en este caso a

```
{
    'svm__C': 10,
    'svm__gamma': 0.01,
    'svm__kernel': 'rbf'
}
```

En las gráficas de ROC curve se puede apreciar también que el *transformer* `StandardScaler` es el que proporciona mejores resultados:

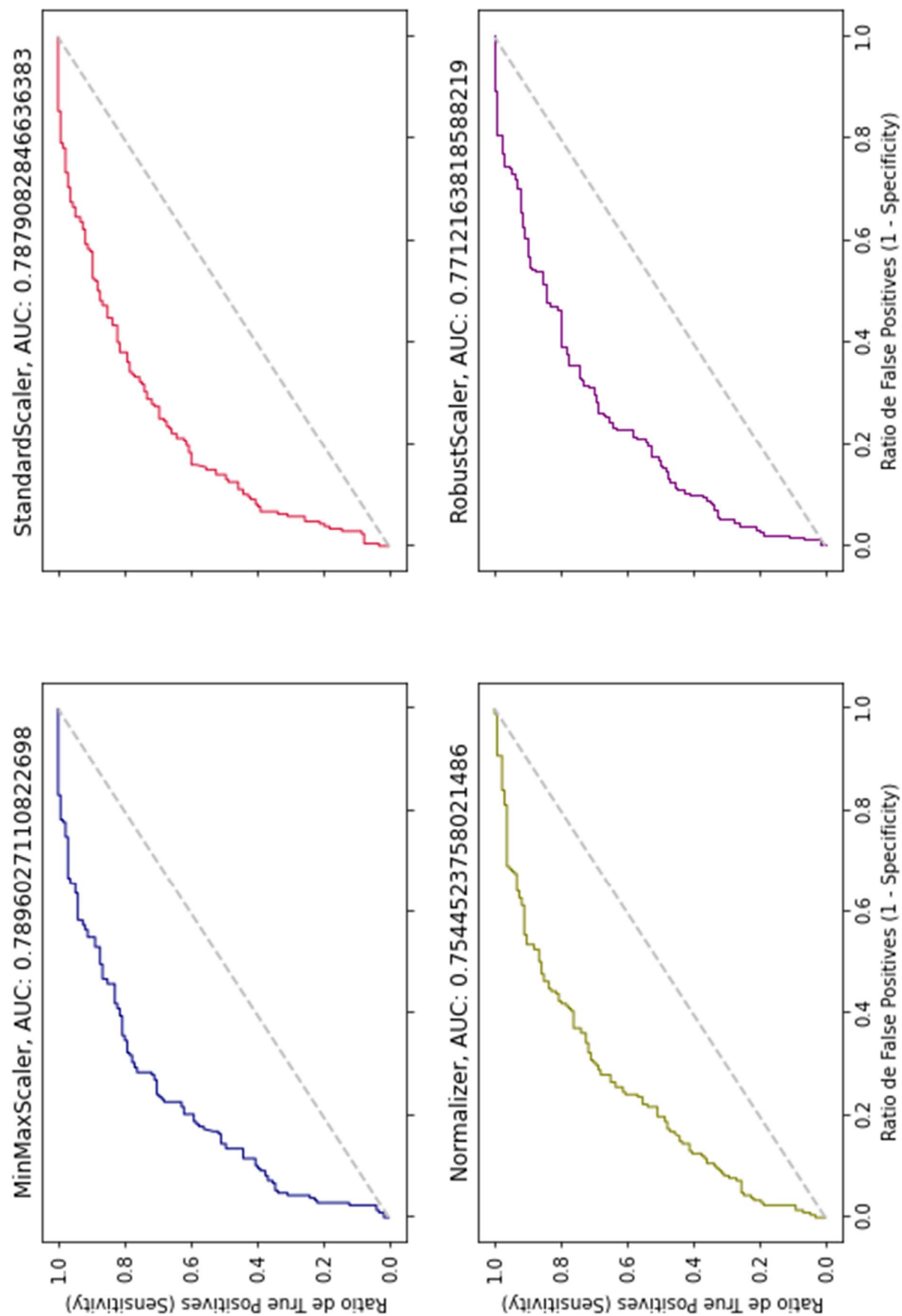


Figura 8 ROC Curves *Support Vector Machine*

3.7. Random Forest

Los *Random Forest* es un método de agregación de diferentes *árboles de decisión*. Los *árboles de decisión* son un tipo de algoritmo de aprendizaje supervisado y son especialmente eficaces en problemas de clasificación, lo que son un buen candidato para el tipo de problema que intenta resolver este trabajo. La idea es formar un árbol en función de las *features* disponibles, siendo cada nodo interno del árbol una evaluación de cada una de las *features* donde el resultado de la decisión es transmitido a la evaluación de la siguiente *feature* o a un nodo final de resultado. Un ejemplo muy simple sería el siguiente: un árbol de decisión para determinar si una persona ha de vacunarse de COVID o no en función de una serie de *features*. Éstas podrían ser *Grupo de Riesgo* (una persona mayor de 60 años y/o con un tipo de enfermedad específico), *Personal Sanitario* (si el individuo forma parte del personal de centros médicos y relacionados) y si *Estado de Infección* de COVID (si ha sufrido la enfermedad en los últimos 6 meses). Este árbol de decisión sería:

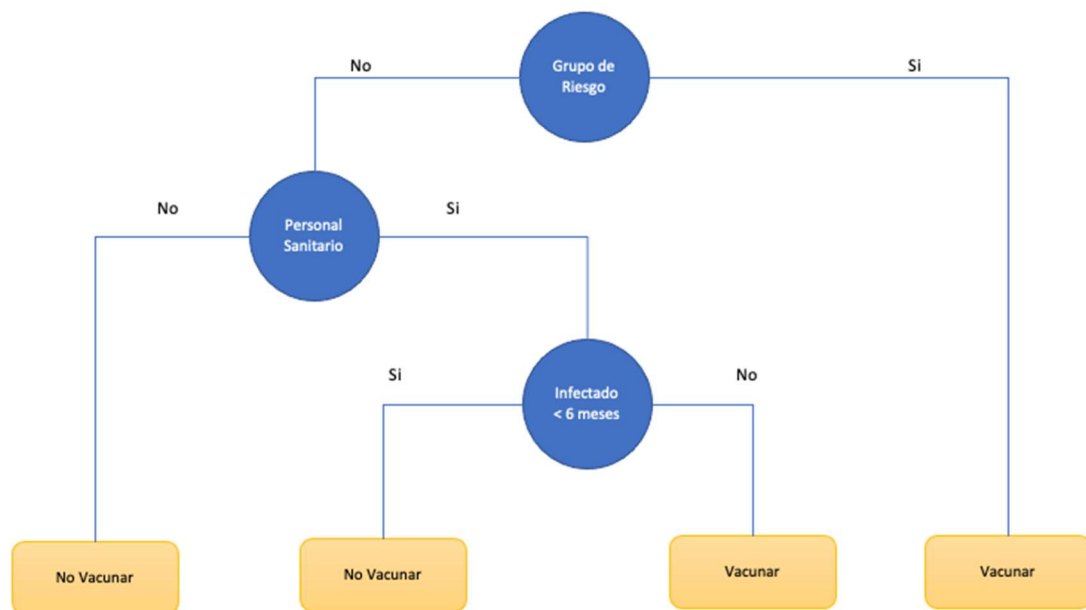


Figura 9 Ejemplo de árbol de decisión

En el caso de *Random Forest* la agregación de los diferentes árboles de decisión se realiza mediante la técnica de *Bagging*. Con ésta se persigue disminuir la varianza de las predicciones estableciendo diferentes subconjuntos formados por combinaciones de *features* del conjunto de datos de entrada, calculando un *árbol de decisión* para cada subconjunto y estableciendo el

resultado final en función de los diferentes resultados de cada árbol calculado (*Understanding Random Forest. How the Algorithm Works and Why It Is... | by Tony Yiu | Towards Data Science*, n.d.).

Éstos son los principales inconvenientes y ventajas:

| Ventajas | Inconvenientes |
|--|---|
| Eficaz en un espacio de altas dimensiones y de muestras. | Puede llevar cierto tiempo determinar los hiperparámetros más adecuados para el modelo en función de los datos. |
| Se trata de un método que proporciona buenos resultados para la mayoría de los problemas de clasificación. | No hay mucho control con lo que hace el modelo, convirtiéndose en una especie de <i>caja negra</i> . |
| No necesita de una preparación de datos muy exhaustiva. Funciona bien con la existencia de datos irrelevantes y/o inexistentes en algunas <i>features</i> . Trabaja bien tanto con variables categóricas como continuas. | Como se ha comentado en las Ventajas es muy bueno para clasificación, pero no para problemas de regresión. |
| Más fácil de usar que los algoritmos de Redes Neuronales. | |

La clase utilizada para este algoritmo es `sklearn.ensemble.RandomForestClassifier` (*Sklearn.Ensemble.RandomForestClassifier — Scikit-Learn 1.2.0 Documentation*, n.d.).

Para la ejecución de este algoritmo, el conjunto de parámetros es el siguiente:

```
params = [{'rf__random_state': [0, None],
          'rf__bootstrap': [True, False],
          'rf__n_estimators': [10, 30, 50, 80, 100],
          'rf__max_features': ['sqrt', 'log2']}]
```

Donde:

- *random_state*: En caso de utilizar Bootstrap, especificar si siempre se utilizará el mismo *seed* para crear los diferentes subconjuntos (consiguiendo así siempre la misma configuración de subconjuntos) o la creación siempre será diferente en todos los casos.
- *bootstrap*: Aplicación de *bootstrapping* o no.
- *n_estimators*: Número de árboles de decisión en el *Forest*.
- *max_features*: Número de *features* a considerar para cada árbol. En este caso se especifica la raíz cuadrada del número de *features* del conjunto de datos de entrada o el logaritmo en base 2 del mismo.

El resumen de mejores métricas obtenidas por *transformer* es:

| | classifier | transformer | best_parameter_set | kappa | auc | accuracy | precision | sensitivity | specificity |
|---|------------------------|----------------|--------------------|----------|----------|----------|-----------|-------------|-------------|
| 0 | RandomForestClassifier | MinMaxScaler | 12 | 0.435303 | 0.819778 | 0.741935 | 0.725490 | 0.552239 | 0.864734 |
| 1 | RandomForestClassifier | StandardScaler | 12 | 0.440952 | 0.820229 | 0.744868 | 0.732673 | 0.552239 | 0.869565 |
| 2 | RandomForestClassifier | Normalizer | 26 | 0.444011 | 0.804798 | 0.744868 | 0.723810 | 0.567164 | 0.859903 |
| 3 | RandomForestClassifier | RobustScaler | 12 | 0.449649 | 0.820553 | 0.747801 | 0.730769 | 0.567164 | 0.864734 |

Figura 10. Métricas para *Random Forest*

Se puede observar que, tal y como apuntan todas las métricas, el *transformer* *RobustScaler* es el que proporciona mejores resultados, aunque seguido muy de cerca por *Normalizer* y *StandardScaler*, basado en los valores de *kappa* y *auc* están muy cercanos. El rendimiento de este modelo es muy similar al anteriormente visto (*SVM*) si tomamos *kappa* como valor comparativo de referencia.

El *best_parameter_set* se corresponde en este caso a

```
{
    'rf__bootstrap': True,
    'rf__max_features': 'log2',
    'rf__n_estimators': 30,
    'rf__random_state': 0
}
```

Esto es, utilizando *log2* como número máximo de *Features*, con un número de árboles de 30 y utilizando Bootstrap con la misma selección aleatoria de entradas.

En las gráficas de ROC curve se puede apreciar también que el *transformer* *RobustScaler* es el que proporciona mejores resultados:

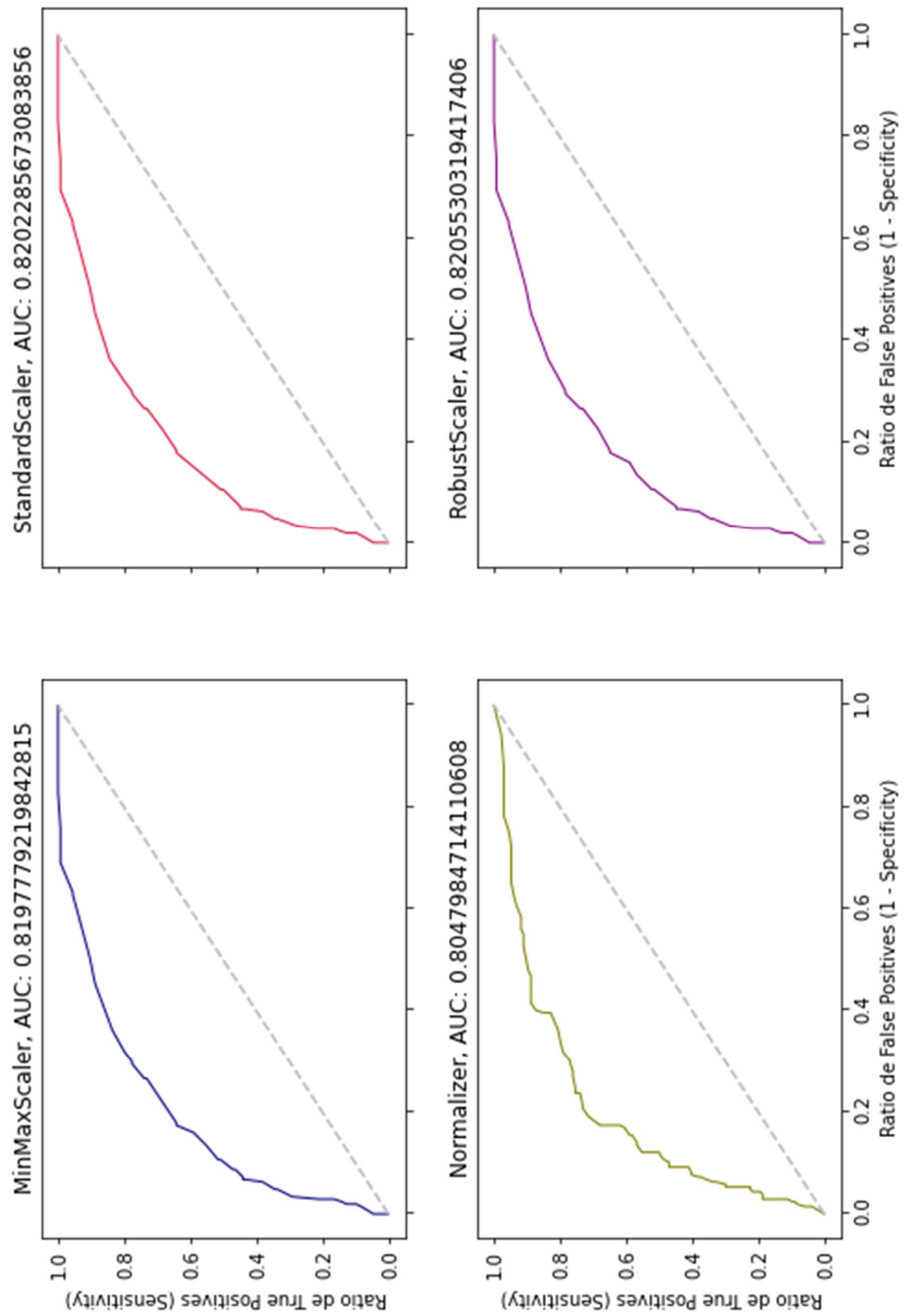


Figura 11 ROC Curves para *Random Forest*

3.8. Deep Learning (Artificial Neural Networks)

La idea que hay detrás del *Deep Learning* es la creación de modelos capaces de representar conceptos complejos a partir de otros más sencillos. Este proceso se realiza mediante la creación automática de una jerarquía en la que se empieza por estos conceptos más sencillos y conforme se va adentrando en esta jerarquía los conceptos pasan a ser más complejos (calculados a partir de los resultados obtenidos en el nivel anterior de la jerarquía). Esto permite descomponer el concepto a representar a partir de representaciones más sencillas y fáciles de manejar. Si, como es usual, esta jerarquía consta de diferentes niveles (capas) entonces se está hablando de la profundidad del modelo, de ahí el concepto de *Deep Learning*.

Aunque existen diferentes modelos que están relacionados con el *Deep Learning*, el más extendido es el de las *Artificial Neural Networks* (ANN). Este modelo está inspirado en el mecanismo de comunicación de las neuronas biológicas (Bosch Rué et al., n.d.).

En este caso, se presenta un tipo específico de ANN llamado *Multilayer Perceptron* (MLP).

Éstos son los principales inconvenientes y ventajas del *Deep Learning*:

| Ventajas | Inconvenientes |
|---|---|
| Facilita la gestión de datos no estructurados, como por ejemplo reconocimiento de imágenes y aplicaciones de visión artificial. | Requiere de importantes recursos tanto de CPU (GPUs preferiblemente) y de memoria para ejecutarlo. |
| Si está bien parametrizado permite disminuir el número de errores y por tanto obtener resultados más fiables. | Dificultad para establecer el <i>tuning</i> adecuado a nivel de capas y numero de nodos. Es un proceso más cercano al prueba y error. |
| Una vez en funcionamiento el sistema el posterior mantenimiento y ajuste requiere pocos recursos. | Implementación inicial costosa. |

La clase utilizada para implementar este modelo ha sido `sklearn.neural_network`
`import MLPClassifier (Sklearn.Neural_network.MLPClassifier — Scikit-Learn 1.2.0`
Documentation, n.d.). Para la ejecución de este algoritmo, el conjunto de parámetros es el siguiente:

```
params = {'nn__solver': ['sgd', 'adam'],
          'nn__activation' : ['tanh', 'relu'],
          'nn__max_iter': [4000],
          'nn__alpha': 10.0 ** -np.arange(1, 3),
          'nn__hidden_layer_sizes': np.arange(10, 15),
          'nn__batch_size' : [600]}
```

Donde:

- *solver*: Algoritmo de cálculo de optimización de pesos:
 - o *sgd*: *Stochastic Gradient Descent*
 - o *adam*: optimizador estocástico basado en gradientes propuesto por Kingma, Diederik y Jimmy Ba. Es una versión de *sgd*.
- *activation*: Función de activación para la capa oculta:
 - o *tanh*: función tangente hiperbólica
 - o *relu*: $f(x) = \max(0, x)$
- *max_iter*: Número máximo de iteraciones
- *alpha*: Peso del término de regularización L2. El término de regularización L2 se divide por el tamaño de la muestra cuando se suma al *loss*.
- *hidden_layer_sizes*: Número de *neuronas* (nodos)
- *batch_size*: Tamaño de cada *batch*.

El resumen de mejores métricas obtenidas por *transformer* es:

| | classifier | transformer | best_parameter_set | kappa | auc | accuracy | precision | sensitivity | specificity |
|---|-------------------|--------------------|---------------------------|--------------|------------|-----------------|------------------|--------------------|--------------------|
| 0 | MLPClassifier | MinMaxScaler | 39 | 0.368706 | 0.749946 | 0.706745 | 0.649123 | 0.552239 | 0.806763 |
| 1 | MLPClassifier | StandardScaler | 27 | 0.383698 | 0.761158 | 0.706745 | 0.628788 | 0.619403 | 0.763285 |
| 2 | MLPClassifier | Normalizer | 37 | 0.255444 | 0.739419 | 0.668622 | 0.626506 | 0.388060 | 0.850242 |
| 3 | MLPClassifier | RobustScaler | 23 | 0.330845 | 0.772442 | 0.686217 | 0.611570 | 0.552239 | 0.772947 |

Figura 12 Métricas para *Deep Learning*

Se puede observar que, tal y como apuntan todas las métricas con diferencia, el *transformer* *StandardScaler* es el que proporciona mejores resultados. Estos resultados son bastante deficientes basándonos en el valor de *kappa*. El mejor resultado es considerado como *justo*.

El `best_parameter_set` se corresponde en este caso a

```
{
  'nn__activation': 'relu',
  'nn__alpha': 0.1,
  'nn__batch_size': 600,
  'nn__hidden_layer_sizes': 13,
  'nn__max_iter': 4000,
  'nn__solver': 'adam'
}
```

En las gráficas de ROC curve se puede apreciar también que el *transformer* `StandardScaler` es el que proporciona mejores resultados:

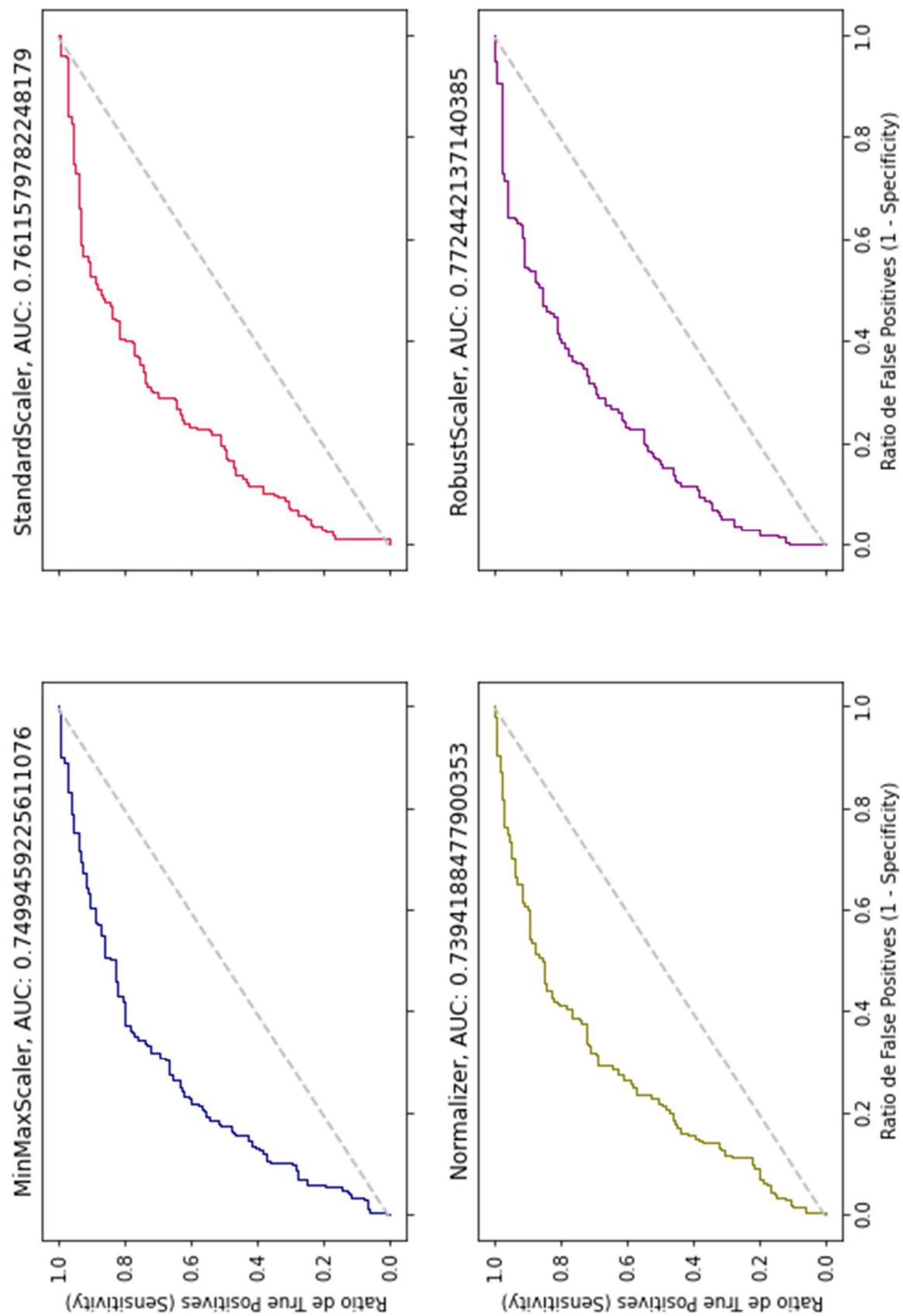


Figura 13 ROC Curves para *Deep Learning*

3.9. Comparación de resultados

En la siguiente tabla se muestran los mejores resultados obtenidos para cada uno de los algoritmos contemplados en este trabajo:

| algorithm | transformer | kappa | auc | accuracy | precision | sensitivity | specificity | time (seconds) |
|---------------------------|----------------|----------|----------|----------|-----------|-------------|-------------|-------------------|
| kNN | Normalizer | 0,400097 | 0,794704 | 0,724340 | 0,688679 | 0,544776 | 0,840580 | 12,34 |
| Naïve Bayes | MinMaxScaler | 0,184157 | 0,675211 | 0,571848 | 0,471429 | 0,738806 | 0,463768 | 26,13 |
| Support Vector Machine | StandardScaler | 0,433367 | 0,787908 | 0,736070 | 0,689655 | 0,597015 | 0,826087 | 407,40 |
| Random Forest | StandardScaler | 0,440952 | 0,820229 | 0,744868 | 0,732673 | 0,552239 | 0,869565 | 208,24 |
| Deep Learning | StandardScaler | 0,383698 | 0,761158 | 0,706745 | 0,628788 | 0,619403 | 0,763285 | 1273,70 |

Como puede observarse el algoritmo *Random Forest* es el que proporciona los mejores resultados, siendo el algoritmo *Naïve Bayes* el que peores números presenta con diferencia.

De todas formas, los resultados en general no son muy impresionantes. Basándose en la interpretación de Landis y Koch sobre el valor de *Kappa*, éste no proporciona más que un resultado moderado para el mejor caso.

Respecto a los algoritmos de transformación mayoritariamente la transformación *Standard*. En el caso del algoritmo de *Naïve Bayes*, aunque el mejor resultado estricto se produjo con el algoritmo *MinMax*, el siguiente mejor resultado (prácticamente igual al del mejor resultado) fue también con la transformación *Standard*.

Finalmente se ha añadido una columna con el tiempo en segundos del tiempo que ha llevado cada ejecución. Este tiempo es puramente estimativo (puesto que esto depende del entorno de ejecución), para poder establecer una idea comparativa de tiempos dentro del mismo entorno de ejecución. Obviamente lo que se persigue al final es obtener el mejor resultado posible, pero esta métrica proporciona una idea de los requisitos de CPU y memoria en el entorno de ejecución.

Por ejemplo, considerando la rapidez del algoritmo kNN, éste no difiere mucho en cuanto a resultados con el de mejor rendimiento, Random Forest, y los requisitos de CPU y memoria no

son comparables en ningún caso, por no compararlo con el modelo de *Deep Learning* donde además está presentando mucho mejor rendimiento.

4. Continuidad del trabajo

Una vez finalizado el trabajo tal y como estaba definido inicialmente, y dados los resultados obtenidos, una ampliación de este para la mejora de los resultados podría contemplar:

- Un estudio más profundo de los datos de entrada iniciales para poder hacer una selección de las *features* más acurada y ajustada a la realidad de los datos. Las decisiones tomadas para los pasos ejecutados para *depurar* el *dataset* inicial han sido tomadas de una forma más o menos estandarizada (decidir la correlación entre columnas de 0.8, la decisión sobre la baja varianza de una *feature*, etc...) y probablemente una elección de estos filtrajes más *ad-hoc* para los datos tratados proporcionaría mejores resultados.
- Una mejor elección o ampliación de los conjuntos de hiperparámetros a utilizar.
- Aprovechar el propósito final de la clase `Pipeline` para combinar diferentes mecanismos de transformación antes de la ejecución del modelo (en lugar de sólo aplicar un único método de transformación en cada iteración), así como experimentar con más métodos o variaciones de los ya utilizados.
- Por supuesto, utilizar otros modelos/algoritmos de Machine Learning/Deep Learning o alternativas de los ya utilizados.
- Investigar en otras librerías disponibles. No ya para mejora de resultados (deberían proporcionar los mismos resultados) sino para buscar alternativas donde los requisitos de los recursos de CPU/GPU y Memoria sean menos exigentes.

5. Glosario

COX-1: Ciclooxygenasa-1

TIC: Tecnologías de la Información las Comunicaciones

BSD: Berkeley Software Distribution

kNN: k-Nearest Neighbor

SVM: Support Vector Machine

CSV: Comma Separated Values

ROC curve: Receiver Operating Characteristic Curve

AUC: Area Under the Curve

CPU: Centra Processing Unit

GPU: Graphics Processing Unit

6. Bibliografía

- 1.9. *Naive Bayes* — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved January 5, 2023, from https://scikit-learn.org/stable/modules/naive_bayes.html
- An Introduction to GridSearchCV | What is Grid Search | Great Learning*. (n.d.). Retrieved January 5, 2023, from <https://www.mygreatlearning.com/blog/gridsearchcv/>
- Assay Report Card*. (n.d.). Retrieved January 5, 2023, from https://www.ebi.ac.uk/chembl/assay_report_card/CHEMBL1909130/
- Bosch Rué, A., Casas-Roma, J., & Lozano Bagén, Toni. (n.d.). *Deep learning : principios y fundamentos*. colabtools/drive.py at master · googlecolab/colabtools · GitHub. (n.d.). Retrieved January 5, 2023, from <https://github.com/googlecolab/colabtools/blob/master/google/colab/drive.py>
- Gómez-Luque A. (2005). *Inhibidores de la COX ¿hacia dónde vamos?*
- Máquinas de Vector Soporte (Support Vector Machines, SVMs)*. (n.d.). Retrieved January 7, 2023, from https://www.cienciadedatos.net/documentos/34_maquinas_de_vector_soporte_support_vector_machines
- pandas.read_csv* — *pandas 1.5.2 documentation*. (n.d.). Retrieved January 5, 2023, from https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
- R: The R Project for Statistical Computing*. (n.d.). Retrieved January 5, 2023, from <https://www.r-project.org/>
- RDKit*. (n.d.). Retrieved January 5, 2023, from <https://rdkit.org/>
- scikit-learn: machine learning in Python* — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved January 5, 2023, from <https://scikit-learn.org/stable/>
- sklearn.ensemble.RandomForestClassifier* — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>
- sklearn.feature_selection.VarianceThreshold* — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved January 5, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html
- sklearn.model_selection.GridSearchCV* — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved January 5, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

sklearn.model_selection.train_test_split — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved January 5, 2023, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

sklearn.naive_bayes.GaussianNB — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.GaussianNB.html

sklearn.neighbors.KNeighborsClassifier — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html>

sklearn.neural_network.MLPClassifier — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html

sklearn.pipeline.Pipeline — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved January 5, 2023, from <https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html>

sklearn.preprocessing.MinMaxScaler — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>

sklearn.preprocessing.Normalizer — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.Normalizer.html>

sklearn.preprocessing.RobustScaler — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html>

sklearn.preprocessing.StandardScaler — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html>

sklearn.svm.SVC — *scikit-learn 1.2.0 documentation*. (n.d.). Retrieved December 21, 2022, from <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html>

The Measurement of Observer Agreement for Categorical Data on JSTOR. (n.d.). Retrieved January 5, 2023, from <https://www.jstor.org/stable/2529310>

Understanding Random Forest. How the Algorithm Works and Why it Is... | by Tony Yiu | Towards Data Science. (n.d.). Retrieved January 7, 2023, from <https://towardsdatascience.com/understanding-random-forest-58381e0602d2>

Vane J.R. (1971). Inhibition of prostaglandin synthesis as a mechanism of action for aspirin-like drugs. *Nature New Biology*, 231, 232–235.

Welcome To Colaboratory - Colaboratory. (n.d.). Retrieved January 5, 2023, from <https://colab.research.google.com/>

Welcome to Python.org. (n.d.). Retrieved January 5, 2023, from <https://www.python.org/>

What is Cross Validation and its types in Machine learning? Great Learning. (n.d.). Retrieved January 5, 2023, from <https://www.mygreatlearning.com/blog/cross-validation/>

7. Anexos

A continuación, se detallan todos los archivos fuente que forman parte de este trabajo. De todas formas, también son accesibles los mismos, así como el archivo de datos inicial (CHEMBL221_COX1Prostaglandin_Desc.csv) como los archivos ya procesados (X_train.csv, X_test.csv, y_train.csv, y_test.csv) en el siguiente repositorio GitHub: <https://github.com/jvillara-tfm/tfm>.

7.1. Librería *Python* común

tfm.py

```
# Librerías necesarias
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import cohen_kappa_score
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import Normalizer
from sklearn.preprocessing import RobustScaler
from google.colab import drive
import matplotlib as mpl
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import time

"""
Librerías que implementa las funciones necesarias para el desarrollo del
TFM
"""

# Lista de transformadores por defecto
default_transformers = [MinMaxScaler(), StandardScaler(), Normalizer(),
RobustScaler()]

# Scoring Method por defecto para GridSearchCV
default_scoring_method = 'accuracy'

# Lista de colores para diferentes gráficos
colors = ["navy", "crimson", "olive", "purple", "lightsalmon",
"chartreuse", "cyan", "hotpink", "lime"]
```

```

def load_data(basePath = '/content/drive/My Drive/TFM'):
    """
        Esta función carga los datasets correspondientes a los conjuntos de
        datos de train (X, y) y test (X, y)
        calculados y distribuidos previamente.

        PARAMETROS
        -----
        basePath : str
            El path base donde residen los datos de entrada. Opcional. Valor
            por defecto '/content/drive/My Drive/TFM'

        RETORNA
        -----
        X_train: matrix
            features X de train
        y_train: vector (0/1)
            resultados y de train
        X_test: matrix
            features X de test
        y_test: vector (0/1)
            resultados y de test
    """

    # Los datos se obtienen de Google Drive. Se establece una conexión
    drive.mount('/content/drive', force_remount=True)

    # Lectura de los ficheros correspondientes a training y testing
    with open(basePath + '/X_train.csv', 'r') as f:
        X_train = pd.read_csv(f).values

    with open(basePath + '/y_train.csv', 'r') as f:
        y_train = pd.read_csv(f).values

    with open(basePath + '/X_test.csv', 'r') as f:
        X_test = pd.read_csv(f).values

    with open(basePath + '/y_test.csv', 'r') as f:
        y_test = pd.read_csv(f).values

    # Cierre de la conexión a Google Drive
    drive.flush_and_unmount()

    # Adaptación de los dataframes y_train e y_test a un vector
    unidimensional
    y_train = np.reshape(y_train, y_train.shape[0])
    y_test = np.reshape(y_test, y_test.shape[0])

```



```

# Retorno de los datos
return X_train, y_train, X_test, y_test

def calculate_models(id, params, X_train, y_train, X_test, y_test,
                    classifier, transformers = default_transformers, scoring_method =
                    default_scoring_method):
    """
    Esta función calcula el mejor resultado a partir de un conjunto de
    parámetros dados para un clasificador
    aplicando diferentes métodos de transformación.

    PARAMETROS
    -----
    id : str
        Identificador del clasificador. Está relacionado con los
        identificadores especificados en 'params'
    params: dictionary
        Conjuntos de parámetros para el clasificador. Cada parámetro ha
        de ir prefijado con el id (anterior) y 2 underscores
    X_train: matrix
        Matriz con las features de entrada de train
    y_train: vector
        Vector con los resultados (0/1) de train
    X_test: matrix
        Matriz con las features de entrada de test
    y_test: vector
        Vector con los resultados (0/1) de train
    classifier: object
        Instancia del clasificador a utilizar
    transformers: list
        Lista de transformadores de datos a utilizar. Por defecto es la
        variable global default_transformers
    scoring_method: str
        Método de evaluación a utilizar en el parámetro 'scoring' del
        constructor la clase GridSearchCV

    RETORNA
    -----
    outcome: dictionary
        relación de métricas y resultados del cálculo de las diferentes
        combinaciones clasificador/transformadores
    """

    # Medida del tiempo de ejecución
    start_time = time.time()

    # Definición de las listas que contendrán diferentes resultados y
    métricas para cada transformador

```

```

best_scores = []
train_scores = []
test_scores = []
best_parameters = []
predictions = []
predictions_proba = []
confusion_matrixes = []
fprs = []
tprs = []
thresholds = []
aucls = []
kappas = []
best_indexes = []
accuracies = []
sensitivities = []
specificities = []
precisions = []
classifier_names = []
transformer_names = []

# Dictionary resultado
outcome = {}

# Para cada transformer
for transformer in transformers:
    # Definición del pipeline del transformer en curso con el
    # clasificador
    pipeline = Pipeline([('transformer', transformer),
                          (id, classifier)])

    # Definición del GridSearchCV combinando el pipeline calculado
    # con los parámetros y estableciendo el método de scoring
    grid = GridSearchCV(pipeline,
                        param_grid = params,
                        scoring = scoring_method)

    # Cálculo de los modelos
    grid.fit(X_train, y_train)

    # Nombres de clasificador y transformador
    classifier_names.append(type(classifier).__name__)
    transformer_names.append(type(transformer).__name__)
    # Agregación de las diferentes métricas obtenidas
    best_scores.append(grid.best_score_)
    best_indexes.append(grid.best_index_)
    test_scores.append(grid.score(X_test, y_test))
    train_scores.append(grid.score(X_train, y_train))
    best_parameters.append(grid.best_params_)
    y_pred = grid.predict(X_test)

```

```

y_pred_proba = grid.predict_proba(X_test)[: ,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
auc = roc_auc_score(y_test, y_pred_proba)
aucs.append(auc)
fprs.append(fpr)
tprs.append(tpr)
thresholds.append(thresholds)
predictions.append(y_pred)
predictions_proba.append(y_pred_proba)
kappas.append(cohen_kappa_score(y_test, y_pred))
cm = confusion_matrix(y_test, y_pred)
# Cálculo de accuracy, precision, sensitivity, specificity
TN = cm[0, 0] # True Negatives
FN = cm[1, 0] # False Negatives
FP = cm[0, 1] # False Positives
TP = cm[1, 1] # True Positives
accuracy = (TP + TN) / (TP + TN + FP + FN)
precision = TP / (TP + FP)
sensitivity = TP / (TP + FN)
specificity = TN / (TN + FP)
accuracies.append(accuracy)
precisions.append(precision)
sensitivities.append(sensitivity)
specificities.append(specificity)
confusion_matrixes.append(cm)

# Asignación de los resultados al diccionario
outcome["best_scores"] = best_scores
outcome["test_scores"] = test_scores
outcome["train_scores"] = train_scores
outcome["best_parameters"] = best_parameters
outcome["best_indexes"] = best_indexes
outcome["predictions"] = predictions
outcome["predictions_proba"] = predictions_proba
outcome["confusion_matrixes"] = confusion_matrixes
outcome["aucs"] = aucs
outcome["fprs"] = fprs
outcome["tprs"] = tprs
outcome["thresholds"] = thresholds
outcome["classifier"] = classifier
outcome["transformers"] = transformers
outcome["y_pred"] = predictions
outcome["y_pred_proba"] = predictions_proba
outcome["elements"] = len(transformers)
outcome["kappas"] = kappas
outcome["accuracies"] = accuracies
outcome["precisions"] = precisions
outcome["sensitivities"] = sensitivities

```

```

outcome["specificities"] = specificities

# Generación del report de métricas para comparación
report = {
    'classifier': classifier_names,
    'transformer': transformer_names,
    'best_parameter_set': best_indexes,
    'kappa': kappas,
    'auc': aucs,
    'accuracy': accuracies,
    'precision': precisions,
    'sensitivity': sensitivities,
    'specificity': specificities
}

outcome["report"] = pd.DataFrame(report)

# Cálculo final del tiempo de ejecución
end_time = time.time()
outcome["execution_time"] = (end_time - start_time)

# Retorno del dictionary calculado
return outcome

def show_graphs(outcome):
    """
    Esta función muestra los gráficos de los resultados obtenidos para un
    clasificador y los diferentes transformadores

    PARAMETROS
    -----
    outcome : dictionary
        Diccionario obtenido de la ejecución de los modelos dados para un
        clasificador y diferentes transformadores
    """

    # Obtención del ratio de false positives y true negatives
    fprs = outcome["fprs"]
    tprs = outcome["tprs"]

    # Información para el título del gráfico
    classifier = outcome["classifier"]

    # Lista de transformadores de cada uno de los resultados
    transformers = outcome["transformers"]

    # Valores de AUC de cada cálculo del modelo
    aucs = outcome["aucs"]

```

```

# Definición del grid de presentación
fig, axs = plt.subplots(2, 2)
row = 0
col = 0

# Definición del tamaño total
fig.set_size_inches((12,8))

# Definición del tamaño de la fuente
mpl.rcParams.update({'font.size': 10})

# Iteración por cada elemento (true positive ratio)
for index, tpr in enumerate(tprs):
    fpr = fprs[index] # false positive ratio

    # Plot a partir de los valores de tpr y fpr
    axs[row, col].plot(fpr, tpr, linewidth = 1, color =
colors[index%len(colors)])

    # Dibujo de la diagonal
    axs[row, col].plot([0,1], [0,1], 'k--', color = '#c0c0c0')

    # Construcción del título del gráfico
    title = type(transformers[index]).__name__ + ", "
    title = title + "AUC: " + str(aucs[index])
    axs[row, col].set_title(title)

    # Actualiza la posición del grid
    col = col + 1
    if(col == 2):
        col = 0
        row = row + 1

# Leyenda para los ejes X, Y
for ax in axs.flat:
    ax.set(xlabel='Ratio de False Positives (1 - Specificity)',
          ylabel = 'Ratio de True Positives (Sensitivity)')

for ax in axs.flat:
    ax.label_outer()

def execute_model(id, classifier, params, transformers =
default_transformers, scoring_method = default_scoring_method,
                basePath = '/content/drive/My Drive/TFM'):
    """

```

```

    Clase de utilidad para ejecutar un modelo con diferentes
    transformadores con valores por defecto

    PARAMETROS
    -----
    id : str
        Identificador del clasificador. Está relacionado con los
        identificadores especificados en 'params'
    classifier: object
        Instancia del clasificador a utilizar
    params: dictionary
        Conjuntos de parámetros para el clasificador. Cada parámetro ha
        de ir prefijado con el id (anterior) y 2 underscores
    transformers: list
        Lista de transformadores de datos a utilizar. Por defecto es la
        variable global default_transformers
    scoring_method: str
        Método de evaluación a utilizar en el parámetro 'scoring' del
        construtor la clase GridSearchCV

    RETORNA
    -----
    outcome: dictionary
        relación de métricas y resultados del cálculo de las diferentes
        combinaciones clasificador/transformadores
    """

    # Obtención de los datasets necesarios de training y test
    X_train, y_train, X_test, y_test = load_data()

    # Cálculo de los modelos
    outcome = calculate_models(id, params, X_train, y_train, X_test,
                               y_test, classifier, transformers, scoring_method)

    return outcome

```

7.2. Notebook de preparación de datos

data_setup.ipynb

```
# Instalacion de las dependencias (developed from Google Colab)
! pip install -q scikit-learn matplotlib tensorflow numpy pandas
```

```
# Librerías a utilizar
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from google.colab import drive
from sklearn import preprocessing
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split
```

```
# Los datos originales se obtienen de Google Drive. Se establece una
conexión
```

```
drive.mount('/content/drive')
```

```
# Lectura del fichero
```

```
with open('/content/drive/My
Drive/TFM/CHEMBL221_COX1Prostaglandin_Desc.csv', 'r') as f:
    original_data = pd.read_csv(f)
```

```
# Cierre de la conexión con Google Drive
```

```
drive.flush_and_unmount()
```

```
# Preparación de los datos
```

```
# Eliminación de las 7 primeras columnas que no son de utilidad para
nuestro propósito
```

```
work_data = original_data.drop(original_data.iloc[:, 0:7], axis = 1)
```

```
# Recogida de la columna Activity (el resultado a estudiar) y eliminación
de la misma
```

```
activity = work_data['Activity']
work_data = work_data.drop('Activity', axis = 1) # Eliminación del resto
de datos
```

```
# Eliminación de columnas con baja varianza (threshold = 1)
```

```
lvf = VarianceThreshold(threshold = 1)
lvf.fit(work_data)
```

```
# Recogemos las columnas donde su varianza está por debajo del threshold
establecido
```

```
lvc = [column for column in work_data.columns
```

```

        if column not in
work_data.columns[lvf.get_support()]]

cr = [i.strip() for i in lvc] # Columnas a borrar
work_data = work_data.drop(cr, axis=1) # Eliminación de columnas con baja
varianza (columnas constantes tienen varianza 0)

# Eliminación de columnas con alta correlacion (>= 0.8)
cm = work_data.corr().abs() # Matriz positiva de correlación

# Obtención de los elementos de la matriz por encima de la diagonal de la
matriz de correlación
upper_triangle = cm.where(np.triu(np.ones(cm.shape), k=1).astype(bool))

# Selección de las columnas donde la correlación sea >=0.8
cr = [column for column in upper_triangle.columns if
any(upper_triangle[column] >= 0.8)]
# Eliminación de las columnas con alta correlación
work_data = work_data.drop(cr, axis=1)

# Obtain Training and Test sets (80%-20%)
X_train, X_test, y_train, y_test = train_test_split(work_data, activity,
test_size=0.20, random_state = 19680925, stratify=activity)

# Escritura de los ficheros obtenidos
# Serán almacenados en Google Drive
drive.mount('/content/drive', force_remount = True)

X_train.to_csv('/content/drive/My Drive/TFM/X_train.csv', index = False)
y_train.to_csv('/content/drive/My Drive/TFM/y_train.csv', index = False)
X_test.to_csv('/content/drive/My Drive/TFM/X_test.csv', index = False)
y_test.to_csv('/content/drive/My Drive/TFM/y_test.csv', index = False)

drive.flush_and_unmount()

```


7.3. Notebook de implementación *k-NN*

knn_implementation.ipynb

```
# Instalacion de las dependencias (developed from Google Colab)
! pip install -q scikit-learn matplotlib tensorflow numpy pandas

# Librerias a utilizar
import sys
from google.colab import drive
from sklearn.neighbors import KNeighborsClassifier
# Carga de la librería "tfm_lib"
drive.mount('/content/drive', force_remount=True)

sys.path.append('/content/drive/My Drive/TFM')
import tfm_lib as tfm
drive.flush_and_unmount()

# Definición del set de parámetros del clasificador
params = [{'knn__n_neighbors': [3, 5, 7, 9, 11, 13]}]

# Ejecución del modelo en función de los parámetros y los transformadores
por defecto
outcome = tfm.execute_model('knn', KNeighborsClassifier(), params)
# Sumario de diferentes métricas para cada transformador
outcome["report"].style

# Best parameters
outcome["best_parameters"]
# Representación ROC del cálculo del modelo para cada clasificador
tfm.show_graphs(outcome)
```

7.4. Notebook de implementación *Naïve-Bayes*

naivebayes_implementation.ipynb

```
# Instalacion de las dependencias (developed from Google Colab)
! pip install -q scikit-learn matplotlib tensorflow numpy pandas

# Librerías a utilizar
import sys
import numpy as np
from google.colab import drive
from sklearn.naive_bayes import GaussianNB

# Carga de la librería "tfm_lib"
drive.mount('/content/drive', force_remount=True)

sys.path.append('/content/drive/My Drive/TFM')
import tfm_lib as tfm
drive.flush_and_unmount()

# Definición del set de parámetros del clasificador
params = {'nb__var_smoothing': np.logspace(0, -9, num=100)}

# Ejecución del modelo en función de los parámetros y los transformadores
por defecto
outcome = tfm.execute_model('nb', GaussianNB(), params)
# Sumario de diferentes métricas para cada transformador
outcome["report"].style

# Best Parameters
outcome["best_parameters"]

# Representación ROC del cálculo del modelo para cada clasificador
tfm.show_graphs(outcome)
```

7.5. Notebook de implementación SVM

svm_implementation.ipynb

```
# Instalacion de las dependencias (developed from Google Colab)
! pip install -q scikit-learn matplotlib tensorflow numpy pandas

# Librerías a utilizar
import sys
from google.colab import drive
from sklearn.svm import SVC

# Carga de la librería "tfm_lib"
drive.mount('/content/drive', force_remount=True)

sys.path.append('/content/drive/My Drive/TFM')
import tfm_lib as tfm
drive.flush_and_unmount()

# Definición del set de parámetros del clasificador
params = {'svm__C': [0.1, 1, 10],
          'svm__gamma': [1, 0.1, 0.01],
          'svm__kernel': ['rbf', 'linear', 'sigmoid']}

# Ejecución del modelo en función de los parámetros y los transformadores
por defecto
outcome = tfm.execute_model('svm', SVC(probability = True), params)
# Sumario de diferentes métricas para cada transformador
outcome["report"].style

# Best Parameters
outcome["best_parameters"]

# Representación ROC del cálculo del modelo para cada clasificador
tfm.show_graphs(outcome)
```

7.6. Notebook de implementación *Random Forest*

randomforest_implementation.ipynb

```
# Instalacion de las dependencias (developed from Google Colab)
! pip install -q scikit-learn matplotlib tensorflow numpy pandas

# Librerias a utilizar
import sys
from google.colab import drive
from sklearn.ensemble import RandomForestClassifier

# Carga de la librería "tfm_lib"
drive.mount('/content/drive', force_remount=True)

sys.path.append('/content/drive/My Drive/TFM')
import tfm_lib as tfm
drive.flush_and_unmount()

# Definición del set de parámetros del clasificador
params = [{'rf__random_state': [0, None], 'rf__bootstrap': [True, False],
'rf__n_estimators' : [10, 30, 50, 80, 100], 'rf__max_features': ['sqrt',
'log2']}]

# Ejecución del modelo en función de los parámetros y los transformadores
por defecto
outcome = tfm.execute_model('rf', RandomForestClassifier(), params)

# Sumario de diferentes métricas para cada transformador
outcome["report"].style

# Best Parameters
outcome["best_parameters"]

# Representación ROC del cálculo del modelo para cada clasificador
tfm.show_graphs(outcome)
```

7.7. Notebook de implementación *Deep Learning*

nn_implementation.ipynb

```
# Instalacion de las dependencias (developed from Google Colab)
! pip install -q scikit-learn matplotlib tensorflow numpy pandas

# Librerias a utilizar
import sys
import numpy as np
from google.colab import drive
from sklearn.neural_network import MLPClassifier

# Carga de la librería "tfm_lib"
drive.mount('/content/drive', force_remount=True)

sys.path.append('/content/drive/My Drive/TFM')
import tfm_lib as tfm
drive.flush_and_unmount()

# Definición del set de parámetros del clasificador

params = {'nn__solver': ['sgd', 'adam'],
          'nn__activation' : ['tanh', 'relu'],
          'nn__max_iter': [4000],
          'nn__alpha': 10.0 ** -np.arange(1, 3),
          'nn__hidden_layer_sizes': np.arange(10, 15),
          'nn__batch_size' : [600]}

# Ejecución del modelo en función de los parámetros y los transformadores
por defecto
outcome = tfm.execute_model('nn', MLPClassifier(), params)

# Sumario de diferentes métricas para cada transformador
outcome["report"].style

# Best Parameters
outcome["best_parameters"]

# Representación ROC del cálculo del modelo para cada clasificador
tfm.show_graphs(outcome)
```