Reducción de la dimensionalidad mediante métodos de selección de características en microarrays de ADN

Miguel Maseda Tarin
Grado en Informática
Inteligencia artificial

Dr. David Isern Alarcón
Dr. Carles Ventura Royo

02/01/2019

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Reducción de la dimensionalidad mediante métodos de selección de características en microarrays de ADN*
Nombre del autor:	*Miguel Maseda Tarin*
Nombre del consultor/a:	*Dr. David Isern Alarcón*
Nombre del PRA:	*Dr. Carles Ventura Royo*
Fecha de entrega (mm/aaaa):	01/2019
Titulación:	*Grado en Informática*
Área del Trabajo Final:	*Inteligencia artificial*
Idioma del trabajo:	*Castellano*
Palabras clave	*Reducción de dimensionalidad, selección de características, métodos híbridos.*

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

El trabajo de fin de grado se centra en la maldición de la dimensionalidad que podemos encontrar en ciertos conjuntos de datos, en concreto, en aquellos conjuntos de datos donde el número de características que se disponen superan los cientos o miles en cada una de las muestras. Buscamos una mejora en la clasificación realizando un tratamiento previo al conjunto de datos, la selección de características.

La aplicación de métodos de selección de características a los conjuntos de datos pretende reducir la dimensionalidad, con la intención de encontrar un subconjunto de características que sea capaz de describir el problema de manera apropiada. Para ello, utilizaremos un conjunto de datos que es un referente en los estudios de *microarrays* de ADN [2], al que le aplicaremos cuatro métodos de selección de características, para luego comprobar los resultados ante tres métodos de aprendizaje computacional. Nos centraremos en dos métodos de filtro (*f-score* y *mRMR*), un método *wrapper* (*SFS_forward*) y, especialmente, en un método híbrido, que es una adaptación del trabajo [25].

Podremos observar las distintas ventajas e inconvenientes que ofrece cada uno de los métodos de selección de características y las variaciones que nos encontramos en los resultados en función del método de aprendizaje computacional elegido para la clasificación.

Abstract (in English, 250 words or less):

This document deals with the curse of dimensionality that we can find in datasets, to be specific, in those datasets where the number of features are counted as hundreds or thousands in each sample. We are looking for an improvement in the classification of our dataset through feature selection methods.

The application of feature selection methods to datasets aims to reduce their dimensionality, with the intention of finding a feature subset that explains the problem in the appropriate way. With that in mind, we will use one benchmark dataset in the DNA microarrays studies [2], we will apply four feature selection methods so that we can verify the results with three different machine learning algorithms. We will use two filter methods (f-score and mRMR), one wrapper (SFS_forward) and, specially, a hybrid method, an adaptation of the work [25].

We can observe the different advantages and issues offered by each of the feature selection methods and the different results that we obtain based on the machine learning method used for the classification task.

Índice

Lista de figuras

Tabla de ilustraciones

1 INTRODUCCIÓN

1.1 CONTEXTO Y JUSTIFICACIÓN DEL TRABAJO

En la medicina, el diagnosticar correctamente una enfermedad es el punto de inflexión para el tratamiento de ésta. Si, además, se diagnostica en un estadio temprano, los resultados del tratamiento pueden ser mucho más eficaces en el paciente. Por lo tanto, clasificar correctamente una enfermedad en un paciente puede llevar a su pronta recuperación.

En la actualidad, en el campo de la medicina se disponen de grandes cantidades de datos relacionados con los pacientes y uno de los temas clave es cómo clasificar los datos disponibles para poder hacer un uso adecuado de éstos. Mediante técnicas de minería de datos (*datamining,* DM) y de aprendizaje computacional (*machine learning*, ML) con métodos supervisados, se pretende que los conjuntos de datos utilizados sean clasificados correctamente.

Ahora bien, muchos de los conjuntos de datos que se manejan sufren de la denominada "maldición de la dimensionalidad" [1]. En nuestro caso, trabajaremos con datos referentes a las expresiones genéticas de la leucemia mieloide aguda (AML) y la leucemia linfoblástica aguda (ALL) del trabajo de Golub et al. [2]. El conjunto de datos consiste en 72 muestras divididas en dos clases, cada clase corresponde a uno de los tipos de cáncer mencionados anteriormente. Cada una de las muestras corresponde a una expresión genética que consta de 7.129 características (genes).

Las técnicas de DM y de ML necesitan un tratamiento previo de los datos antes de ser utilizadas. El tratamiento previo de los datos combina diferentes técnicas de distintos ámbitos, los algoritmos de preprocesamiento de datos más influyentes, que vienen indicados por el artículo de García S. et al [3], serían:

- Imputación de valores ausentes (EM, MI y kNNI)
- Filtrado de ruido (EF y IPF)
- Reducción de la dimensionalidad (LVF/W, MIFS, *Relief*, mRMR, PCA y LLE)
- Reducción de instancias (CNN, ENN, DROP, ICF y LVQ)
- Discretización (MDLP, ChiMerge y CAIM)
- Aprendizaje no balanceado (SMOTE)

Mediante algoritmos de aprendizaje supervisado se pretende clasificar correctamente los dos tipos de cáncer de que se dispone en el conjunto de datos. Pero para poder hacerlo, deberemos tratar previamente los datos. Debido al gran volumen de características que se disponen en nuestro conjunto de datos, la técnica que utilizaremos será la reducción de la dimensionalidad, y lo haremos mediante técnicas de

selección de características. Los distintos métodos que se disponen de selección de características son:

- Métodos de filtro.
- Métodos *wrapper*.
- Métodos empotrados.
- Métodos híbridos.

Los métodos de selección de características pretenden encontrar un subconjunto óptimo de atributos del conjunto de datos que nos describan de manera apropiada el problema. Es decir, el objetivo es la selección de características más influyentes de nuestro conjunto de datos mediante la eliminación de aquellos atributos que sean irrelevantes o redundantes.

La aplicación de procesos de selección de características a los conjuntos de datos con problemas de dimensionalidad nos da como resultado modelos más precisos, debido a la eliminación de los atributos irrelevantes y/o redundantes que, de no ser eliminados, puede dar lugar a problemas de correlación en los algoritmos de aprendizaje. Además, reducen el espacio necesario de almacenamiento de los datos, al igual que también reducen el espacio de búsqueda. Otros beneficios que se derivan de la aplicación de la reducción de la dimensionalidad son una mayor comprensión y visualización de los datos al ser modelos más simples que su original o la reducción de costes a la hora de recopilar la información.

Queremos demostrar que, mediante los métodos híbridos de selección de características, se obtienen unos resultados de clasificación mejores que mediante las técnicas simples de selección de características, al igual que se mejoran los tiempos de ejecución en los algoritmos de aprendizaje después de haber reducido la dimensionalidad mediante estas técnicas.

1.2 Objetivos del Trabajo

1.2.1 Objetivos generales

Mediante la aplicación del método híbrido creado se quiere demostrar que se obtienen unos resultados que mejoran la clasificación del conjunto de datos frente a la utilización de métodos simples de selección de características. Los resultados que se quieren mejorar son:

- Clasificación obtenida.
- Tiempos de ejecución sobre el algoritmo de aprendizaje.

1.2.2 Objetivos específicos

1. Preparación del conjunto de datos para ser estudiado.
2. Definición de métodos de filtro de selección de características a utilizar.
    a. *F-score*.
    b. *mRMR*.
    c. Aplicación de los métodos al conjunto de datos.
    d. Recolección de resultados.
3. Definición de los métodos *wrapper* de selección de características a utilizar.
    a. *SVM-backward*.
    b. *SVM-forward*.
    c. Aplicación de los métodos al conjunto de datos.
    d. Recolección de resultados.
4. Creación de un método híbrido de selección de características.
    a. Aplicación del método al conjunto de datos.
    b. Recolección de resultados.
5. Definición de las métricas para la comparación de resultados.
    a. Exactitud (*accuracy*).
    b. Exhaustividad (*recall*).
    c. Precisión.
6. Creación de un script en Python de los métodos indicados en los puntos 2, 3 y 4.
    a. Definición del algoritmo de aprendizaje a utilizar.
    b. Definición del uso de un método de validación cruzada.
7. Comparación de resultados de los puntos 2, 3 y 4.
8. Conclusiones.

1.2.3 Objetivos opcionales

- Aplicación de diferentes conjuntos de datos para el estudio.
- Creación de más de un método híbrido de selección de características.

1.3 ENFOQUE Y MÉTODO SEGUIDO

En conjuntos de datos donde la dimensionalidad supone un problema para una correcta aplicación de un método de aprendizaje, la aplicación de los métodos de selección de características ayuda a su posterior tratamiento, convirtiéndose en un paso previo fundamental para realizar los posteriores estudios. Por lo tanto, nos centramos en los cuatro métodos de selección de características que existen, para posteriormente

comprobar sus efectos en un conjunto de datos conocido y utilizado en muchos trabajos [2, 25, 41, 42, 43] en problemas de clasificación.

En una primera aproximación, veremos la descripción de los distintos métodos de selección de características, sus ventajas e inconvenientes. De esta manera, sabremos cómo se debe afrontar el estudio del conjunto de datos y ver qué herramientas nos pueden ser más útiles.

Seguidamente, aplicaremos cuatro métodos de selección de características al conjunto de datos seleccionado. Así, podremos ver qué resultados nos ofrece cada uno de los métodos y relacionarlo respecto a sus ventajas e inconvenientes. Dos de los métodos serán de filtro, uno *wrapper* y, por último, una adaptación del método híbrido descrito en [25] basado en los tres métodos vistos anteriormente.

Con los distintos resultados podremos hacer una comparación de los métodos para, así, ver su comportamiento tanto en tiempos de ejecución como niveles de clasificación que nos pueden ofrecer.

1.4 Planificación del Trabajo

1.4.1 Diagrama de Gantt

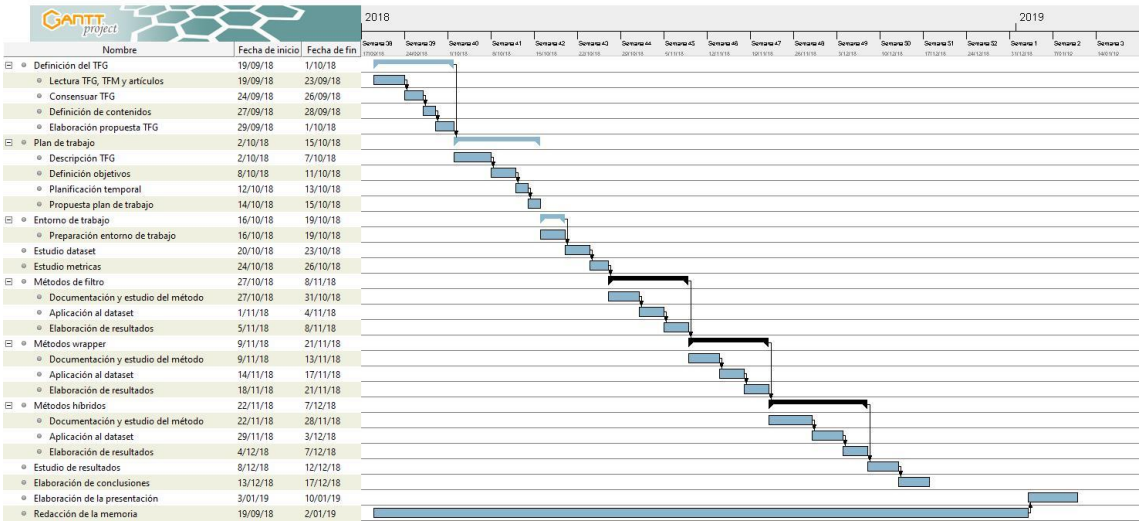


Ilustración 1 - Diagrama de Gantt

1.4.2 Recursos utilizados

- Python 3. Es un lenguaje de programación que soporta la programación orientada objetos, imperativa y funcional, haciendo hincapié en una sintaxis que favorezca un código legible. Se desarrolla bajo una licencia de código abierto [fuente Wikipedia: https://es.wikipedia.org/wiki/Python]

- Jupyter Notebook. Es una aplicación web de licencia abierta, que permite crear y compartir código, tiene soporte para más de 40 tipos de lenguajes de programación distintas, incluido Python. Los documentos "*notebooks*" permiten combinar texto enriquecido con código, permitiendo la creación de documentos junto con código que puede ser ejecutado. https://jupyter.org/.

- Scikit-learn. Biblioteca desarrollada en Python que incluye algoritmos tanto para uso en minería de datos como de aprendizaje computacional. https://scikit-learn.org/stable/.

- Scikit-feature. Biblioteca desarrollada en Python, con licencia de código abierto, que se centra en algoritmos de selección de características. http://featureselection.asu.edu/index.php.

- Mlxtend. Biblioteca desarrollada en Python, bajo una licencia de código abierto BSD, de algoritmos de aprendizaje computacional. De esta biblioteca, utilizaremos aquellos algoritmos que se centran en la selección de características. http://rasbt.github.io/mlxtend/.

- NumPy. Es uno de los módulos más completos de Python para la manipulación de datos numéricos, vectores y matrices, diseñado para el cálculo científico. http://www.numpy.org/.

- Pandas. Es una biblioteca para el lenguaje Python, basada en NumPy, que permite una fácil manipulación y uso de estructuras de datos y herramientas para el análisis de datos proporcionando un alto rendimiento. https://pandas.pydata.org/.

- Matplotlib. Es una biblioteca de Python que permite la generación de una gran variedad de gráficos: de líneas, de puntos, barras, histogramas, … Sirve como herramienta para la exploración de los datos generados tanto en Python como con sus bibliotecas NumPy o Pandas. https://matplotlib.org/.

1.5 Breve sumario de productos obtenidos

Al finalizar el TFG, los documentos presentados serán:

- Memoria del TFG.

- Presentación del TFG.
- Código en Python desarrollado.

1.6 BREVE DESCRIPCIÓN DE LOS OTROS CAPÍTULOS DE LA MEMORIA

En la segunda sección realizamos un análisis de los motivos que generan la aplicación de métodos de selección de características sea un paso previo en el estudio de los conjuntos de datos que sufren de una gran dimensionalidad. Se da una breve descripción de los distintos métodos que nos encontramos (filtro, *wrapper*, empotrados e híbridos), acabando con una serie de campos donde su aplicación se hace necesaria.

Seguidamente, en la tercera sección, utilizamos uno de los conjuntos de datos utilizados como referencia para el estudio de *microarrays* de ADN [2] para la aplicación de dos métodos de selección de características de filtro, uno *wrapper* y uno híbrido. Se realizará una descripción de cada uno de los métodos utilizados al igual que se mostrarán los resultados obtenidos.

Por último, en la cuarta sección, se realizará una comparación de los resultados de cada uno de los métodos. También, se realiza una reflexión de los resultados obtenidos respecto a los objetivos del trabajo. Por último, se describen una serie de observaciones que se deberían tener presente a la hora de tratar conjuntos de datos que no estén previamente tratados para su estudio.

2 SELECCIÓN DE CARACTERÍSTICAS

La dimensionalidad de un conjunto de datos viene definida por el número de atributos que se disponen en éste al igual que del número de muestras que tiene. Los conjuntos de datos que se manejan tanto para la detección de enfermedades, clasificación de textos, reconocimiento de imágenes, series temporales, etc. se caracterizan por tener un gran volumen de atributos. Si denotamos un conjunto de datos como X, el número de muestras que dispone serán $X = \{x_1, x_2, \dots, x_n\}$, donde cada una de estas muestras tiene una serie de características, $x_i = \{x_{i1}, x_{i2}, \dots, x_{im}\}$. El problema de la dimensionalidad se da cuando el valor m es excesivamente grande y, más aún, cuando el valor es mucho mayor que el número de muestras de las que disponemos ($m \gg n$), en este tipo de problemas, el algoritmo de aprendizaje puede sobrespecializar el problema si no se actúa sobre las características que tiene.

Debido a esta situación, para la aplicación correcta de los algoritmos de aprendizaje o de DM, como paso previo del procesamiento del conjunto de datos, es necesario una reducción de la dimensionalidad ya que, el conjunto de características que describe de manera correcta el modelo puede ser menor que el conjunto total de las que disponemos.

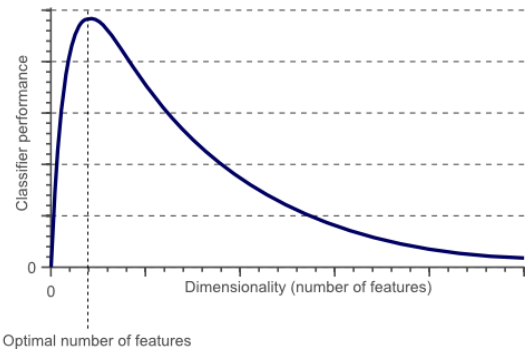
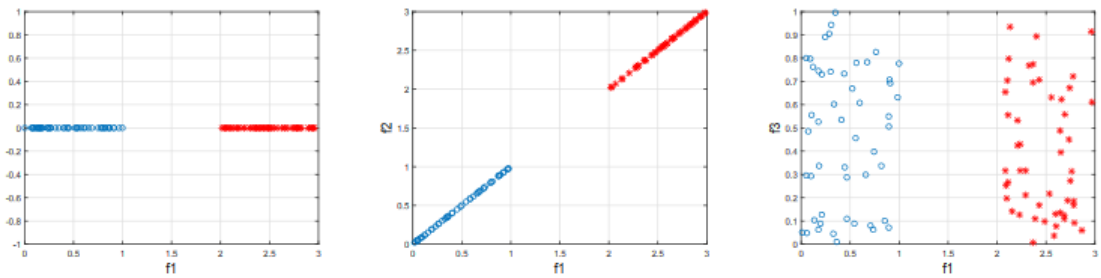


Ilustración 2 - Efecto de la reducción de la dimensionalidad en la clasificación [44]

La selección de características junto con la extracción de características son técnicas para la reducción de la dimensionalidad de conjuntos de datos. Mientras que la extracción de características se basa en la transformación de las características, obteniendo un conjunto menor de características que el original, la selección de características se basa en seleccionar aquellas que sean más relevantes, descartando las características redundantes e irrelevantes. Una de las diferencias entre Las técnicas de extracción y de selección, es en que las segundas mantienen el conjunto de datos original, no lo transforma, por lo tanto, son muy útiles en aplicaciones donde las características originales son necesarias para interpretar el modelo correctamente y, de esta manera, extraer conocimiento. La extracción de características es un método útil para modelos en los que la interpretabilidad del modelo no es tan importante como su exactitud.

La selección de características consiste en la detección de las características relevantes de un conjunto de datos, mientras descarta aquellas que sean irrelevantes o redundantes.



(a) relevant feature f_1 (b) redundant feature f_2 (c) irrelevant feature f_3

Ilustración 3 - Características relevantes, redundantes e irrelevantes [45]

En la ilustración 2, podemos ver en el apartado a, que la característica f_1 es relevante a la hora de discriminar los dos grupos que se observan. Ahora bien, en el apartado b vemos que al añadir la característica f_2, ésta es redundante. Por último, en el apartado c, podemos ver que la característica f_3 es irrelevante frente a f_1.

Una característica se considera redundante cuando sus valores son totalmente correlativos a otra. Se complica el detectar características redundantes cuando una característica es correlativa ante un subconjunto de características del conjunto de datos. Del artículo de Kohavi, R. et al [4], obtenemos una definición de características relevantes. Define dos grados de relevancia con la ayuda de un clasificador bayesiano:

- Característica fuerte. Si se elimina la característica X del conjunto de datos, la predicción del modelo queda deteriorada.
- Característica débil. Si no es una característica fuerte y si existe un subconjunto de características S, al que pertenece esta característica, tal que la predicción del modelo es peor que cuando se utiliza el subconjunto $S \cup \{X\}$.
- Característica irrelevante. Cuando la característica no es ni fuerte ni débil.

Por lo tanto, las características de un conjunto de datos se las puede diferenciar en cuatro grupos:

1. Características fuertes y relevantes.
2. Características débiles, pero no redundantes.
3. Características débiles y redundantes.
4. Características irrelevantes.

Ilustración 4 - Tipos de características

Los métodos de selección de características buscan encontrar aquellas características que pertenezcan a los grupos 1 y 2. Los beneficios de la aplicación de selección de características al conjunto de datos con problemas de dimensionalidad, entre otros, son:

- Modelos más precisos y rápidos, debido a la eliminación de características redundantes e irrelevantes.
- Reducción del espacio de almacenamiento y de búsqueda.
- Mayor comprensión sobre el conjunto de datos.
- Al ser modelos más simples, pueden ayudar a una mejor visualización.
- Reducción en los costes a la hora de recopilar nueva información.

2.1 MÉTODOS DE SELECCIÓN DE CARACTERÍSTICAS

Existen diferentes métodos de selección de características. Primero, se les puede diferenciar entre aquellos que realizan un estudio de cada característica con independencia del resto de características del conjunto de datos, métodos univariantes, y los que sí que tienen en consideración las demás características o subconjunto de éstas, métodos multivariantes. Por otro lado, hay métodos que evalúan a cada una de las características, clasificándolas en un ranking, mientras que otros métodos evalúan subconjuntos de características.

La evaluación individual de características, clasificándolas en un ranking, puede conllevar la no eliminación de características redundantes, por tener estas una misma clasificación en el ranking. Mientras que, la evaluación de subconjuntos de características puede hacer frente a la redundancia, se encuentran con el problema de no poder hacer un estudio exhaustivo de todos los subconjuntos existentes, debido al alto coste computacional que conlleva localizarlos dentro del conjunto de datos.

De manera estructural, los métodos de selección de características se pueden clasificar en:

- Métodos de filtro.

- Métodos *wrapper*.
- Métodos empotrados.
- Métodos híbridos o combinación de métodos.

2.1.1 Métodos de filtro

Los métodos de filtro realizan una selección de características de manera independiente al algoritmo de aprendizaje que será utilizado y se realiza como un paso previo a la utilización del método de aprendizaje a utilizar. Estos métodos realizan una clasificación de las características mediante una función que evalúa cada una de éstas asignándolas un valor. Aquellas características que obtengan una mayor valoración serán más significativas. Posteriormente, las características se ordenan de mayor a menor y serán seleccionadas aquellas que tengan una mayor puntación para ser utilizadas en el método de aprendizaje.

Este tipo de métodos tiene un coste computacional muy bajo, respecto a los demás métodos, además de ser simples y rápidos de calcular. Se caracterizan por tener una gran capacidad de generalización. Encontramos métodos de filtro univariantes y multivariantes, siendo éstos últimos los que más recursos computacionales consumen. Los métodos de filtro son utilizados para la selección de características en aquellos conjuntos de datos que tienen un gran número de características.

2.1.2 Métodos *wrapper*

Este tipo de métodos se basan en la utilización de un algoritmo de aprendizaje que mide la eficacia de las características según el nivel de predicción que se obtiene en éste. Consisten en escoger un subconjunto de características y asignarles una valoración, que se obtienen en función de su capacidad de predicción en el algoritmo de aprendizaje. El resultado será el subconjunto de características que obtenga la mejor evaluación.

Este es un proceso que consume muchos recursos computacionales, principalmente debido a que el algoritmo es aplicado varias veces para cada uno de los subconjuntos. Este tipo de métodos son utilizados cuando los costes computacionales no suponen un problema.

2.1.3 Métodos empotrados

La selección de características en los métodos empotrados se realiza en el propio algoritmo de aprendizaje, dónde la búsqueda de las características más significativas es

guiada por el propio proceso, es decir, la selección de características no es un paso previo al modelo de aprendizaje.

Los métodos empotrados realizan la selección de características en el proceso de entrenamiento del algoritmo de aprendizaje y esta selección es propia y específica de cada algoritmo. Realizan una búsqueda de las características más significativas entre los distintos subconjuntos de características e hipótesis. De este modo, este tipo de métodos localiza dependencias con un menor coste computacional que los métodos *wrapper*.

2.1.4 Métodos híbridos

La combinación en la utilización de métodos de filtro y métodos *wrapper* nos da como resultado los métodos híbridos de selección de características. Este tipo de métodos busca obtener las ventajas de los métodos de filtro y *wrapper*. Por un lado, los métodos de filtro son rápidos y sencillos de calcular, pero no siempre se obtienen unos resultados que satisfagan el modelo. Por otra parte, los métodos *wrapper* ofrecen buenos resultados, pero su coste computacional y temporal es excesivo ante un gran volumen de características, debido a la complejidad que conlleva la selección de subconjuntos de características para su estudio.

Los métodos híbridos obtienen mejores resultados que los métodos de filtro, pero no llegan a ser tan rápidos como éstos. También tenemos que, los métodos híbridos tienen un menor coste computacional comparados con los métodos *wrapper*. Este tipo de métodos suele aplicar primero, al conjunto de características, un método de filtro y al conjunto seleccionado se le aplicará un método *wrapper*. Los métodos híbridos de selección de características ofrecen buenos resultados ante conjuntos de datos que disponen un gran volumen de características.

2.2 APLICACIÓN DE LOS MÉTODOS DE SELECCIÓN DE CARACTERÍSTICAS

La selección de características es aplicada en aquellos conjuntos de datos donde la dimensionalidad supone un problema a la hora de aplicar un método de aprendizaje. Como veremos, el problema de la dimensionalidad puede venir dado tanto por el número de atributos del conjunto de datos (*microarrays* de ADN), como por el número de muestras de la que dispongamos (accesos no autorizados).

Como primer paso para reducir la dimensionalidad, se suele proponer un método de filtro debido a las ventajas que ofrece en tiempos de ejecución frente a otros métodos. Esta primera aplicación puede ser seguida de otros métodos, o bien *wrapper* u otras técnicas de reducción de dimensionalidad como extracción de características, para

obtener un conjunto refinado de atributos a los que luego se le aplicará el modelo de aprendizaje.

En este apartado, veremos ejemplos de cómo aplican la reducción de la dimensionalidad en *microarrays* de ADN, en la detección de accesos no autorizados a redes o en la clasificación de textos. Pero nos encontramos más campos donde la reducción de la dimensionalidad ayuda en los problemas de clasificación como en problemas de clasificación o reconocimiento en conjuntos de datos multimedia (video, imágenes en videos, reconocimiento de textos en videos, …) [40].

2.2.1 Microarrays de ADN

En los conjuntos de datos de expresiones de genes (*microarrays* de ADN) obtener un conjunto de atributos (genes) del menor tamaño posible que discriminen las distintas clases del conjunto de datos, es uno de los campos dónde la aplicación de métodos de selección de características se convierte en un paso fundamental. La reducción del conjunto de atributos permite una mejora en la precisión de la clasificación de las distintas clases que los componen.

En el trabajo [29] se propone un método de filtro basado en la reducción de la redundancia de los atributos mediante dos pasos, dónde primero analizan la relevancia de los atributos mediante la correlación respecto a la clase a la que pertenecen, al igual que la correlación en entre pares de atributos, seguido de un análisis de la redundancia entre atributos en función de los datos obtenidos en el paso anterior. En su estudio, utilizan cuatro de los conjuntos de datos públicos más utilizados para este tipo de análisis: cáncer de colon, leucemia (el que utilizaremos en nuestro trabajo), cáncer de pulmón y cáncer de mama. La reducción de características obtenida con este método mejora los resultados de clasificación en los cuatro conjuntos de datos, comparados con la no utilización de métodos de selección de características o frente al método ReliefF [30].

2.2.2 Accesos no autorizados

La detección de accesos no autorizados a sistemas en una red o a una computadora, es objeto de estudio en el cual se trabaja para poder diferenciar los distintos tipos de acceso y poder identificar cuales son legítimos de aquellos que pretenden realizar algún tipo de ataque. Por lo tanto, la detección de este tipo de accesos se convierte en un problema de clasificación entre los accesos normales y aquellos que pretenden realizar un ataque.

En [31] proponen una serie de métodos para la clasificación del conjunto de datos del KDD Cup 99 [32], este conjunto de datos consta de casi cinco millones de muestras, con

41 características cada una de ellas. El método que proponen, que consiste en una discretización, una reducción de características mediante métodos de filtro y, posteriormente, su clasificación. Viene dado por la intención de mantener los niveles de clasificación mediante una reducción de sus características, ya que otros métodos se basaban en buscar los atributos más significativos, pero perdiendo poder de clasificación. Los métodos de filtro utilizados fueron: *correlation-based Feature Selection* (CFS) [33], INTERACT [34] y *Consistency-based* [35]. El método propuesto reduce el número de atributos utilizados para la clasificación de conexiones, tanto para el caso de clases binarias como multiclase, manteniendo e incluso mejorando (en el caso de clasificación binaria) los resultados que obtuvo el método ganador usando todas las características del conjunto de datos.

2.2.3 Clasificación de textos

El problema de la clasificación de los textos viene dado por la distribución de las clases dentro del conjunto de datos ya que, en conjuntos de datos en los que los textos (muestras) se van añadiendo constantemente, nos encontramos en la situación en la que un perfil (clase) puede tener muy poca representación dentro del conjunto de datos. Esta situación puede provocar una mala clasificación de las clases. Además, hay que añadir la dimensionalidad que nos encontramos en estos conjuntos de datos, donde nuestro conjunto puede representarse por una matriz en la que cada columna (característica) muestra cada una de las palabras o frases que se usan en el texto y cada fila a un texto en concreto, modelo denominado “bag of words” [36].

En el artículo [37] se nos muestra un proceso de dos pasos en la clasificación de textos en dos conjuntos de datos utilizados que son punto de referencia para la comprobación en los modelos de clasificación de textos (Reuters [38] y Ohsumed [39]). Proponen, en un primer paso, la aplicación de un método de filtro para que, en un segundo paso, queden refinados las características utilizadas mediante un método de extracción de características o mediante un método *wrapper* de selección de características. En total utilizan una combinación de cuatro métodos de filtro en el primer paso: chi-cuadrado (*CHI2*), desviación de la distribución de Poisson (*DP*), método discriminativo de selección de características (*DFSS*) y criterio de discriminación relativa (*RDC*). A cada método de filtro le aplica luego un método de extracción o un método *wrapper*. Los métodos de extracción de características utilizados son: análisis de componentes principales (PCA) e indexación semántica latente (LSI); mientras que el método *wrapper* utilizado es el algoritmo genético (GA). Por último, utilizan el método lineal de SVM para comprobar la eficacia que se obtiene con los métodos de reducción de dimensionalidad.

3 CONJUNTO DE DATOS Y MÉTODOS

El conjunto de datos que utilizaremos para el estudio de los distinto métodos de selección de características consiste en los denominados *DNA microarrays* (chip de ADN [5]) de expresiones de genes. Este conjunto de datos nos muestra, para cada ejemplo, el tipo de cáncer al que pertenece y la expresión de genes que tiene asociada. Nos encontramos dos tipos de cáncer en el conjunto de datos:

- AML, leucemia mieloide aguda [6]. Es un tipo de cáncer que se da en adultos, que puede ser curable con un tratamiento adecuado y si es descubierto en un estadio temprano. Este tipo de cáncer se caracteriza por la producción de células anormales en la médula ósea, que entorpecen la formación de glóbulos rojos [fuente Wikipedia].
- ALL, leucemia linfoblástica aguda [7]. Este tipo de cáncer se da a una edad temprana, pero también puede presentarse en adultos. Se debe a la producción de un elevado número de glóbulos blancos inmaduros, los cuales van reemplazando las células en la médula ósea, impidiendo la formación de células sanguíneas saludables [fuente Wikipedia]

En nuestro conjunto de datos tenemos un total de 72 muestras, de los cuales 47 pertenecen al cáncer ALL y el resto al AML. Cada muestra consiste en una expresión genética que consta de 7.129 genes. Por lo tanto, uno de los primeros problemas que encontramos es el pequeño conjunto de muestras que se disponen seguido de la disposición de las clases, la clase que pertenece al cáncer ALL consiste en, aproximadamente, 65% de las muestras, siendo el 35% restante del cáncer AML. El tamaño que tiene cada una de las representaciones de cada clase en el conjunto de datos se puede traducir en una incorrecta clasificación de aquella que tiene un peso menor dentro de ésta.

3.1 MÉTODOS DE VALIDACIÓN

Lo primero que se realizará será la selección de características sobre nuestro conjunto de datos. Le aplicaremos dos métodos de filtro, uno *wrapper* y por último el método híbrido. Una vez realizada la selección de características, el conjunto de datos lo dividiremos mediante el método de validación cruzada *stratified k-fold* [8]. Este método es una versión del método de validación cruzada *k-fold* [9], se caracteriza por dividir el conjunto de datos de manera que haya una misma representación en cada hoja de cada una de las clases. Como resultado, puede que cada hoja disponga de un número distinto de elementos de entrenamiento y test. Haremos una partición de cinco hojas, como se puede ver en la imagen.

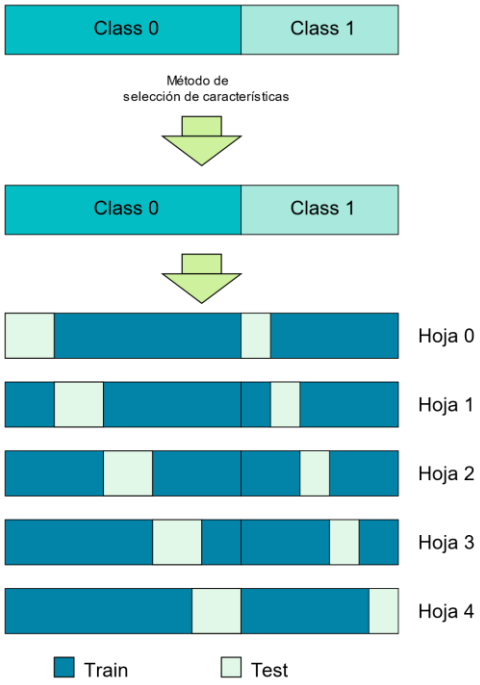


Ilustración 5 - Método de validación I

A cada una de las hojas se le aplicará el método de validación cruzada *k-fold*, dividendo el conjunto de datos, por una parte, el que utilizaremos para el entrenamiento del método y, por otra, para su validación. Posteriormente aplicaremos el método de aprendizaje al conjunto de entrenamiento y mediremos su resultado con el conjunto de *test* en función de las características seleccionadas en cada hoja.

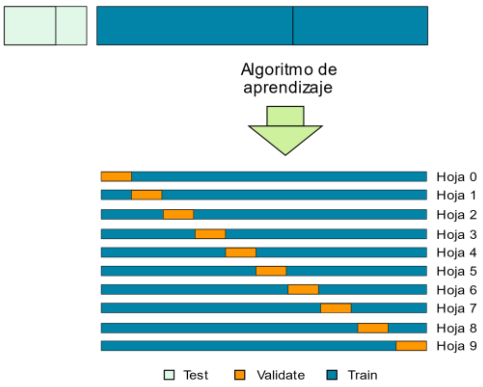


Ilustración 6 - Método de validación II

En cada una de las iteraciones de la validación cruzada *k-fold*, calcularemos la exactitud (*accuracy*), la precisión (*precision*) y la exhaustividad (*recall*) mediante los métodos de aprendizaje *KNeighboursClassifier* [10], *DecisionTreeClassifier* [11] y *SupportVectorClassifier* [12]. Las mejores características las iremos introduciendo una a una, para ver los distintos resultados que obtenemos en función de éstas.

Posteriormente calcularemos la media de las cinco hojas de la validación cruzada *stratified k-fold*, en función del número de características elegidas.

En cada una de las hojas de la validación cruzada *stratified k-fold*, calcularemos la exactitud (*accuracy*) media, entrenando el método con el conjunto de entrenamiento frente al conjunto de *test*.

Como resultado, en función de cada método de selección de características, obtendremos la exactitud media de las iteraciones del *stratified k-fold*, junto a la media de la exactitud, precisión y exhaustividad que se ha calculado en cada iteración de la validación cruzada de cada conjunto de *test*.

3.2 MÉTRICAS

Las medidas que utilizaremos para puntuar los métodos de aprendizaje serán: exactitud, precisión y exhaustividad.

- Exactitud (*accuracy*). Nos mide las predicciones que el modelo ha clasificado correctamente. En nuestro caso, al tratarse de una clasificación binaria, podemos ver la exactitud en función de las clasificaciones positivas verdaderas (*TP*), clasificaciones positivas falsas (*TN*), clasificaciones positivas falsas (*FP*) y clasificaciones negativas falsas (*FN*):

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

- Precisión. Nos mide el número de clasificaciones positivas correctas realizadas. Es decir, la relación entre las clasificaciones positivas verdaderas (*TP*) realizadas en función de las clasificaciones positivas verdaderas (TP) y las clasificaciones positivas falsas (*FN*):

$$Precision = \frac{TP}{TP + FP}$$

- Exhaustividad (*recall*). Mide la proporción del número de clasificaciones positivas identificadas correctamente. Mide la relación entre las clasificaciones positivas verdaderas (*TP*) entre las clasificaciones positivas verdaderas (*TP*) y las clasificaciones negativas falsas (*FN*):

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

3.3 MÉTODOS DE FILTRO

A nuestro conjunto de datos le aplicaremos dos métodos de filtro distintos. Por un lado, se le aplicará un método univariante que jerarquiza los atributos mediante la función *f-score* y, por otro lado, aplicaremos un método multivariante que nos evaluará subconjuntos de atributos, mediante el algoritmo mRMR (*minimun Redundancy Maximum Relevance*).

3.3.1 *f-score*

A cada una de las características se le calculará su *f-score.* Mediante el algoritmo facilitado por la biblioteca *scikit-feature* [13], que utilizando la función *f_score* nos dará como resultado el valor que obtiene cada uno de los atributos del conjunto de datos.

La función *f_score* se basa en la función *f_classif* [14] de la biblioteca *sklearn* [15]. La función *f_classif* hace un análisis de la varianza del Valor-F [16] (*ANOVA F-value*). Este valor consiste en la suma de cuadrados que hay entre cada uno de los grupos, dividido por sus grados de libertad, como numerador y, como denominador, tendremos la suma de cuadrados dentro de cada grupo, dividido por su grado de libertad [17]. Cuanto mayor sea su valor, indicará que existe mayor diferencia entre las dos clases, por tanto, ese atributo tendrá una capacidad mayor de diferenciación entre las clases. Como resultado nos devuelve una lista ordenada de los atributos ordenados de mejor a peor resultado.

Hay que tener en cuenta que, este tipo de función no tiene en cuenta el resto de las características del conjunto de datos, por lo tanto, no realiza una diferencia entre características redundantes. Realizando este método para seleccionar características, podremos estar seleccionando aquellas que realizan una mayor diferencia entre las clases pero que, a su vez, sean similares entre ellas.

Una de las ventajas de este método es su tiempo de ejecución y poca complejidad computacional. El tiempo en calcular el *f-score* en nuestro conjunto de datos es inferior a un segundo.

Podemos ver la exactitud, precisión y exhaustividad de la media de cada hoja del método de validación *stratified k-fold* (*train* y *validate*). En cada una de las gráficas se muestra la media de la exactitud lograda entre las cinco iteraciones respecto al conjunto de *test*.

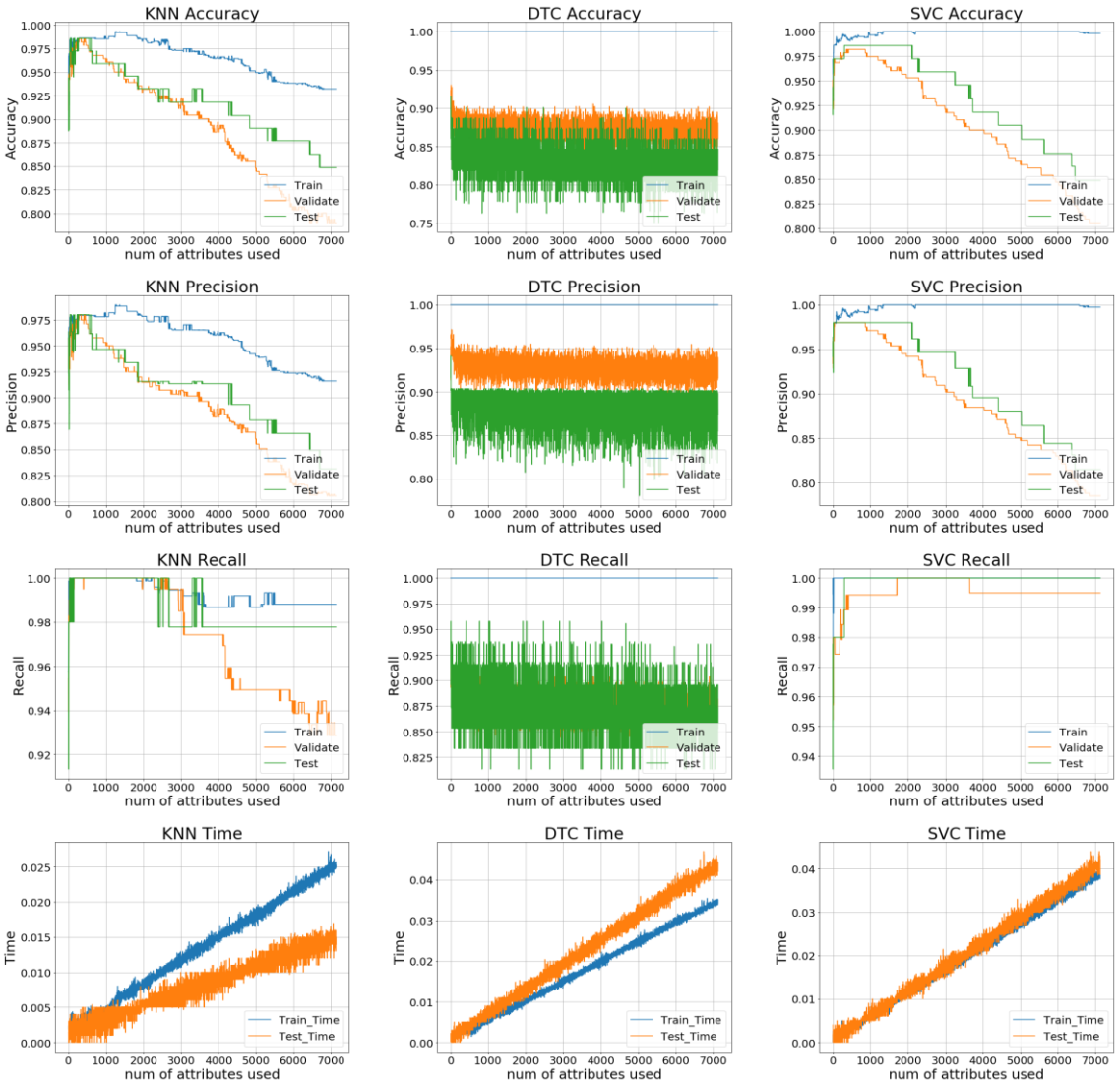


Ilustración 7 - Resultados método f-score

- Resultados

	KNN		DTC		SVC	
	Atributos utilizados	Resultado	Atributos utilizados	Resultado	Atributos utilizados	Resultado
Accuracy	45	98.57%	6	91.52%	305	98.57%
Precision	45	98.00%	6	94.14%	13	98.00%
Recall	27	100.00%	8	95.77%	1	100.00%

Tabla 1 - Resultados f-score

3.3.2 *mRMR*

El método de selección de características mRMR [18] (mínima redundancia, máxima relevancia), busca un subconjunto de características del conjunto de datos que sea capaz de clasificar lo máximo posible las diferentes clases mientras que, las características que lo componen son lo más diferentes posibles entre sí. Es decir, busca un subconjunto de datos que muestre la máxima relevancia, mientras intenta eliminar la mayor redundancia posible.

La aplicación del método se realiza a través de la función *MRMR* de la biblioteca *scikit-feature*, que para su puntuación se basa en la combinación lineal de información mutua [19] y [20], que calcula su puntuación mediante la fórmula:

$$J = I(X_i; Y) - \beta \sum_{j \in S} I(X_j; X_i) + \gamma \sum_{j \in S} I(X_j; X_i | Y)$$

Con un valor de gamma (γ) igual a cero (ya que solo nos interesan los dos primeros parámetros de la función) y un valor de beta (β) que va en función del número de características que se van a seleccionar. En nuestro caso, $\beta = \frac{1}{|S|}$, que nos dará el valor medio de la redundancia del atributo seleccionado en función del número de atributos seleccionado como subconjunto a encontrar. De la fórmula, podemos distinguir:

- Relevancia: $I(X_i; Y)$ que es la información mutua entre una característica X_i y la clase Y.
- Redundancia: $\sum_{j \in S} I(X_j; X_i)$ que es la información mutua entre una característica X_i y el conjunto de características $X_j \in S$, donde S es el subconjunto de características que buscamos.

El tiempo que tarda en ejecutarse este método sobre todo el conjunto de datos supera las cuatro horas de duración. Sin embargo, si seleccionamos un conjunto de datos de quinientos atributos, el tiempo de ejecución se reduce a media hora. Al tratarse de un método multivariante, podemos observar que los tiempos de ejecución son mucho más elevados que los del método anterior.

Al igual que en el apartado anterior, vemos la exactitud, precisión y exhaustividad de la media de cada hoja del método de validación *stratified k-fold* (*train* y *validate*). En cada una de las gráficas se muestra la media de la exactitud lograda entre las cinco iteraciones respecto al conjunto de *test*.

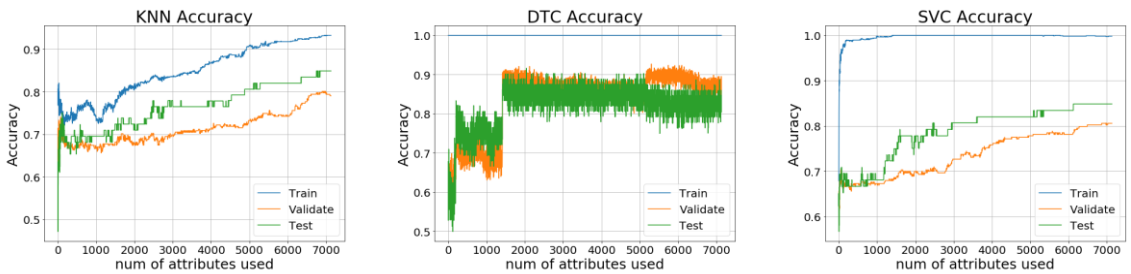


Ilustración 8 - Resultados método mRMR

- Resultados

Seleccionando todos los atributos

	KNN		DTC		SVC	
	Atributos utilizados	resultado	Atributos utilizados	Resultado	Atributos utilizados	Resultado
Accuracy	6749	84.85%	2040	91.61%	6123	84.85%
Precision	6749	83.13%	4875	93.77%	6123	81.49%
Recall	104	100.00%	5164	96.00%	71	100.00%

Seleccionando 500 atributos

	KNN		DTC		SVC	
	Atributos utilizados	Resultado	Atributos utilizados	Resultado	Atributos Utilizados	resultado
Accuracy	114	75.04%	221	84.76%	71	70.85%
Precision	114	72.62%	213	89.64%	62	69.82%
Recall	104	100.00%	289	96.00%	71	100.00%

Tabla 2 - Resultados mRMR

3.4 MÉTODOS *WRAPPER*

Los métodos *wrapper* que utilizaremos se basan en una selección de características secuencial en la que se van añadiendo o eliminando características, en función del método, al subconjunto de características que sea más relevante de nuestro conjunto de datos. Veremos dos versiones, implementadas mediante la biblioteca *mlxtend* [21] en Python: *sequential forward selection* y *sequential backward selection*.

Los métodos empotrados de selección de características se caracterizan por la utilización de un método de aprendizaje para la evaluación de las características y, en función de esta evaluación, la inclusión en el subconjunto de características seleccionadas. En nuestro caso, el algoritmo de aprendizaje que utilizaremos es el *SupportVectorClassifier* [12].

Los algoritmos basados en los *support vector machines* (*SVM*), buscan la maximización de los márgenes de los hiperplanos que utilizan para la separación de las distintas clases, en función de los *kernels*[1] que utilizan. Nosotros utilizaremos el algoritmo facilitado por la biblioteca scikit-learn con los parámetros que vienen por defecto. Por lo tanto, el *kernel* utilizado será el *radial basis kernel function* [22] (RBF *kernel*). Otros dos factores que hay que tener en cuenta, en los parámetros del SVC que viene por defecto, son el valor de *gamma*, el cual nos indica la influencia que puede llegar a tener una muestra, y el valor *C*, que nos indica la flexibilidad que tenemos en nuestro algoritmo a la hora de establecer los márgenes.

3.4.1 *Sequential forward selection*

Implementamos el algoritmo de selección secuencial de atributos hacia delante. Este algoritmo empieza con un subconjunto de atributos vacío, secuencialmente irá introduciendo atributos más significativos de nuestro conjunto de datos hasta que lleguemos al número de atributos total que hayamos indicado como máximo de nuestro subconjunto. La decisión de introducir o no un nuevo atributo vendrá dado por el resultado que haya obtenido en el algoritmo de aprendizaje que se utiliza en el método, el SVC.

La implementación de este método se realiza con la biblioteca *mlxtend* [23] y el algoritmo *Sequential Feature Selector* [24] (SFS), en el cual podemos elegir el algoritmo que se utilizará para la decisión de inclusión de las características en nuestro subconjunto.

Una de las características de los métodos empotrados son el tiempo necesario para su realización debido a los recursos que utiliza en su ejecución. En nuestro caso, para la obtención de un subconjunto de mil atributos ha tardado trece horas en ejecutarse.

Los resultados obtenidos los podemos ver en las gráficas siguientes.

[1] Un *kernel* es una función de proyección entre espacios.

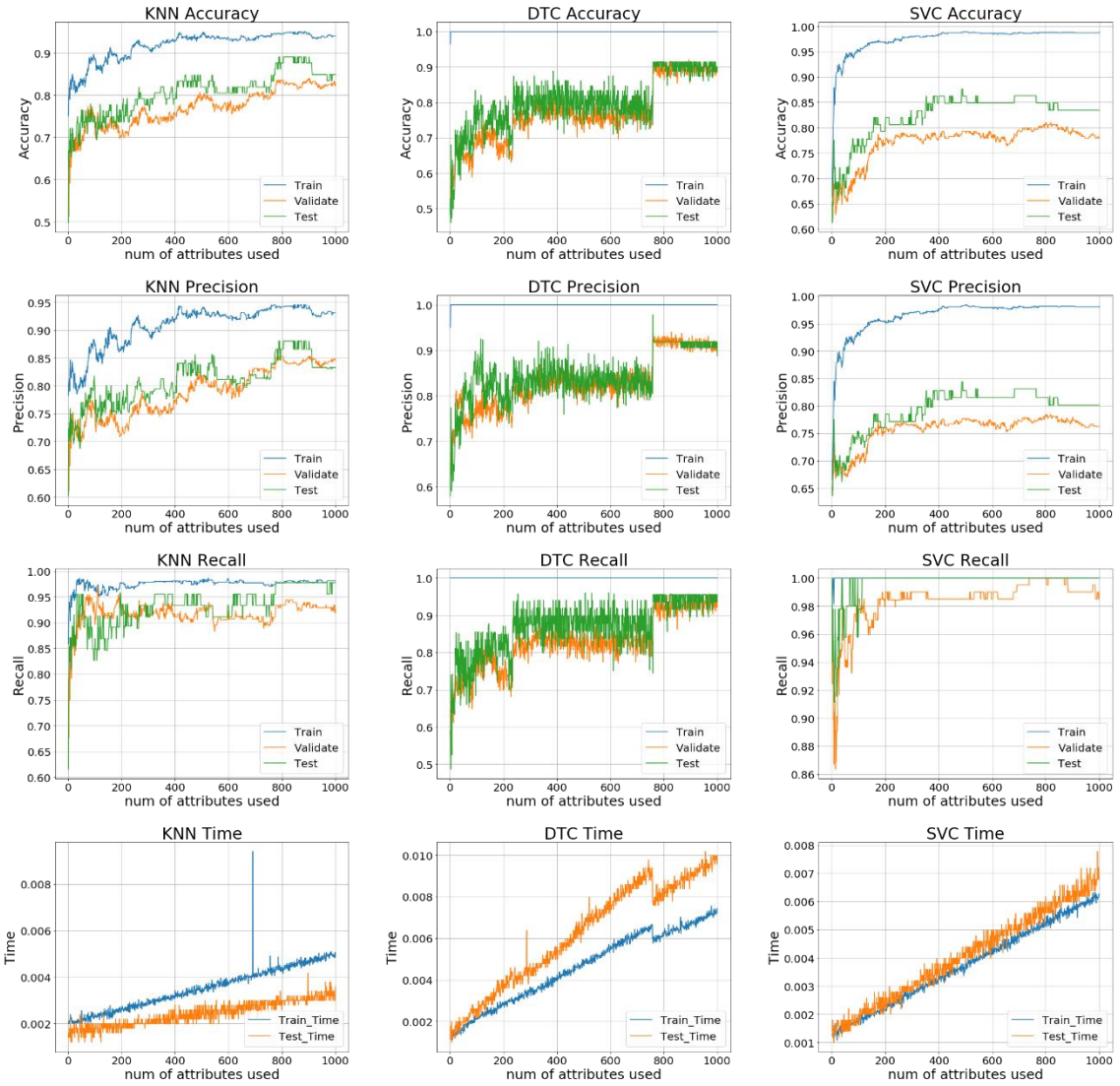


Ilustración 9 - Resultados método SFS_f

- Resultados

	KNN		DTC		SVC	
	Atributos utilizados	resultado	Atributos utilizados	Resultado	Atributos utilizados	Resultado
Accuracy	788	89.04%	760	91.42%	486	87.61%
Precision	788	88.05%	759	97.77%	486	84.41%
Recall	777	97.77%	506	96.00%	1	100.00%

Tabla 3 - Resultados SFS_f

3.4.2 *Sequential backward selection*

El método de selección de atributos hacia atrás se asemeja al SFS en la utilización como algoritmo de selección de características mediante el SVC. Su diferencia es que, en este caso, se empieza con el conjunto completo de nuestros datos y se van eliminando características hasta llegar a un subconjunto de atributos de una cantidad de atributos definido previamente. Los atributos que se van eliminando son aquellos que sean más irrelevantes en función del resultado que hayan obtenido en algoritmo seleccionado para su evaluación.

La *mlxtend* [23] implementa el algoritmo *Sequential Feature Selector* [24] (SFS), visto en el apartado anterior, el cual se le puede indicar si la selección debe ser hacia delante o hacia atrás. Este método no ha sido utilizado en nuestro conjunto de datos debido al tiempo que requiere su ejecución y, por tanto, solamente será utilizado para el refinamiento de atributos que se hará en el método híbrido.

3.5 MÉTODO HÍBRIDO

El método híbrido seleccionado para aplicar en nuestro conjunto de datos es una adaptación del descrito en el artículo [25], en el cual se hace una combinación entre los métodos de selección de características de filtro y los *wrapper*. Por un lado, realiza una primera selección de características mediante las técnicas de filtro, que son más rápidas. Posteriormente, se realiza una mejora de esta selección de características mediante técnicas *wrapper*, para encontrar el mejor subconjunto de las ya seleccionadas en el paso anterior.

Primero empezamos realizando una selección a nuestro conjunto de datos mediante el método de filtro *f-score* y, también, mediante el método *mRMR* seleccionando un conjunto de quinientos atributos. Estos métodos son sencillos y rápidos de calcular, vistos en los apartados anteriores. Los resultados que se obtienen son dos subconjuntos de atributos evaluados de diferente manera.

Ilustración 10 - Aplicación f-score y mRMR en el conjunto de datos

El siguiente paso es diferenciar dos grupos con los subconjuntos que tenemos. Por un lado, seleccionaremos cierto número de atributos que se encuentren en los dos subconjuntos que hemos encontrado. Estos serán atributos que ambos algoritmos han

decidido que son relevantes para nuestro conjunto de datos. Por otro lado, nos quedan los dos primeros subconjuntos de atributos a los que les hemos quitado los que tienen en común. De estos dos subconjuntos, elegiremos los primeros, escogiendo uno a uno de cada grupo, hasta tener un subconjunto del mismo tamaño que el subconjunto de atributos similares.

Por lo tanto, tendremos un subconjunto con los mejores atributos que tanto el algoritmo *f-score* como el *mRMR* han considerado más relevantes y, por otro lado, un subconjunto con los atributos que, al no encontrarse en el grupo anterior, son los mejores evaluados por sus respectivos algoritmos.

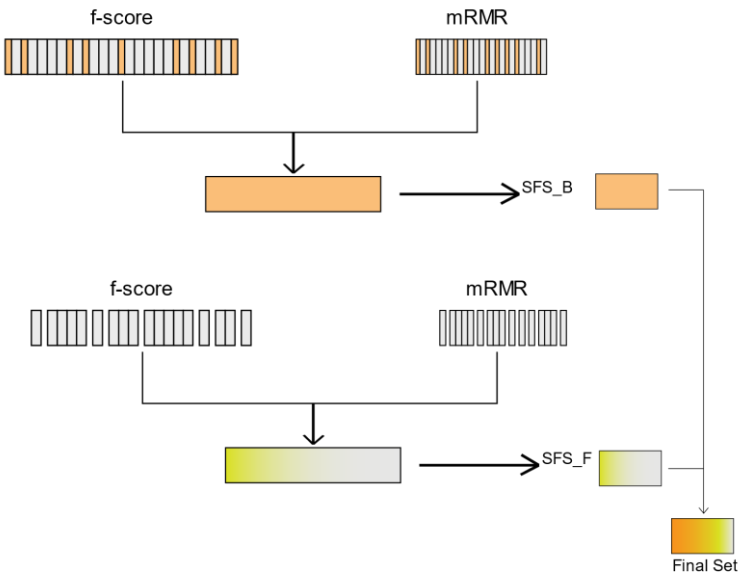


Ilustración 11 - Aplicación SFS_b y SFS_f

Ahora, a cada uno de estos subconjuntos se le hará un refinamiento mediante un método *wrapper* de selección de características. Al subconjunto que contiene los atributos similares se le aplicara el método de selección de características hacia atrás. Se irán eliminando atributos uno a uno hasta encontrar un subconjunto que proporcione los mejores resultados. Mientras, al segundo subconjunto se le aplicará el método de selección de características hacia delante, se irán seleccionando características una a una, hasta encontrar un subconjunto que presente los mejores resultados.

Con nuestro estudio hemos acabado con un subconjunto de treinta y seis atributos. Los resultados que hemos obtenido aplicando nuestros métodos de validación son los siguientes.

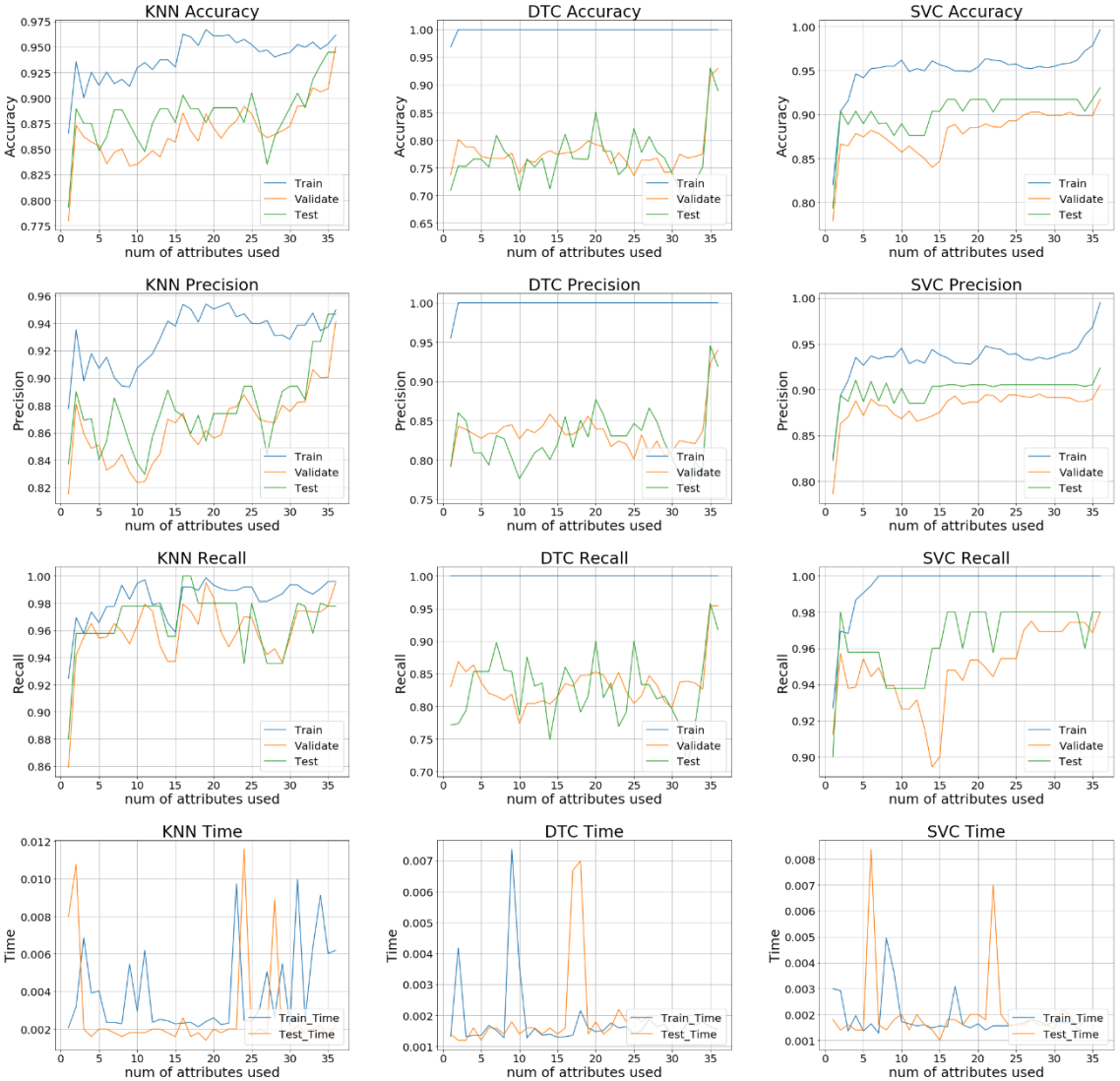


Ilustración 12 - Resultados método híbrido

- Resultados

	KNN		DTC		SVC	
	Atributos utilizados	resultado	Atributos utilizados	Resultado	Atributos utilizados	Resultado
Accuracy	35	94.47%	35	93.04%	36	93.04%
Precision	35	94.46%	35	94.54%	36	92.36%
Recall	16	100.00%	35	95.77%	2	98.00%

Tabla 4 - Resultados método híbrido

4 CONCLUSIONES

Se ha realizado una aplicación de distintos métodos de selección de características a un conjunto de datos que se caracterizan por tener un gran volumen de características, nuestro conjunto dispone de 7.129 características (genes) distintas y, además, de disponer de un número muy pequeño de muestras, 72 muestras. Este tipo de conjunto de datos (DNA *microarrays* [5, 26]) proponen un desafío a los algoritmos de aprendizaje computacional por las complicaciones que se derivan del gran número de atributos y el pequeño tamaño de la muestra. Por lo tanto, una reducción de la dimensionalidad es un paso obligado para poder aplicar estos métodos.

4.1 RESULTADOS OBTENIDOS

En nuestro estudio hemos realizado la reducción de la dimensionalidad mediante tres métodos distintos (filtro, *wrapper* e híbrido) utilizando un total de cuatro algoritmos distintos de selección de características. En la siguiente tabla podemos ver la exactitud que hemos llegado a conseguir con cada uno de nuestros métodos:

	Accuracy					
	KNN		DTC		SVC	
	Atributos utilizados	Resultado	Atributos utilizados	Resultado	Atributos utilizados	Resultado
Filtro: *f-score*	45	**98.57%**	6	91.52%	305	**98.57%**
Filtro: *mRMR*	6749	84.85%	2040	91.61%	6123	84.85%
Filtro: *mRMR* (500)	114	75.04%	221	84.76%	71	70.85%
Wrapper: SVC-SFS	788	89.04%	760	91.42%	486	87.61%
Híbrido	35	94.47%	35	**93.04%**	36	93.04%

Tabla 5 - Accuracy: comparación de resultados entre los distintos métodos

Podemos ver que los resultados varian en función del método de clasificación, al igual que el número de atributos necesarios para alcanzar el mejor resultado. Los mejores resultados los obtenemos mediante el método de filtro *f-score*, los cuales utilizan un conjunto muy pequeño de atributos, obteniendo los mejores resultados en dos de los tres métodos de clasificación utilizados. Hay que tener en cuenta las desventajas que ofrece este método de selección de características, ya que evalúan individualmente los atributos, por lo tanto, podemos encontrar atributos que sean redundantes dentro de los seleccionados.

Por otro lado, métodos que si que realizan una diferenciación entre atributos redundantes muestran unos resultados menores que los que obtenemos con el *f-score*, que es un método que puede incluir características redundantes como aquellas que mejor discriminan las clases en sus resultados.

Por último, nuestro método híbrido mantiene muy buenos resultados frente a los demás métodos, obteniendo el segundo mejor resultado con los algoritmos de clasificación *KNN* y *SVC*, y superando los resultados del método *f-score* con el método *DTC*. Otra observación que se puede resaltar es que el método de filtro *f-score* y el método híbrido son los que menos atributos utilizan para la obtención de los mejores resultados.

También podemos comparar el resultado de nuestro método híbrido con el resultado del método del que nos hemos basado para construirlo [25], al igual que con los métodos que éste se compara.

Método	Accuracy (%)	Atributos utilizados
Fujibuchi and Kato (2007) [41]	97.8	170
Cho and Ryu (2002) [42]	94.1	30
Cho and Won (2007) [43]	97.1	50
Hui-Huang Hsu, Cheng-Wei Hsieh, Ming-Da Lu (2011) [25]	98.6	70
Método híbrido propuesto	94.4	35

Tabla 6 - Comparación de resultados con trabajos originales

Respecto a la precisión y a la exhaustividad obtenida, podemos observar que la precisión va seguida de la exactitud con unos resultados muy similares. Por otra parte, encontramos resultados en las que la exhaustividad llega al 100%, por lo tanto, los resultados obtenidos con ese conjunto de datos podemos indicar que no obtendremos falsos negativos. Esta es una característica que a la hora de clasificar las enfermedades puede resultar un factor para tener en cuenta. Podemos ver la tabla de los distintos resultados y los atributos necesarios para calcular la exhaustividad en nuestro conjunto de datos.

	Recall					
	KNN		DTC		SVC	
	Atributos utilizados	Resultado	Atributos utilizados	Resultado	Atributos utilizados	Resultado
Filtro: *f-score*	27	**100.00%**	8	95.77%	1	**100.00%**
Filtro: *mRMR*	104	**100.00%**	5164	96.00%	71	**100.00%**
Filtro: *mRMR* (500)	104	**100.00%**	289	96.00%	71	**100.00%**
Wrapper: SVC-SFS	777	97.77%	506	96.00%	1	**100.00%**
Híbrido	16	**100.00%**	35	95.77%	2	98.00%

Tabla 7 - Recall: comparación de resultados entre los distintos métodos

Otra característica que hay que tener en cuenta es, que tanto en los algoritmos de clasificación como en los de selección de características, se ha trabajado con los valores que vienen definidos por defecto. Por lo tanto, se pueden obtener resultados distintos en caso de modificar estos valores para ajustarlos mejor al conjunto de datos a estudiar.

4.2 CONCLUSIONES RESPECTO A LOS OBJETIVOS

Visto el análisis de los resultados, podemos observar nuestro método híbrido de selección de características no ha superado los resultados que ofrece el método de filtro *f-score*, pero sí que ha superado a los otros dos métodos estudiados en este trabajo.

Hay que tener en cuenta que, nuestro método híbrido de selección de características es una adaptación del trabajo realizado en [25]. Por lo tanto, los resultados que obtenemos no son los mismos que se demuestran en ese documento. Se ha intentado hacer una aproximación lo más fiel al original, teniendo en cuenta que el paso de refinamiento de selección de características mediante métodos *wrapper* no ha sido el mismo que el seguido en este trabajo. Una de las características que hay que indicar del método híbrido facilitado es que sus resultados pueden variar en función de los conjuntos que se seleccionen para combinar los dos métodos de filtro, ya que los conjuntos de características que se creen pueden variar y los resultados, por tanto, también. En este caso se han presentado unos resultados que ofrecen unos resultados aceptables.

Respecto a los tiempos de ejecución en función de los atributos utilizados, las diferencias son insignificantes ya que para la realización del estudio cada uno de los métodos lo realiza en menos de un segundo, independientemente del conjunto de características seleccionado. Sin embargo, los tiempos de ejecución de los métodos de selección de características si que influyen a la hora de realizar el estudio, ya que en función de cual se escoja, el tiempo en obtener los subconjuntos deseados puede diferenciarse entre unos segundos y varias horas. El método híbrido se realiza en unos tiempos razonables cuando se aplican los métodos *wrapper* debido a que los conjuntos de características ya han sido reducidos anteriormente mediante los métodos de filtro.

4.3 OBSERVACIONES RESPECTO AL CONJUNTO DE DATOS

En nuestro estudio hemos utilizado un conjunto de datos que consistía en únicamente dos clases y los problemas que nos ofrecía solo consistía en el gran volumen de características y el pequeño número de muestras, propios de los *DNA microarrays*, pero no incluyen todos los problemas que éstos ofrecen. Otro tipo de problemas que nos podemos encontrar a la hora de realizar estudios con estos tipos de datos son [27]:

- Clases no balanceadas. En este tipo de situaciones encontramos que una clase tiene una presencia mucho mayor dentro del conjunto de datos que otra.
- Complejidad de los datos.
- *Dataset shift*. Viene dado cuando los conjuntos de datos ya están divididos en muestras de entrenamiento y de prueba, donde algunos atributos de entrenamiento pueden diferenciar claramente las distintas clases del conjunto de datos. Esta situación en los conjuntos de entrenamiento puede dar lugar a una errónea clasificación en las muestras de prueba.

- Datos atípicos.

Por lo tanto, hay que tener en cuenta estos tipos de características que se pueden presentar en los conjuntos de datos que se van a estudiar.

5 BIBLIOGRAFÍA

[1] https://es.wikipedia.org/wiki/Maldici%C3%B3n_de_la_dimensi%C3%B3n (10/2018)

[2] T.R. Golub et al. Molecular classification of cancer: class discovery and class prediction by gene expression monitoring. Science 286:531-537 (1999).

[3] García S. Luengo J. Herrera F. Tutorial on practical tips of the most influential data preprocessing algorithms in data mining. Knowledge-Based Systems 98 (2016) 1-29.

[4] Kohavi, R., John, G.: Wrappers for feature subset selection. Artif. Intell. 97(1), 273–324 (1997).

[5] https://es.wikipedia.org/wiki/Chip_de_ADN (11/2018)

[6] https://es.wikipedia.org/wiki/Leucemia_mieloide_aguda (11/2018)

[7] https://es.wikipedia.org/wiki/Leucemia_linfoide_aguda (11/2018)

[8] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html (11/2018)

[9] https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html (11/2018)

[10] https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html (11/2018)

[11] https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html (11/2018)

[12] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html (11/2018)

[13] https://github.com/jundongl/scikit-feature (11/2018)

[14] https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html (11/2018)

[15] https://scikit-learn.org/stable/index.html (11/2018)

[16] https://es.wikipedia.org/wiki/Valor-F (11/2018)

[17] https://es.khanacademy.org/math/statistics-probability/analysis-of-variance-anova-library/analysis-of-variance-anova/v/anova-3-hypothesis-test-with-f-statistic (11/2018)

[18] Hanchuan Peng, Fuhui Long, and Chris Ding, "Feature selection based on mutual information: criteria of max-dependency, max-relevance, and min-redundancy," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 27, No. 8, pp.1226-1238, 2005. [PDF]

[19] http://philsci-archive.pitt.edu/10911/1/What_is_Shannon_Information.pdf (11/2018)

[20] http://www.scholarpedia.org/article/Mutual_information (11/2018)

[21] http://rasbt.github.io/mlxtend/api_subpackages/mlxtend.feature_selection/#sequentialfeatureselector (12/2018)

[22] https://en.wikipedia.org/wiki/Radial_basis_function_kernel (12/2018)

[23] http://rasbt.github.io/mlxtend/ (12/2018)

[24] http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/ (12/2018)

[25] Hui-Huang Hsu, Cheng-Wei Hsieh, Ming-Da Lu, "Hybrid feature selection by combining filters and wrappers". Expert Systems with Applications 38 (2011) 8144–8150.

[26] https://en.wikipedia.org/wiki/DNA_microarray (12/2018)

[27] V. Bolón-Canedo, N. Sánchez-Maroño, A. Alonso-Betanzos, J.M. Benítez, F. Herrera, "A review of microarray datasets and applied feature selection methods". Information Sciences 282 (2014) 111–135

[28] John E. Grable, Angela C. Lyons, "An introduction to Big Data". Journal of financial service professionals vol.72, nº5 (September 2018) 17-20.

[29] Yu, L. and Liu, H. Redundancy based feature selection for microarray data. In 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 737–742. ACM, 2004.

[30] https://en.wikipedia.org/wiki/Relief_(feature_selection) (12/2018)

[31] Bolon-Canedo, V., Sanchez-Maroño, N. and Alonso-Betanzos, A. Feature selection and classification in multiple class datasets: An application to KDD Cup 99 dataset. Expert Systems with Applications, 38(5):5947–5957, 2011.

[32] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html (12/2018)

[33] Hall, M. A. (1999). Correlation-based feature selection for machine learning. Ph.D. thesis, University of Waikato, Hamilton, New Zealand.

[34] Zhao, Z., & Liu, H. (1991). Searching for interacting features. In Proceedings of international joint conference on artificial intelligence (IJCAI) (pp. 1156–1167).

[35] Dash, M., & Liu, H. (2003). Consistency-based search in feature selection. Artificial Intelligence Journal, 151(1–2), 155–176.

[36] https://en.wikipedia.org/wiki/Bag-of-words_model (12/2018)

[37] Uysal, A. K. (2018). On Two-Stage Feature Selection Methods for Text Classification. IEEE Access, vol. 6 (pp. 43223-43251)

[38] https://archive.ics.uci.edu/ml/datasets/reuters-21578+text+categorization+collection (12/2018)

[39] http://davis.wpi.edu/xmdv/datasets/ohsumed.html (12/2018)

[40] Lee, P. Y., Loh, W. P., Chin, J. F. Feature selection in multimedia: The state-of-the-art review. Image and Vision Computing 67 (2017) 29–42

[41] Fujibuchi, W., & Kato, T. (2007). Classification of heterogeneous microarray data by maximum entropy kernel. BMC Bioinformatics, 8, 267–277.

[42] Cho, S., & Ryu, J. (2002). Classifying gene expression data of cancer using classifier ensemble with mutually exclusive features. Proceedings of the IEEE, 90(11), 1744–1753.

[43] Cho, S., & Won, H. (2007). Cancer classification using ensemble of neural networks with multiple significant gene subsets. Applied Intelligence, 26(3), 243–250.

[44] http://www.visiondummy.com/2014/04/curse-dimensionality-affect-classification/ (12/2018)

[45] http://www.public.asu.edu/~jundongl/tutorial/KDD17/survey.pdf (12/2018)

6 ANEXO I

Junto a la memoria de este trabajo se adjuntan tres archivos en formato *jupyter notebook* y un conjunto de datos en formato *Matlab*. Para poder ejecutarlo correctamente se deberá tener el archivo con el conjunto de datos dentro de la misma carpeta que los archivos *jupiter notebook*.

Primero se tendrá que ejecutar el archivo con el nombre: *TFG_basic_functions.ipynb*, para después ejecutar (sin ningún orden específico):

- Los métodos de filtro y *wrapper*: *TFG_F_W_Methods.ipynb*.
- El método híbrido: *TFG_Hybrid_Method.ipynb*.

Hay que tener en cuenta que la ejecución completa de los métodos de filtro y *wrapper* supera las veinte horas de ejecución en el sistema que se ha utilizado. Podrá variar en función del equipo disponible.

6.1 ERRORES EN LA BIBLIOTECA *SCIKIT-FEATURE*

La biblioteca *scikit-feature* está escrita en Python 2.7, y han hecho una conversión para poder utilizarse en Python 3. Para que funcionen correctamente las funciones se deberá tener en cuenta las correcciones que se hacen en: https://github.com/bacalfa/scikit-feature/commit/f99c6bb5db4dddef233847dc5d4f33a2adac94a1 y hacer las modificaciones oportunas para el funcionamiento correcto.

7 ANEXO II

Empezaremos, este anexo, describiendo las partes comunes para los dos archivos que describen los métodos de filtro, *wrapper* e híbrido. Después explicaremos la funcionalidad del método de filtro *f-score*, explicación que vale también para el método de filtro *mRMR* y para el método *wrapper sequential forward selection*. Por último, veremos la implementación del método híbrido.

7.1 COMÚN PARA LOS TRES MÉTODOS

Empezamos cargando nuestro conjunto de datos. Al estar en un formato *Matlab* debemos convertirlo mediante "*scipy.io.loadmat*" para poder trabajar con él. Posteriormente, diferenciamos las distintas características y muestras, conjunto X, de la clase de cada muestra, conjunto Y.

Load dataset

```
# Load dataset ALLAML
ds = scipy.io.loadmat("ALLAML.mat")
# Features -> X
X = ds['X']
# Class labels -> y
y = ds['Y'][:,0]
# Shape of the data array
n_samples, n_features = np.shape(X)
# Shape of the class labels
n_labels = np.shape(y)
```

Ilustración 13 - Load dataset

Crearemos un *array* con los diferentes métodos de aprendizaje computacional, que llamaremos *models*, para así utilizarlos más fácilmente cuando sea necesario. Al igual que creamos otro *array* de las distintas métricas que utilizaremos.

ML models

```
# Models
clf_KNN = KNeighborsClassifier(n_neighbors=3)
clf_DTC = DecisionTreeClassifier()
clf_SVC = SVC()
models = [clf_KNN,clf_DTC,clf_SVC]
# Scoring
scoring=['accuracy','precision','recall']
```

Ilustración 14 - ML models

Por último, definimos nuestro método de validación cruzada estratificado, en 5 hojas, con aleatoriedad.

cv

```
# cross validation method
skf = StratifiedKFold(n_splits=5,shuffle=True,random_state=32)
```

Ilustración 15 - StratifiedKFold

7.2 COMÚN PARA MÉTODOS DE FILTRO Y *WRAPPER*

Haremos la explicación en función del método de selección de características *f_score*, pero que es válido tanto para el método *mRMR* como para el *sequential forward selection*.

En este punto lo que buscamos son las características ordenadas de mejor a peor. Nuestra función nos devolverá un *array* con el nombre de cada característica donde las primeras posiciones serán las que mejor evaluación obtuviesen en el algoritmo de selección de características.

F_score

```
startTime = time.time()
idx_f_score = select_FS_method(fs_method[0],X,y,num_fea)
endTime = time.time()
timeUsed = timedelta(seconds=endTime-startTime)
print('- Time used in FS f_score method: ' + str(timeUsed))
```

```
- Time used in FS f_score method: 0:00:00.015009
```

Ilustración 16 - FS method

Para ello, hacemos uso de la función *select_FS_method*, que en función del método de selección de características solicitado nos dará el resultado.

```
def select_FS_method(fs_method,X_train,y_train,num_fea):
    if fs_method == 'f_score':
        score = f_score.f_score(X_train,y_train)
        idx = f_score.feature_ranking(score)
    elif fs_method == 'mrmr':
        idx = MRMR.mrmr(X_train,y_train,n_selected_features=num_fea)
    elif fs_method == 'svm_f':
        clf_SVC = SVC()
        svm_f = SFS(clf_SVC,k_features=num_fea,forward=True,floating=False,verbose=1,scoring='accuracy',cv=0,n_jobs=1)
        svm_f = svm_f.fit(X_train,y_train)
        idx = svm_f.k_feature_idx_
    elif fs_method == 'svm_b':
        clf_SVC = SVC()
        svm_f = SFS(clf_SVC,k_features=num_fea,forward=False,floating=False,verbose=1,scoring='accuracy',cv=0,n_jobs=1)
        svm_f = svm_f.fit(X_train,y_train)
        idx = svm_f.k_feature_idx_
    return idx
```

Ilustración 17 - select_FS_method

Una vez que tenemos ordenados las características, buscamos los resultados que se obtienen con cada uno de los algoritmos de aprendizaje, mediante la función *model_result2*. Esta función nos devolverá los resultados para cada algoritmo: *knn, dtc, svc*. También nos devuelve la matriz de confusión que obtenemos en cada una de las aplicaciones del algoritmo en función de las características elegidas, que hemos querido guardar por si queríamos visualizarla.

```
knnResult,dtcResult,svcResult,cMatrix = model_result2(X,y,skf,models,num_fea,scoring,idx_f_score)

        - Time used in KNeighbors 0:29:28.405672
        - Time used in DecisionTr 0:17:07.772784
        - Time used in SVC(C=1.0, 0:29:47.006431
------- Time used in FOLD 0:       1:16:23.184887
        - Time used in KNeighbors 0:29:23.821775
        - Time used in DecisionTr 0:15:18.610123
        - Time used in SVC(C=1.0, 0:29:47.369061
------- Time used in FOLD 1:       1:14:29.800959
        - Time used in KNeighbors 0:30:22.369062
        - Time used in DecisionTr 0:15:05.734127
        - Time used in SVC(C=1.0, 0:30:35.232996
------- Time used in FOLD 2:       1:16:03.341184
        - Time used in KNeighbors 0:30:03.917842
        - Time used in DecisionTr 0:14:33.089130
        - Time used in SVC(C=1.0, 0:31:54.315552
------- Time used in FOLD 3:       1:16:31.322525
        - Time used in KNeighbors 0:30:25.622165
        - Time used in DecisionTr 0:13:07.355622
        - Time used in SVC(C=1.0, 0:30:46.852808
------- Time used in FOLD 4:       1:14:19.840587
```

Ilustración 18 - Resultados: model_result2

La función *model_result2* toma como valores de entrada:

- X, conjunto de datos.
- y, clase asociada a cada muestra.
- *skf*, modelo de validación cruzada estratificado de 5 hojas.
- *models*, *array* con los distintos modelos de aprendizaje computacional.
- *num_fea*, número total de características de nuestro conjunto de datos.
- *scoring*, *array* con las métricas que utilizaremos.
- *idx_f_score*, *array* de las características ordenadas de mejor a peor en función del método de selección de características.

En esta función, empezamos definiendo un *array* para cada método que nos guarde los valores de las distintas métricas y tiempos utilizados.

Seguidamente, aplicamos el método de validación cruzada estratificada de cinco hojas para calcular los resultados de cada hoja. Aquí es donde se nos divide el conjunto de datos en conjunto de entrenamiento y conjunto de test.

Para cada modelo de aprendizaje computacional, le aplicaremos la función de entrenamiento *train_ml_method*, que nos devuelve los valores de *accuracy, precision, recall* y matriz de confusión de cada modelo.

En este punto, debemos explicar la función *train_ml_method*. Esta función, toma como parámetros de entrada:

- *X_train, X_test, y_train, y_test*, que son los conjuntos de entrenamiento y de test, que hemos calculado con el método de validación cruzada estratificada de cinco hojas.

- *num_fea*, número de características de nuestro conjunto de datos.
- *model*, modelo de aprendizaje computacional.
- *scoring*, las distintas métricas para evaluar el modelo.
- *idx*, índice de los mejores atributos seleccionados por el método de selección de características.

En esta función iremos calculando los resultados del modelo elegido en función del número de características. Empezaremos con una, e iremos añadiendo las demás una a una y calculando los valores en función de las mejores características que ha seleccionado nuestro método de selección de características.

Para ello, dividimos el conjunto de entrenamiento en un conjunto de entrenamiento y otro de validación. Tanto el conjunto de entrenamiento, como el de validación, estará formado por las muestras y las características seleccionadas para su iteración.

Seguidamente, utilizamos el modelo de validación cruzada de 5 hojas (**importante: en un principio debía ser de 10 hojas, pero por los tiempos de ejecución se ha reducido a la mitad**). Este método utiliza como parámetro de entrada el conjunto de entrenamiento y es el que nos devuelve los resultados de entrenamiento y validación. Posteriormente, aplicamos el método al conjunto de entrenamiento para evaluar los resultados con el conjunto de test.

Una vez aplicado, tendremos para el modelo seleccionado, la exactitud media del conjunto de entrenamiento para el número de características seleccionado y el método de validación cruzada de 5 hojas, la exactitud media del conjunto de validación para el número de características seleccionado y el método de validación cruzada de 5 hojas y la exactitud que se obtiene en el conjunto de test para el número de características seleccionado. Ocurre lo mismo con la precisión, exhaustividad, tiempos de ejecución y matrices de confusión.

Esta función, *train_ml_method*, nos devuelve un array con todos los resultados por número de características y métrica utilizada en función del algoritmo de aprendizaje seleccionado.

```
def train_ml_method(X_train,X_test,y_train,y_test,num_fea,model,scoring,idx):
    startMl = time.time()
    modelResultAcc = []
    modelResultPrec = []
    modelResultRec = []
    conf_matrix = []
    modelTime = []
    for i in range(1,num_fea+1):
        train_sf = X_train[:,idx[0:i]]
        test_sf = X_test[:,idx[0:i]]
        cv = cross_validate(model,train_sf,y_train,cv=5,scoring=scoring,return_train_score=True)
        start = time.time()
        model.fit(train_sf,y_train)
        # acc,prec,rec
        myPredict = model.predict(test_sf)
        test_acc_score = accuracy_score(y_test,myPredict)
        test_prec_score = precision_score(y_test,myPredict)
        test_rec_score = recall_score(y_test,myPredict)
        test_conf_matrix = confusion_matrix(y_test,myPredict)
        #test_score = model.score(test_sf,y_test)
        end = time.time()
        modelResultAcc.append([cv['train_accuracy'].mean(),cv['test_accuracy'].mean(),test_acc_score])
        modelResultPrec.append([cv['train_precision'].mean(),cv['test_precision'].mean(),test_prec_score])
        modelResultRec.append([cv['train_recall'].mean(),cv['test_recall'].mean(),test_rec_score])
        modelTime.append([cv['fit_time'].mean()+cv['score_time'].mean(),end-start])
        conf_matrix.append(test_conf_matrix)
    endMl = time.time()
    totalTime = timedelta(seconds=endMl-startMl)
    print('         - Time used in ' + str(model)[0:10] + ' ' + str(totalTime))
    return modelResultAcc, modelResultPrec, modelResultRec, modelTime,conf_matrix
```

Ilustración 19 - train_ml_method

Una vez explicada la función *train_ml_method*, podemos volver a la función *model_result2* que estábamos explicando.

Para cada modelo (*knn, dtc, svc*) aplicaremos la función *train_ml_method*, y los resultados de cada métrica y cada método los iremos añadiendo a un *array*. Una vez que hayamos hecho las cinco hojas de la validación cruzada estratificada, nuestra función devolverá en función de cada método, los resultados de exactitud, precisión, exhaustividad y tiempo. Añadimos también los resultados de la matriz de confusión, por si nos llegase a interesar su representación.

```
def model_result2(X,y,crossVal,model,num_fea,scoring,idx):
    modelResult = []
    modelTime = []
    myResult = []
    totalTimeFS = 0

    c_matrix = []

    knn_modelResult_acc = []
    knn_modelResult_prec = []
    knn_modelResult_rec = []
    knn_modelResult_time = []

    dtc_modelResult_acc = []
    dtc_modelResult_prec = []
    dtc_modelResult_rec = []
    dtc_modelResult_time = []

    svc_modelResult_acc = []
    svc_modelResult_prec = []
    svc_modelResult_rec = []
    svc_modelResult_time = []

    k = 0
    for train_index,test_index in crossVal.split(X,y):
        startFS = time.time()
        X_train, X_test = X[train_index],X[test_index]
        y_train, y_test = y[train_index],y[test_index]
        for i in range(0,len(model)):
            acc,prec,rec,modelTime,cMatrix = train_ml_method(X_train,X_test,y_train,y_test,num_fea,model[i],scoring,idx)
            c_matrix.append(cMatrix)
            if i == 0:
                knn_modelResult_acc.append(acc)
                knn_modelResult_prec.append(prec)
                knn_modelResult_rec.append(rec)
                knn_modelResult_time.append(modelTime)
            if i == 1:
                dtc_modelResult_acc.append(acc)
                dtc_modelResult_prec.append(prec)
                dtc_modelResult_rec.append(rec)
                dtc_modelResult_time.append(modelTime)
            if i == 2:
                svc_modelResult_acc.append(acc)
                svc_modelResult_prec.append(prec)
                svc_modelResult_rec.append(rec)
                svc_modelResult_time.append(modelTime)
        endFold = time.time()
        time_fold = timedelta(seconds=endFold-startFS)
        print('------- Time used in FOLD ' + str(k) + ':       '  + str(time_fold))
        k += 1

    knn_results = [knn_modelResult_acc,knn_modelResult_prec,knn_modelResult_rec,knn_modelResult_time]
    dtc_results = [dtc_modelResult_acc,dtc_modelResult_prec,dtc_modelResult_rec,dtc_modelResult_time]
    svc_results = [svc_modelResult_acc,svc_modelResult_prec,svc_modelResult_rec,svc_modelResult_time]
    model_cmatrix = [c_matrix]
    totalTimeFS = timedelta(seconds=totalTimeFS)
    print('- TOTAL TIME IN FS: ' + str(totalTimeFS))
    return knn_results,dtc_results,svc_results,model_cmatrix
```

Ilustración 20 - Función: modelResult2

Ahora bien, tenemos los resultados de cada algoritmo de aprendizaje computacional, en función del número de características. Necesitamos darles forma ya que están todos en un mismo *array*. Para ello, utilizamos la función *formatArray*.

En la función *formatArray* introducimos los valores de un modelo de aprendizaje y, lo primero que hacemos es convertirlos a un *dataFrame* de Pandas, que nos es muy útil para luego su visualización. Tenemos que hacer uso de la función *getResults*.

La función *getResults*, nos devolverá un array que contendrá un *dataFrame* de Pandas de los resultados de exactitud, precisión, exhaustividad y tiempo de ejecución, en función del número de características utilizados.

```
def getResults(myArray):
    myResults = []
    for i in range(0,len(myArray)):
        df_to_return = pd.DataFrame(myArray[i][0])
        for j in range(1,len(myArray[i])):
            df_aux = pd.DataFrame(myArray[i][j])
            df_to_return += df_aux
        df_to_return /= len(myArray[i])
        myResults.append(df_to_return)
    return myResults
```

Ilustración 21 - Función: getResults

Con estos resultados, volvemos a nuestra función *formatArray*, que dará formato a cada *dataFrame*. A los resultados de exactitud, precisión y exhaustividad les dará el título a cada columna correspondientes: *Train, Validate, Test* y, además, definirá el número de cada fila. Para el *dataFrame* de los tiempos de ejecución, también les dará el título de cada columna: *Train_time*, *Test_time* y también definirá el número de cada fila.

```
def formatArray(myArray):
    df = getResults(myArray)
    n_samples,n_features = df[0].shape
    index_list = list(np.arange(1,n_samples+1))
    columnName = ['Train','Validate','Test']
    columnName2 = ['Train_Time','Test_Time']

    for i in range(0,len(df)):
        df[i].index = index_list
        if i!=3:
            df[i].columns = columnName
        else:
            df[i].columns = columnName2
    return df
```

Ilustración 22 - Función: formatArray

Uno de los últimos pasos es el de visualizar los resultados. Tenemos la función *plotModel2*, que toma los valores, con el formato correcto para visualizar, de la función *formatArray*. Esta función nos muestra tres columnas, una por método de aprendizaje, los resultados que se obtienen de la exactitud, precisión, exhaustividad y tiempos de ejecución, que se obtienen en función del número de características utilizadas.

```
def plotModel2(myDf1,myDf2,myDf3,figsize,wspace,hspace,fontsize):

    #fig, axes = plt.subplots(nrows=4, ncols=3, figsize=(15,15))
    fig, axes = plt.subplots(nrows=4, ncols=3, figsize=figsize)

    #KNN
    a0 = myDf1[0].plot(ax=axes[0,0],grid=True, fontsize=fontsize)
    axes[0,0].set_title('KNN Accuracy',fontsize=1.5*fontsize)
    a0.legend(fontsize=fontsize,loc=4)
    a0.set_xlabel('num of attributes used',fontsize=1.25*fontsize)
    a0.set_ylabel('Accuracy',fontsize=1.25*fontsize)

    a1 = myDf1[1].plot(ax=axes[1,0],grid=True, fontsize=fontsize)
    axes[1,0].set_title('KNN Precision',fontsize=1.5*fontsize)
    a1.legend(fontsize=fontsize,loc=4)
    a1.set_xlabel('num of attributes used',fontsize=1.25*fontsize)
    a1.set_ylabel('Precision',fontsize=1.25*fontsize)

    a2 = myDf1[2].plot(ax=axes[2,0],grid=True, fontsize=fontsize)
    axes[2,0].set_title('KNN Recall',fontsize=1.5*fontsize)
    a2.legend(fontsize=fontsize,loc=4)
    a2.set_xlabel('num of attributes used',fontsize=1.25*fontsize)
    a2.set_ylabel('Recall',fontsize=1.25*fontsize)

    a3 = myDf1[3].plot(ax=axes[3,0],grid=True, fontsize=fontsize)
    axes[3,0].set_title('KNN Time',fontsize=1.5*fontsize)
    a3.legend(fontsize=fontsize,loc=4)
    a3.set_xlabel('num of attributes used',fontsize=1.25*fontsize)
    a3.set_ylabel('Time',fontsize=1.25*fontsize)
```

Ilustración 23 - Función: plotModel2

El último paso, es mostrar los mejores resultados. Lo hacemos mediante la función *printTotalBestResult*. Toma como parámetros de entrada los *dataFrame* del paso anterior y muestra, para cada método de aprendizaje, el mejor resultado obtenido en función del número de características usados y la métrica utilizada.

```
def printBestResult(myDf):
    maxAccuracy = myDf[0]['Test'].idxmax()
    maxPrecision = myDf[1]['Test'].idxmax()
    maxRecall = myDf[2]['Test'].idxmax()
    print('- Max Accuracy:   ' + str(myDf[0].loc[maxAccuracy,'Test'])  + ' using: ' + str(maxAccuracy)  + ' attributes.')
    print('- Max Precission: ' + str(myDf[1].loc[maxPrecision,'Test']) + ' using: ' + str(maxPrecision) + ' attributes.')
    print('- Max Recall:     ' + str(myDf[2].loc[maxRecall,'Test'])    + ' using: ' + str(maxRecall)    + ' attributes.')


def printTotalBestResult(KNNdf,DTCdf,SVCdf):
    print('KNN')
    printBestResult(KNNdf)

    print('\nDTC')
    printBestResult(DTCdf)

    print('\nSVC')
    printBestResult(SVCdf)
```

Ilustración 24 - Función: printBestResult

```
printTotalBestResult(fmyKNN,fmyDTC,fmySVC)
```

```
KNN
- Max Accuracy:   0.9857142857142858 using: 45 attributes.
- Max Precission: 0.9800000000000001 using: 45 attributes.
- Max Recall:     1.0 using: 27 attributes.

DTC
- Max Accuracy:   0.9152380952380952 using: 6 attributes.
- Max Precission: 0.9414141414141415 using: 6 attributes.
- Max Recall:     0.9577777777777777 using: 8 attributes.

SVC
- Max Accuracy:   0.9857142857142858 using: 305 attributes.
- Max Precission: 0.9800000000000001 using: 13 attributes.
- Max Recall:     1.0 using: 305 attributes.
```

Ilustración 25 - Resultados método f_score

7.3 APLICACIÓN DEL MÉTODO HÍBRIDO

El método híbrido usa la mayoría de las funciones vistas anteriormente, modificando algunos parámetros.

Primero calculamos las mejores 500 características mediante el método de selección de características *f_score*. También hacemos uso de la función *model_result2* para obtener los resultados y darles forma con la función *formatArray*. Después visualizaremos mediante *plotModel2*.

Filter - f-score

```
startTime = time.time()
idx_f_score = select_FS_method(fs_method[0],X,y,num_fea)
endTime = time.time()
timeUsed = timedelta(seconds=endTime-startTime)
print('- Time used in FS f_score method: ' + str(timeUsed))
```

```
- Time used in FS f_score method: 0:00:00.014290
```

```
knnResult,dtcResult,svcResult,cMatrix = model_result2(X,y,skf,models,500,scoring,idx_f_score)
```

```
         - Time used in KNeighbors 0:00:24.583281
         - Time used in DecisionTr 0:00:13.321155
         - Time used in SVC(C=1.0, 0:00:16.726166
------- Time used in FOLD 0:       0:00:55.007593
         - Time used in KNeighbors 0:00:22.711991
         - Time used in DecisionTr 0:00:13.006940
         - Time used in SVC(C=1.0, 0:00:17.070154
------- Time used in FOLD 1:       0:00:53.168066
         - Time used in KNeighbors 0:00:23.076941
         - Time used in DecisionTr 0:00:12.953666
         - Time used in SVC(C=1.0, 0:00:17.336519
------- Time used in FOLD 2:       0:00:53.790621
         - Time used in KNeighbors 0:00:23.610147
         - Time used in DecisionTr 0:00:12.725068
         - Time used in SVC(C=1.0, 0:00:18.336618
------- Time used in FOLD 3:       0:00:55.072847
         - Time used in KNeighbors 0:00:26.936946
         - Time used in DecisionTr 0:00:14.427194
         - Time used in SVC(C=1.0, 0:00:21.974776
------- Time used in FOLD 4:       0:01:03.959512
```

```
fmyKNN = formatArray(knnResult)
fmyDTC = formatArray(dtcResult)
fmySVC = formatArray(svcResult)
```

```
plotModel2(fmyKNN,fmyDTC,fmySVC,(35,35),0.3,0.3,20)
```

Ilustración 26 - Método híbrido: f_score 01

```
printTotalBestResult(fmyKNN,fmyDTC,fmySVC)
```

```
KNN
- Max Accuracy:   0.9857142857142858 using: 45 attributes.
- Max Precission: 0.9800000000000001 using: 45 attributes.
- Max Recall:     1.0 using: 27 attributes.

DTC
- Max Accuracy:   0.9285714285714286 using: 4 attributes.
- Max Precission: 0.9414141414141415 using: 2 attributes.
- Max Recall:     0.9577777777777777 using: 16 attributes.

SVC
- Max Accuracy:   0.9857142857142858 using: 305 attributes.
- Max Precission: 0.9800000000000001 using: 13 attributes.
- Max Recall:     1.0 using: 305 attributes.
```

Ilustración 27 - Método híbrido: f_score 02

Haremos lo mismo, pero con el método de selección de características *mRMR*

Filter - mRMR

```
# mRMR with 500 features
num_fea_mrmr = 500
startTime = time.time()
idx_mrmr = select_FS_method(fs_method[1],X,y,num_fea_mrmr)
endTime = time.time()
timeUsed = timedelta(seconds=endTime-startTime)
print('- Time used in FS f_score method: ' + str(timeUsed))
```

```
- Time used in FS f_score method: 0:35:40.588381
```

```
knnResult,dtcResult,svcResult,cMatrix = model_result2(X,y,skf,models,num_fea_mrmr,scoring,idx_mrmr)
```

```
         - Time used in KNeighbors 0:00:23.763180
         - Time used in DecisionTr 0:00:16.957090
         - Time used in SVC(C=1.0, 0:00:20.741971
------- Time used in FOLD 0:       0:01:01.882118
         - Time used in KNeighbors 0:00:25.292075
         - Time used in DecisionTr 0:00:18.134435
         - Time used in SVC(C=1.0, 0:00:22.468296
------- Time used in FOLD 1:       0:01:06.357625
         - Time used in KNeighbors 0:00:25.978263
         - Time used in DecisionTr 0:00:16.433923
         - Time used in SVC(C=1.0, 0:00:19.489078
------- Time used in FOLD 2:       0:01:02.347073
         - Time used in KNeighbors 0:00:23.213535
         - Time used in DecisionTr 0:00:16.557642
         - Time used in SVC(C=1.0, 0:00:19.509302
------- Time used in FOLD 3:       0:00:59.734125
         - Time used in KNeighbors 0:00:23.062670
         - Time used in DecisionTr 0:00:15.663587
         - Time used in SVC(C=1.0, 0:00:19.308607
------- Time used in FOLD 4:       0:00:58.387920
```

```
mmyKNN = formatArray(knnResult)
mmyDTC = formatArray(dtcResult)
mmySVC = formatArray(svcResult)
```

```
plotModel2(mmyKNN,mmyDTC,mmySVC,(35,35),0.3,0.3,20)
```

Ilustración 28 - Método híbrido: mRMR 01

```
printTotalBestResult(mmyKNN,mmyDTC,mmySVC)
```

```
KNN
- Max Accuracy:   0.7504761904761905 using: 114 attributes.
- Max Precission: 0.7262737262737262 using: 114 attributes.
- Max Recall:     1.0 using: 104 attributes.

DTC
- Max Accuracy:   0.8476190476190476 using: 221 attributes.
- Max Precission: 0.8964141414141414 using: 213 attributes.
- Max Recall:     0.96 using: 289 attributes.

SVC
- Max Accuracy:   0.7085714285714285 using: 71 attributes.
- Max Precission: 0.6982017982017982 using: 62 attributes.
- Max Recall:     1.0 using: 71 attributes.
```

Ilustración 29 - Método híbrido: mRMR 02

Seguidamente, calcularemos los conjuntos de características que son similares a cada método (*AND*) y los que no coinciden, pero son los mejores evaluados (*XOR*)

Hybrid method

```
idxAND = []
idxXOR = []
idxXOR_f = []
idxXOR_m = []
select_feat = 2500
for i in idx_f_score[:select_feat]:
    if i in idx_mrmr:
        idxAND.append(i)
len(idxAND)
```

```
103
```

```
for i in idx_f_score[:select_feat]:
    if i not in idxAND:
        idxXOR_f.append(i)
for i in idx_mrmr[:select_feat]:
    if i not in idxAND:
        idxXOR_m.append(i)
print(str(len(idxXOR_f)))
print(str(len(idxXOR_m)))
```

```
2397
397
```

```
aux = 0
while len(idxXOR) < len(idxAND):
    idxXOR.append(idxXOR_f[aux])
    idxXOR.append(idxXOR_m[aux])
    aux += 1
len(idxXOR)
```

```
104
```

Ilustración 30 - Método híbrido: AND-XOR

En este punto, empezamos el proceso de refinamiento hacia atrás del conjunto de características comunes de los dos métodos de filtro. Para ello, utilizamos la función *testBackward*. Esta función toma como parámetros de entrada:

- *model*, algoritmo de aprendizaje que se utilizará en el método *wrapper*.
- X, conjunto de datos.
- y, clases del conjunto de datos.
- *idx*, posición de las características mejor evaluadas.
- *limite*, resultado mínimo de exactitud que queremos obtener.

SVC_backward (AND)

```
def testBackward(model,X,y,idx,models,limite):
    num_feat = len(idx)
    parar = False
    while parar==False:
        idx_test,result_test = selectBestBackward2(num_feat,model,X,y,idx,models,limite)
        if result_test >= limite:
            limite = result_test
            num_feat = len(idx_test)
            idx = idx_test
            best_idx = idx_test
            best_result = result_test
        else:
            parar = True
    return best_idx,best_result
```

Ilustración 31 – Función: testBackward

Esta función hace una iteración sobre la función *selectBestBackward2*, que nos devolverá la posición de las mejores características del conjunto de datos y el resultado de exactitud que obtenemos en esta función.

La función *selectBestBackward2*, toma como parámetros de entrada:

- *num_feat*, número total de características.
- *sbs_model,* algoritmo de aprendizaje que se utilizará en el método *wrapper*.
- X, conjunto de datos.
- y, clases del conjunto de datos
- *idx*, posición de las características mejor evauadas.
- *models*, *array* con los algoritmos de aprendizaje computacional (*knn, dtc, svc*)
- *limite*, valor mínimo que buscamos.

```
def selectBestBackward2(num_feat,sbs_model,X,y,idx,models,limite):
    X_train = X[:,idx]
    i = 0
    parar = False
    result2 = 0
    while i < num_feat and parar==False:
        startTime = time.time()
        svm_b = SFS(sbs_model,
            k_features=num_feat-i,
            forward=False,
            floating=True,
            verbose=0,
            scoring='accuracy',
            cv=0,
            n_jobs=1)
        svm_b = svm_b.fit(X_train,y,custom_feature_names=idx)
        idx_b = svm_b.k_feature_names_
        endTimeSFS = time.time()
        timeUsedSFS = timedelta(seconds=endTimeSFS-startTime)
        print('- Time used in SBS iteration ' + str(i) +': ' + str(timeUsedSFS))
        print(str(len(idx_b)))
        X_sbs = X[:,idx_b]
        for model in models:
            cv = cross_validate(model,X_sbs,y,cv=5,return_train_score=True)
            result = cv['test_score'].mean()
            if result > limite:
                parar = True
                result2 = result
        i += 1
    return idx_b,result2
```

Ilustración 32 - Función: selectBestBackward2

Esta función hará uso de la función de la biblioteca *mlxtend SFS* hacia atrás. Se le irán eliminando del conjunto de características una a una, mientras se calcula el mejor subconjunto que nos de el mejor resultado en alguno de los tres métodos de aprendizaje que utilizamos.

Seguidamente, utilizaremos la función *testForward* (similar a *testBackward*) pero sobre el otro conjunto de datos que nos queda. Esta función hace uso de otra función *selectBestForward2* (similar a *selectBestBackward2*), para encontrar el menor subconjunto de características que nos de los mejores resultados.

SVC_forward (XOR)

```
def testForward(model,X,y,idx,models,limite):
    num_feat=len(idx)
    parar = False
    i = 0
    while parar==False and i<120:
        aux = []
        i+=1
        idx_test,result_test = selectBestForward2(num_feat,model,X,y,idx,models,limite)
        if result_test >= limite:
            limite = result_test
            for i in idx:
                if i not in idx_test:
                    aux.append(i)
            idx = aux
            num_feat = len(idx)
            best_idx = idx_test
            best_result = result_test
        else:
            parar = True
    return best_idx,best_result
```

Ilustración 33 - Función: testForward

```
def selectBestForward2(num_feat,sbs_model,X,y,idx,models,limite):
    X_train = X[:,idx]
    i = 1
    parar = False
    result2 = 0
    while i < num_feat and parar==False:
        startTime = time.time()
        svm_b = SFS(sbs_model,
            k_features=i,
            forward=True,
            floating=True,
            verbose=0,
            scoring='accuracy',
            cv=0,
            n_jobs=1)
        svm_b = svm_b.fit(X_train,y,custom_feature_names=idx)
        idx_b = svm_b.k_feature_names_
        endTimeSFS = time.time()
        timeUsedSFS = timedelta(seconds=endTimeSFS-startTime)
        print('- Time used in SFS iteration ' + str(i) +': ' + str(timeUsedSFS))
        print(str(len(idx_b)))
        X_sbs = X[:,idx_b]
        for model in models:
            cv = cross_validate(model,X_sbs,y,cv=5,return_train_score=True)
            result = cv['test_score'].mean()
            if result > limite:
                parar = True
                result2 = result
        i += 1
    return idx_b,result2
```

Ilustración 34 - Función: selectBestForward2

Por último, hacemos la combinación de los dos conjuntos de datos y seguimos los mismos pasos que en los métodos de filtro y *wrapper* para calcular los resultados y mostrar las gráficas.