Prediction of online shoppers' purchasing intention

by

Paola Furtado Rosales

A thesis submitted in conformity with the requirements
for the MSc in Economics, Finance and Computer Science

University of Huelva & International University of Andalusia

December 2021

Predicción de la intención de compra de los usuarios en línea

Paola Furtado Rosales

Máster en Economía, Finanzas y Computación

Diego Marín Santos y Manuel Emilio Gegúndez Arias
Universidad de Huelva y Universidad Internacional de Andalucía

2021

Abstract

Businesses are increasingly relying on online commerce, so this study applies supervised machine learning techniques (trees, random forests and support vector machines), to predict the purchase intention (binary classification problem) of online users.

For this purpose, a database composed of 12.330 sessions of different customers that ended (1.908) or did not end (10.422) with a purchase, described by 18 variables, has been used. The database was divided into training (70%) and evaluation (30%) sets, maintaining the proportion of classes present in the sample. The training set was balanced by random downsampling. Moreover, the models were configured to work with two sets of predictors selected according to two attribute selection techniques: filter-based (*MRMR*) and envelopment methods.

The comparison between models was made by calculating their area under the curve (*AUC*, *ROC* curve). The random forest classifier had the highest performance (*AUC* = 91,87%). The point on the curve with the best compromise between the accuracy in the positive class (in our case, the class "sessions that ended with purchase") and false positives had accuracy rate sensitivity and specificity values of 89,29%, 58,04% and 95% respectively. On the other hand, the curve shows points where the classifier can work, for example, with accuracy rate, sensitivity, and specificity of 88,24%, 78,67% and 90% respectively. The numbers indicate that the classifier can correctly predict the outcome of more than 90% of the sessions that ended without purchase (2.814 sessions out of 3.127, according to our test set), at the cost of being wrong in 21% of those that ended with purchase. In our case, given the imbalance of the classes, this percentage amounts to only 657 instances. Therefore, we conclude that it can be an effective tool to detect customers who do not intend to buy and change their minds through subsequent actions as deemed appropriate.

Key words: purchase intention, online commerce, classification, downsampling, features selection, accuracy, sensitivity, specificity, ROC curve.

Resumen

Cada vez es más frecuente que los negocios se apoyen en el comercio en línea, por ello en este trabajo se aplican técnicas de aprendizaje supervisado (árboles de decisión, bosques aleatorios y máquinas de vector soporte), para predecir la intención de compra (problema de clasificación binaria) de usuarios *online*.

Para ello se ha empleado una base de datos compuesta por 12.330 sesiones de diferentes usuarios que finalizaron (1.908) o no (10.422) con compra, descritas por 18 variables. La base de datos se dividió en entrenamiento (70%) y evaluación (30%), manteniendo la proporción de las clases presentes en la muestra. El conjunto de entrenamiento fue balanceado mediante submuestreo aleatorio. Por otra parte, los modelos fueron configurados para trabajar con dos conjuntos de predictores seleccionados según dos técnicas de selección de atributos: basadas en filtros (*MRMR*) y métodos envolventes.

La comparación entre modelos se realizó calculando el área bajo la curva (*AUC,* curva *ROC*). El clasificador bosque aleatorio tuvo el mayor rendimiento (*AUC* = 91,87%). El punto de la curva que mejor compromiso tiene entre el acierto en la clase positiva (sesiones que acabaron con compra) y los falsos positivos, presentaba valores de tasa de acierto, sensibilidad y especificidad del 86,05%, 85,49% y 86,15% respectivamente. Por otra parte, la curva presenta puntos en los que el clasificador puede trabajar, por ejemplo, con tasa de aciertos, sensibilidad y especificidad del 88,24%, 78,67% y 89,99%, respectivamente. Estos números indican que el clasificador es capaz de predecir correctamente el desenlace aproximadamente del 90% de las sesiones que finalizaron sin compra, a costa de equivocarse en torno al 21% de las que finalizaron con compra.

Se concluye que puede ser una herramienta efectiva para detectar a los clientes que no tienen intención de comprar y hacerles cambiar de opinión mediante las acciones posteriores que se estimen oportunas.

Agradecimientos

A mi familia, por el apoyo incondicional.

A mis tutores, Diego Marín Santos y Manuel Emilio Gegúndez Arias, por su dedicación y entrega durante la realización del presente trabajo.

A todos los profesores del máster, por su profesionalidad y su empeño a la hora de transmitir los conocimientos.

Tabla de contenidos

Lista de tablas

Lista de figuras

1 Propuesta del trabajo

1.1 Motivación

El crecimiento en el uso del comercio en línea ha despertado mi interés por conocer la predicción de la intención de compra de los usuarios *online*, debido a que el número de conversiones de compras no crece al mismo nivel que el uso del comercio electrónico. El marketing digital se ha vuelto fundamental, ya que podemos disponer de tanta información de los usuarios, que sería conveniente conocer si va a abandonar el sitio web sin realizar la conversión, antes de que esto ocurra. De forma que, para estos usuarios que no tienen intención de comprar, se puedan tomar acciones en consecuencia que les retenga y les haga cambiar de opinión incitándole a la conversión.

Por todo esto, vamos a aplicar técnicas de aprendizaje supervisado para conocer la intención de compra del usuario *online*.

1.2 Objetivos

Actualmente estamos en un momento en el que cada vez es más frecuente que los negocios se apoyen en el comercio *online* o en línea. Pero, lamentablemente, el número de conversiones de compras no crece al mismo nivel. Por esta razón, el objetivo de este estudio es aplicar técnicas de aprendizaje supervisado para predecir la intención de compra de los consumidores *online*. Es decir, vamos a crear un sistema de predicción de la intención de compra del usuario *online*.

1.3 Organización de la memoria

La memoria presente se organiza en los siguientes aspectos. Comenzaremos explicando los fundamentos teóricos en los que sustenta el estudio, es decir, que es Machine Learning y los tipos de aprendizaje automático, centrándonos en el aprendizaje supervisado, concretamente en la clasificación. Explicaremos las técnicas de selección de predictores que vamos a aplicar al conjunto de datos, así como las técnicas de clasificación utilizadas (máquina de vector soporte, árbol de decisión y bosque aleatorio), y las métricas de rendimiento para evaluar los distintos modelos (curva ROC y métricas de clasificación binaria: tasa de acierto, sensibilidad y especificidad).

A continuación, haremos una pequeña introducción del problema y una revisión de los antecedentes. Mostraremos la base de datos utilizada para la predicción de la intención de compra *online*, y realizaremos un exhaustivo análisis de las variables que la componen.

Una vez realizado esto, seleccionaremos el conjunto de entrenamiento: para ello dividiremos el conjunto de datos en 70% entrenamiento y 30% validación, de forma que se mantenga la misma proporción de las clases. Y, como veremos en el análisis de los datos, existe un gran desbalance de las clases, siendo necesario aplicar técnicas para balancear los datos (submuestreo aleatorio).

Evaluaremos los modelos, siendo la métrica de referencia el área bajo la curva (*AUC*), y siendo el mejor modelo aquel cuya *AUC* sea mayor. Para este modelo, obtendremos las métricas de clasificación binaria tanto en el punto operativo óptimo de la curva (umbral óptimo), como en otros puntos objetivos en los que el clasificador puede trabajar, por ejemplo, para niveles de especificidad de 65-70-75-80-85-90-95.

Y, por último, haremos una pequeña conclusión de los resultados obtenidos.

El análisis de los datos se realizará a través del software Stata[1] y del lenguaje de programación Python[2]. La estandarización, la división del conjunto de datos, el submuestreo, las técnicas de selección de predictores, el entrenamiento y la evaluación de los modelos se realizará con Matlab[3].

2 Fundamentos teóricos

2.1 Machine Learning

Si bien existen varias definiciones de Machine Learning (*ML*), o aprendizaje automático, podemos definirla como una rama de la Inteligencia Artificial (IA) que consiste en crear sistemas capaces de aprender a partir de datos y algoritmos para imitar la forma en la que aprenden los seres humanos.

---

[1] Stata 16.1

[2] Python 3.9.9

[3] Matlab R2021b

También es conocido como reconocimiento de patrones ya que la máquina no se programa para que responda de una determinada forma según las entradas recibidas, sino más bien para que extraiga patrones de comportamiento a partir de esas entradas, y en base a dicha información aprendida, realice la evaluación de nuevas entradas.

Dentro de la familia de algoritmos de Machine Learning encontramos diferentes técnicas como máquinas de vector soporte (*SVM*), árboles de decisión (*Tree*), bosque aleatorio (*Random Forest*), tablas de decisión, redes neuronales, K-vecino más cercano (*KNN*), regresión lineal, regresión logística, entre otros. Algunas técnicas dan solución a problemas de regresión y otros a problemas de clasificación.

En base a las entradas recibidas existen tres tipos de Machine Learning:

- Aprendizaje automático supervisado. Permite realizar predicciones futuras basadas en comportamientos o características de datos históricos o de entrada etiquetados.
- Aprendizaje automático no supervisado. Permite realizar predicciones futuras basadas en comportamientos o características de datos históricos o de entrada no etiquetados. Esta es la principal diferencia respecto al aprendizaje supervisado, si la variable de salida está etiquetada o por el contrario no lo está.
- Aprendizaje automático semi-supervisado. Este aprendizaje se lleva a cabo cuando solo se ha etiquetado una parte de los datos de entrada dados.

2.2 Aprendizaje automático supervisado

En aprendizaje supervisado se entrena al sistema proporcionándoles unos datos históricos (o de entrada) etiquetados, y permite realizar predicciones futuras basadas en comportamientos o características de esos datos. Durante el entrenamiento, el algoritmo buscará patrones en los datos que se correlacionen con los resultados deseados. Tras el entrenamiento, el algoritmo tomará nuevas entradas y determinará su etiqueta según los datos de entrenamiento anteriores, por tanto, el objetivo de un modelo de aprendizaje supervisado es predecir la etiqueta correcta para nuevos datos.

Dependiendo de cómo sea nuestra variable de salida, estaremos ante un problema de clasificación o de regresión, de forma que, si el resultado deseado consiste en predecir un valor

continuo (como ventas, precio, etc.), estamos ante un problema de regresión. Sin embargo, si la variable de salida es de tipo categórica o nominal (si / no; spam / bandeja de entrada) estamos ante un problema de clasificación y, por tanto, el algoritmo a utilizar dependerá del tipo que sea, ya que hay algoritmos que no se pueden emplear, por su naturaleza, con variables categóricas o nominales, permitiendo solo numéricas.

Nosotros llevaremos a cabo un problema de clasificación (binaria), por lo que es en lo que nos centraremos a explicar.

2.2.1 Clasificación

Los problemas de clasificación como forma de aprendizaje supervisado tienen como objetivo predecir valores categóricos o nominales (Y) a partir de datos históricos o de entrada etiquetados (X) de tipo binario, continuo o categóricos e indistintamente. Es decir, asumiendo un problema en el que se observa que la variable de salida Y (la que queremos predecir) tiene relación con un conjunto de variables X = X1, X2, …, Xp. con p variables de entrada, nos interesa saber esa relación que existe, la cual se puede modelar a través de una función f.

$$Y = f(X) + \varepsilon$$

En general, esta función f es desconocida y se debe estimar basándose en las observaciones de entrada-salida disponibles. Por tanto, nuestro objetivo es encontrar una función f que permita predecir Y con un error mínimo.

Estas variables de entrada pueden encontrarse con diferentes denominaciones como variables independientes, atributos, características, predictores o descriptores. Así como a la variable de salida se puede hacer referencia, también, como variable de respuesta.

Hemos comentado anteriormente que el objetivo de la clasificación es predecir las etiquetas categóricas de nuevos registros, en base a unas observaciones pasadas. Dependiendo de los valores que pueda tomar las etiquetas, estaremos ante un problema de clasificación binaria o multiclase. Si la variable de salida presenta dos valores, es decir, dos clases (por ejemplo: comprar / alquilar) se trata de una clasificación binaria, y si existen más de dos clases (por ejemplo: tren / coche / avión), como su nombre indica, se trata de una clasificación multiclase.

2.3 Selección de predictores

El uso de demasiadas características puede degradar el rendimiento de la predicción incluso cuando todas son relevantes y contienen información sobre la variable de respuesta (Mathworks Inc, 2021). Por esta razón, es conveniente seleccionar el mejor vector de predictores antes de entrenar y evaluar los algoritmos o técnicas de Machine Learning.

Los principales beneficios que aporta la selección de atributos son mejorar el rendimiento de la predicción, proporcionar predictores más rápidos, rentables y relevantes para la variable de salida, proporcionar una mejor comprensión del proceso de generación de datos (Mathworks Inc, 2021), eliminar ruido que sea introducido por características no deseables (Calvario, KC, 2019), acelerar el proceso de aprendizaje y mejorar la interpretabilidad del modelo.

Existen diferentes técnicas de selección de atributos: selección de rasgos de tipo filtro, selección de características de tipo envolvente, selección de características de tipo incrustado, entre otras. En esta sección se muestran los métodos de selección de características utilizados en este estudio. Los método utilizados son algoritmos basados en filtro y métodos envolventes. Estos algoritmos buscan un subconjunto de predictores que modele óptimamente las respuestas medidas (Mathworks Inc, 2021).

El algoritmo de selección de rasgos de tipo filtro mide la importancia de los predictores basándose en las características de los mismos, como la varianza y la relevancia de estos para la respuesta. Se utilizan generalmente como un paso de preprocesamiento de datos, y luego se entrena un modelo utilizando las características seleccionadas.

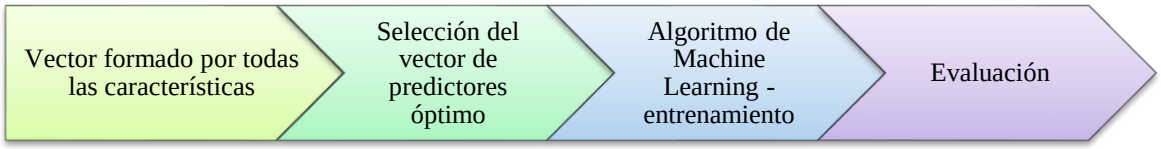


Figura 1. Selección de características basadas en filtros. Fuente: Elaboración propia

Primero se seleccionan las características usando medidas como dependencia o información y, posteriormente, el algoritmo aprende del entrenamiento de los datos con el mejor subconjunto de características obtenido y probando sobre los datos de prueba.

Además, podemos observar que la selección de características de tipo filtro no está relacionada con el algoritmo de entrenamiento, es decir, es independiente a cualquier algoritmo de Machine Learning.

El algoritmo basado en filtro empleado es el de *Minimum Redundancy Maximum Relevance (MRMR)*. Encuentra un conjunto óptimo de características que son mutua, máximamente diferentes y que pueden representar la variable de salida con eficacia. Este algoritmo minimiza la redundancia de un conjunto de características y maximiza la relevancia de un conjunto de características para la variable de respuesta a través de la información mutua de las variables: información mutua por pares de características e información mutua de una característica y la respuesta (Mathworks Inc, 2021).

La otra técnica empelada es la selección de características de tipo envolvente. Esta técnica, a diferencia del basado en filtros, si está relacionada con el algoritmo de entrenamiento. Además, utiliza su rendimiento como criterio de evaluación.

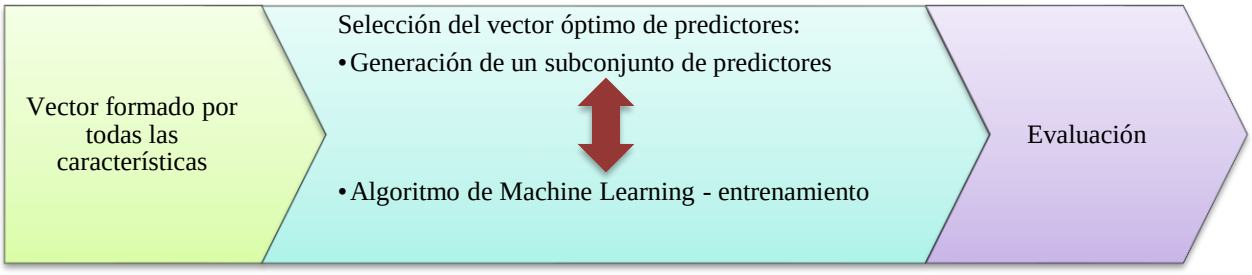


Figura 2. Selección de características métodos envolventes. Fuente: Elaboración propia

Dentro de los métodos envolventes, emplearemos la búsqueda escalonada hacia adelante o *Wrapper*. Como hemos comentado, es una técnica vinculada a una determinada técnica de clasificación, ya que encuentra el vector de predictores que proporciona el mínimo error de la técnica en un conjunto de validación.

En ambos algoritmos, MRMR y *Wrapper*, la estrategia de entrenamiento-evaluación que se aplica es *K-Fold cross validation* con 10 particiones.

En el primero de ellos, se evalúa, por orden de ranking de atributos, los posibles vectores de predictores de diferentes dimensiones. Es decir, para obtener la tasa de acierto del primer vector de predictores de dimensión uno, éste estaría formado por la variable que mejor puntuación ha obtenido. Para obtener la tasa de acierto del vector de predictores de dimensión dos, utilizamos los dos primeros predictores mejores clasificados. Y así hasta incluir todos predictores.

En el segundo de ello, se busca el vector de atributos óptimo de cada dimensión a través de la búsqueda escalonada hacia delante, según máximo valor de tasa de acierto o *Acc*. Para ello, partiendo del vector de predictores sin un orden establecido, se busca el predictor que, individualmente, reporte mayor tasa de acierto. Éste sería el vector de predictor de dimensión uno. Para construir el vector de predictores de dimensión dos, se busca el siguiente predictor que,

juntamente con el seleccionado anteriormente, reporte mayor tasa de acierto. Y así hasta incluir todos los predictores también.

2.4 Técnicas de clasificación

Para el trabajo propuesto, vamos a utilizar tres técnicas de clasificación: máquina de vector soporte con los tres *kernels*: *lineal*, *gaussiano o radial basis function (RBF)* y *polinomial*, árbol de decisión (*Tree*) y bosque aleatorio (*Random Forest*).

Para cada técnica vamos a entrenar y evaluar dos modelos: 1) utilizando el vector de atributos óptimo obtenido de la técnica 1, y 2) el vector de predictores óptimo obtenido con la técnica 2. Por lo que, en total, entrenaremos y evaluaremos diez modelos.

A continuación, vamos a explicar brevemente en qué consiste cada clasificador que vamos a emplear.

Máquinas de vector soporte

Las máquinas de vector soporte (*SVM*, del inglés *Support Vector Machine*) fueron desarrolladas para resolver problemas de clasificación binaria, pero, actualmente, su aplicación se ha extendido a problemas de regresión y multi-clasificación (Rodrigo, 2017).

Se basan en el concepto del hiperplano, es decir, utiliza separadores lineales ya sea en el espacio original de las entradas (si son linealmente separables o cuasi-separables debido a ruido) o en un espacio transformado (si no es linealmente separables). Como veremos más adelante, la búsqueda del hiperplano de separación cuando el espacio es transformado se hará con las funciones *kernels*: *lineal*, *gaussiana* y *polinomial* (Carmona, EJ, 2016).

Se selecciona el hiperplano de separación que equidista de los ejemplos más cercanos de cada clase para conseguir el margen máximo a cada lado del hiperplano, y se consideran los ejemplos de entrenamiento de cada clase que caen justo en la frontera del margen. Estos reciben el nombre de vectores soportes. Resumidamente, los vectores soportes son los puntos que definen el margen máximo de separación del hiperplano que separa las clases (Carmona, EJ, 2016) y se encuentran equidistantes respecto al *maximal margin hyperplane* o hiperplano óptimo de separación. Cualquier modificación que se produzca en estos vectores o puntos conlleva cambios en el hiperplano óptimo de separación (Rodrigo, 2017)

Como hemos comentado anteriormente, puede ser, y es bastante frecuente, que el espacio de las entradas sea cuasi-separable debido a que los datos tienen ruido. Cuando ocurre esto, le indicamos a SVM que generalice bien para la mayoría de los casos. Para controlar esta cantidad de regularización usamos el hiper-parámetro C, el cual controla el número y severidad de las violaciones del margen (y del hiperplano) que se toleran en el proceso de ajuste (Rodrigo, 2017).

Cuando el espacio no es linealmente separable, no podemos encontrar un hiperplano adecuado. Para ello debemos utilizar las funciones *kernels*. Consiste en crear una dimensión nueva con la que podamos encontrar un hiperplano de separación, es decir, el hecho de que los grupos no sean linealmente separables en el espacio original no significa que no lo sea en un espacio de mayores dimensiones.

Árbol de decisión

Árbol de decisión o *Tree* es una técnica de aprendizaje automático supervisada basada en la estratificación o segmentación del espacio de predictores en varias regiones simples utilizando el conjunto de datos de entrenamiento. A cada región se le asocia una respuesta, que suele calcularse de una forma u otra según estemos ante un problema de clasificación o de regresión. Para el caso de regresión esa respuesta asociada corresponde a la media, y en problemas de clasificación, la moda de la variable a explicar cómo valor de predicción, es decir, la clase más frecuente del nodo (Rodrigo, AJ, 2020). La predicción de una instancia de test determinada es la respuesta asociada con la región del espacio particionado donde se posiciona.

El proceso de construcción del árbol de decisión podemos dividirlo en dos etapas: la primera que consiste en la división del espacio de predictores, dando lugar a regiones no solapantes denominadas nodos terminales o finales, y la segunda que es la predicción de la variable a explicar en cada región. Para decidir las divisiones se recurre a la división binaria recursiva (*recursive binaria splitting*) (Rodrigo, AJ, 2020).

¿En qué consiste la división binaria recursiva? Se comienza considerando que todas las observaciones pertenecen a una sola región y se divide sucesivamente el espacio predictor, de forma que, cada división, origina dos nuevas ramas más hacia abajo en el árbol. Es un enfoque voraz ya que, en cada paso del proceso de construcción del árbol, la mejor división se realiza en

ese paso en particular, sin mirar hacia delante y sin elegir la división que conduce a un mejor árbol en un paso futuro.

Tras la creación del árbol, y para evitar el sobreajuste porque tiene muchas ramas que hacen compleja su interpretación y manejo, es necesario realizar una poda de este, que consiste en cortar sucesivamente las ramas y nodos terminales hasta lograr un tamaño adecuado para dicho árbol (Medina & Ñique, 2017). Las observaciones de entrenamiento quedan agrupadas en los nodos terminales, de forma que, para predecir una nueva observación, se recorre el árbol en función del valor de sus predictores (moda) hasta llegar a uno de los nodos terminales (Rodrigo, AJ, 2020).

Este método es simple y útil para la interpretación. Sin embargo, es menos competitivo, en términos de precisión de sus predicciones, que otros enfoques de aprendizaje supervisado. La combinación de una gran cantidad de árboles puede resultar en grandes mejoras de precisión (a costa de una pérdida de interpretación).

Bosque aleatorio

Bosque Aleatorio o *Random Forest* es una técnica de aprendizaje automático supervisada basada en la generación de árboles de decisión, haciendo remuestreo (*bootstrap*) de los datos de entrenamiento, y, por otro lado, muestreo de las variables de entrada en cada nodo del árbol.

Su principal ventaja es que obtiene un mejor rendimiento de generalización al compensar los errores de las predicciones de los distintos árboles de decisión. Para asegurarnos de que los árboles son distintos, lo que se hace es que cada uno de ellos se entrena con una muestra aleatoria de los datos de entrenamiento (Heras, JM, 2020).

Tal y como hemos comentado en la anteriormente en la definición, realiza un muestreo de las variables de entrada, siendo esta otra de las ventajas que presenta. Es capaz de manejar miles de variables de entrada e identificar las variables más significativas, considerándose uno de los métodos de reducción de dimensionalidad (González, L, 2018).

Random Forest utiliza un conjunto de árboles de decisión y cada árbol da una clasificación, es decir, un valor de la variable respuesta, y el bosque elige la clasificación con más votos. Por tanto, cuanto más árboles haya, más robusto será (González, L, 2018).

Además, hay que destacar también que los árboles de decisión individuales tienden a sobreajustarse (Medina & Ñique, 2017). Al combinar los resultados de muchos árboles de decisión, reducimos estos efectos de sobreajuste y mejoramos la generalización.

2.5 Métricas de rendimiento

Curvas ROC

La curva ROC (*Receiver Operating Characteristic* o característica operativa del receptor) constituye una herramienta importante para evaluar el rendimiento de un modelo de Machine Learning. Por lo general, se utilizan en problemas de clasificación binaria, esto es, con dos clases de salidas posibles. Presenta la sensibilidad (verdaderos positivos, *TP*) de una prueba que produce resultados continuos en función de los falsos positivos (*FP* o complementario de la especificidad). Para que un clasificador sea perfecto debe tener una sensibilidad de 1 y una *FP* de 0.

Al hacer la predicción realmente no genera solo 0 o 1, si no un valor continuo en el intervalo [0,1]. Se establece un umbral entre estos valores, y aquellos que superen o sean iguales a dicho umbral pertenecen a una clase.

El área bajo la curva (AUC, del inglés *Area Under Curve*), puede interpretarse como la probabilidad de que, ante un par de individuos, uno pertenezca a una clase y el otro a otra y la prueba los clasifique correctamente. Es conveniente por su invariabilidad con respecto a la escala y con respecto al umbral óptimo. Además, es nuestra métrica de referencia ya que es capaz de medir el rendimiento global de los clasificadores en distintos puntos de operación.

Métricas de clasificación binaria: tasa de acierto, sensibilidad y especificidad

Antes de definir en qué consiste cada una de ellas: tasa de acierto, sensibilidad y especificidad, sería conveniente explicar la matriz de confusión, ya que a raíz de esta pueden obtenerse las métricas nombradas.

La matriz de confusión es una tabla cruzada que muestra el conteo de los aciertos y errores del clasificador en cada una de las clases (predicciones) en relación con la clase real de las observaciones. La clase positiva corresponde a la codificada como 1 y la negativa como 0.

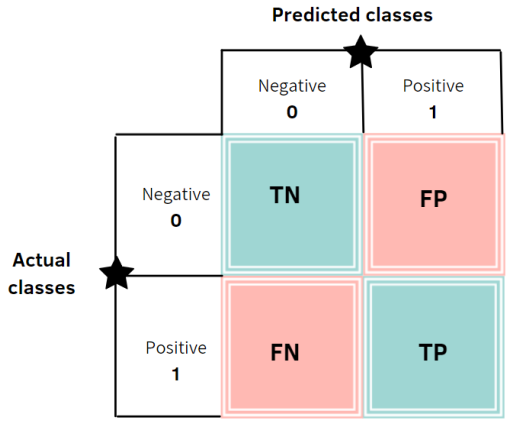


Figura 3. Matriz de confusión. Fuente: Towards Al

- *True Positive* o *TP* hace referencia a aquella observación que es positiva y que es predicha como tal.
- *False Negative* o *FN* representa aquella observación que es positiva y que es predicha como negativa.
- *False Positive* o *FP* indica que la observación es negativa, pero ha sido predicha como positiva.
- *True Negative* o *TN* hace referencia a aquella observación que es negativa y que es predicha como tal.

Una vez tengamos la matriz de confusión podemos obtener la tasa de acierto, de sensibilidad y de especificidad.

La tasa de acierto, *accuracy* o *Acc* es la relación entre el número de predicciones correctas y el número total de muestras de entrada. Mide el porcentaje de las instancias clasificadas correctamente por el clasificador.

$$Acc = \frac{Número\ de\ aciertos\ del\ clasificador}{Número\ total\ de\ observaciones} = \frac{TP\ +\ TN}{TP\ +\ TN\ +\ FP\ +\ TN}$$

La sensibilidad, *sensitivity* o *Se*, por su parte, es la relación entre el número de predicciones correctas de la clase positiva y el número total de muestras de entrada de la clase positiva. Es decir, es la tasa de observaciones positivas clasificadas correctamente.

$$Se = \frac{Número\ de\ aciertos\ del\ clasificador\ de\ la\ clase\ positiva}{Número\ total\ de\ observaciones\ de\ la\ clase\ positiva} = \frac{TP}{TP\ +\ FN}$$

La especificidad, *specificity* o *Sp* es la relación entre el número de predicciones correctas de la clase negativa y el número total de muestras de entrada de la clase negativa. Es decir, es la tasa de observaciones negativas clasificadas correctamente.

$$Sp = \frac{Número\ de\ aciertos\ del\ clasificador\ de\ la\ clase\ negativa}{Número\ total\ del\ observaciones\ de\ la\ clase\ negativa} = \frac{TN}{FP\ +\ TN}$$

3 Experimentación

3.1 Introducción del problema

Cada vez es más frecuente la compra *online*, ya que conlleva numerosas ventajas tanto para los usuarios como para los negocios, como que no hay que esperar colas para comprar, puedes comparar varios productos similares en el mismo momento, pudiendo encontrar una gran cantidad de opciones, las tiendas online están disponibles todos los días a todas horas. Para los negocios es una posibilidad de encontrar más clientes o de localizar mejores tiendas a través de los buscadores, facilita la gestión del inventario, entre otros.

La compra *online* o en línea es una forma de comercio que utiliza dispositivos electrónicos que permite a los consumidores comprar bienes o servicios de los vendedores a través de Internet (Kurniawan, I, y otros, 2020).

Para potenciar estos entornos virtuales, las ofertas de marketing es una de las estrategias más importantes. Siempre estas ofertas se sugerían indiscriminadamente a todos los visitantes de un sitio web de comercio electrónico determinado, pero esto ha cambiado (Baati, K & Mohsil, M, 2020). Somos conscientes de que nos encontramos ante una publicidad masiva, por lo que no es conveniente atosigar a todos los visitantes. Para ello se le ofrecerá dichas promociones personalizadas a aquellos visitantes cuya tasa de abandono de la cesta de la compra sea elevada, no ofreciéndose a aquellos usuarios que tienen alta probabilidad de realizar la conversión.

Hay que tener en cuenta, además, que el aumento del uso del comercio electrónico no va de la mano del aumento de las tasas de conversiones. Y esto lleva a la necesidad de soluciones que presenten promociones personalizadas a los compradores *online* (Baati, K & Mohsil, M, 2020).

3.2 Estado de arte

Es más común que las empresas inviertan en sistemas de detección y predicción del comportamiento que imitan el comportamiento de un vendedor en un entorno de compra virtual, especialmente aquellas empresas que pertenecen al sector del comercio electrónico y las tecnologías (Albert TC, Goes PB, & Gupta A, 2004)

Además, encontramos algunos estudios basados en la categorización de las visitas, y otros con objeto de predecir la intención de compra de los consumidores online, con la finalidad de tomar acciones para mejorar las tasas de abandono de la cesta de la compra y, por tanto, la conversión. Todos ellos utilizan métodos de aprendizaje automático para afrontar el problema (Sakar, CO, Polat, SO, Katircioglu, M, & Kastro, Y, 2019)

En el primer estudio al que hacemos referencia, basados en la categorización de las visitas, presentan y evalúan dos técnicas de agrupación basadas en las transacciones de los usuarios y en las páginas vistas, con el fin de descubrir perfiles agregados que sean eficaces para los sistemas de recomendación a la hora de la realización de acciones específicas en tiempo real. Los resultados demostraron que los perfiles extraídos pueden ser útiles para lograr una personalización efectiva en las primeras etapas de las visitas de los usuarios a un sitio virtual (Mobasher B, Dai H, Luo T, & Nakagawa M, 2002).

Siguiendo esta línea, encontramos otro estudio en el que se propuso un sistema que, utilizando los datos de flujo de clics de las páginas de una determinada tienda *online*, realiza acciones personalizadas según la categoría de la visita. La categorización de las visitas se realiza según su probabilidad de compra y por eso encontramos visita de compra, navegación, búsqueda o creación de conocimiento. Cada tipo de visita responderán de manera diferente a los mensajes de marketing, por ello esta capacidad de categorizar las vistas permite diseñar un mensaje promocional personalizado y más efectivo (Moe WW, 2003).

Otros estudios construyen un nuevo enfoque para analizar los datos históricos obtenidos de una librería en línea real y abordar el problema de la caracterización del comportamiento de los clientes electrónicos. Se basa en el descubrimiento de reglas de asociación en las sesiones de los usuarios para identificar las funciones que indican una alta probabilidad de realizar una compra para dos grupos de clientes: tradicionales e innovadores (Suchacka & Chodak, 2017)

Encontramos estudios donde utilizaron los datos históricos recopilados de una librería en línea para categorizar las sesiones de usuario como sesiones de navegación y de compra (Suchacka G, Skolimowska-Kulig, M, & Potempa, A, 2015).

Basados en el objetivo de predecir la intención de compra de los consumidores *online* también encontramos varios estudios. En uno de ellos se propone un sistema, teniendo en cuenta la pérdida de rendimiento en los servidores web debido a la sobrecarga, para asignar prioridades a las sesiones en sitios webs de acuerdo con los ingresos que generarán utilizando los datos del flujo de clics y la información de la sesión (Poggi, N, Moreno, T, Berral, JL, Gavaldà. R, & Torres, J, 2007).

Otros estudios proponen clasificar los patrones de comportamiento de los visitantes, con el fin de determinar el componente del sitio web que tiene mayor impacto en el cumplimiento del objetivo comercial (Budnikas, G, 2015).

Y por último y por ello no menos importante, encontramos un artículo de 2020 en el que predijeron en tiempo real la intención de compra de los compradores *online* mediante redes neuronales recurrentes y perceptrón multicapa. Para ello crean dos módulos: el primero que define la intención de compra del visitante y el segundo que estima la probabilidad de abandono del sitio web sin finalizar la transacción. Ambos módulos se utilizan conjuntamente para determinar los visitantes que tienen intención de comprar, pero que probablemente abandonará el sitio en el horizonte de predicción, y tomar medidas en consecuencia para mejorar las tasas de abandono del sitio web y de conversión de compras. El primer módulo se activa sólo si el segundo módulo produce un valor mayor que el umbral predeterminado (Sakar, CO, Polat, SO, Katircioglu, M, & Kastro, Y, 2019).

3.3 Materiales

Una vez comprendidos los fundamentos teóricos y la problemática, establecidos los objetivos y estudiados los antecedentes, el siguiente paso es explicar los materiales empleados en la metodología, comenzando por la descripción de la base de datos utilizada.

En primer lugar, se dispuso de un conjunto de datos de sesiones de clientes adecuado para la problemática presentada. Contiene 12.330 instancias descritas por 18 variables relacionadas con diferentes categorías como el tipo de páginas visitadas, métricas medidas por Google Analytics, como la tasa de rebote (*bouncerates*), variables demográficas, entre otras. De las cuales 10 son numéricas y 8 categóricas, incluyendo la variable de salida o de respuesta.

Posteriormente, se analizan y preprocesan los datos obteniendo los estadísticos descriptivos de las características numéricas, la correlación entre las variables numéricas, así como la relación de estas con la variable de salida, y la existencia, o no, de valores perdidos. Se representan los histogramas y los diagramas de barras, dependiendo del tipo de variable. Además, se estandarizan las variables a media cero y varianza unitaria.

Tras la estandarización, se divide el conjunto de datos en 70% entrenamiento - 30% validación, de forma que en el conjunto de entrenamiento haya un 70% de las muestras de la clase minoritaria, y en el de validación un 30% de las instancias de la clase minoritaria.

Dado el desbalance de los datos (10.422 sesiones que no acaban en compra, frente a 1.908 que si finalizan con compra), vamos a aplicar la técnica de submuestreo aleatorio en el conjunto de entrenamiento, eliminando instancias de la clase mayoritaria (en nuestro caso, sesiones que no finalizan con compras) hasta obtener el mismo número de muestras de ambas clases (clases balanceadas). Este es el conjunto de datos que utilizaremos para entrenar los modelos con las diferentes técnicas de clasificación nombradas y explicadas anteriormente: máquina de vector soporte, árbol de decisión y bosque aleatorio.

Aplicaremos las dos técnicas de selección de predictores: basada en filtros y envolventes, y una vez que tengamos los dos vectores de atributos óptimos de cada clasificador, entrenamos y evaluamos cada modelo. La métrica de referencia en la evaluación será el área bajo la curva (*AUC*) obtenido de la curva ROC. Además, para el mejor modelo según este valor, vamos a

obtener las métricas de acierto, sensibilidad y especificidad tanto en el punto operativo o umbral óptimo como en otros puntos objetivos de la tabla.

3.3.1 Base de datos

La tabla 1 describe el conjunto de datos seleccionado (*online_shoppers_intention*) para el análisis de diferentes algoritmos de Machine Learning de clasificación. Ha sido obtenido del repositorio de aprendizaje automático de UC Irvine (*UCI Machine Learning Repository*).

El modelo de intención de compra está diseñado como un problema de clasificación binaria que mide la intención del usuario de finalizar la transacción. Así, se puede ofrecer contenidos sólo a aquellos que tienen intención de comprar y no ofrecer contenidos a los demás usuarios.

El conjunto de datos consta de 12.330 sesiones o instancias y 18 características incluyendo la variable de salida o de respuesta. Ésta es *revenue* y se trata de un atributo categórico o nominal. Decimos que estamos ante un problema de clasificación binaria porque solo puede tomar dos valores: *False* y *True*. *False* indica que el usuario no ha generado ingresos, es decir, no ha finalizado en compra, y es codificada como 1. *True*, por el contrario, indica que el usuario si ha generado ingresos (ha finalizado con compra), y está codificada como 2.

Cada sesión o cada instancia pertenece a un usuario diferente en un periodo de 1 año, evitando así cualquier tendencia a una campaña específica, un periodo concreto o un perfil de usuario.

Tabla 1. Descripción de las variables. Fuente: artículo *Real time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks.*

Variables numéricas	
	Descripción
Administrative	Número de páginas visitadas por el usuario sobre la gestión de cuentas.
Adminstrative_duration	Tiempo total en segundos que el usuario ha pasado en las páginas que ha visitado sobre la gestión de cuentas.
Informational	Número de páginas visitadas por el usuario sobre el sitio web.
Informational_duration	Tiempo total en segundos que el usuario ha pasado en las páginas que ha visitado sobre el sitio web.
Productrelated	Número de páginas visitadas por el usuario sobre las páginas relacionadas con el producto.

Productrelated_duration	Tiempo total en segundos que el usuario ha pasado en las páginas relacionadas con el producto.
Bouncerates	Porcentaje de visitantes que entran en la página web y luego se van sin provocar ninguna otra solicitud al servidor de análisis durante esa sesión.
Exitrates	Porcentaje de abandono de la página web.
Pagevalues	Representa el valor medio de una página web que un usuario visitó antes de completar una transacción de comercio electrónico.
Specialday	Indica la proximidad del tiempo de visita a un día especial específico, como el Día de San Valentín. En estos días es más probable que las sesiones finalizan con una transacción. El valor de este atributo se determina teniendo en cuenta la dinámica del comercio electrónico, como la duración entre la fecha de pedido y la fecha de entrega. Por ejemplo, para San Valentín, se establece un valor no nulo entre el 2 y el 12 de febrero, y cero antes y después de esta fecha a menos que esté cerca de otro día especial.

Variables categóricas

	Descripción	**Números de valores categóricos**
Operatingsystem	Indica el sistema operativo del visitante por el que accede a la Web	8
Browser	Indica el navegador que utiliza el usuario para acceder a la Web	13
Region	Región geográfica desde la que el visitante ha iniciado la sesión.	9
Traffictype	Fuente de tráfico por la que el visitante ha llegado al sitio web: banner, SMS, de forma directa, etc.	20
Visitortype	Indica si estamos ante un visitante nuevo (*New_Visitor*), recurrente (*Returning_Visitor*) u otro (*Other*).	3
Weekend	Valor booleano que indica *False* si no es fin de semana, y *True* si lo es.	2
Month	Mes de la fecha de la visita.	12
Revenue	Etiqueta de clase o variable de salida que indica si la visita ha sido finalizada con una transacción (*True*) o no (*False*).	2

Administrative, *administrative_duration*, *informational*, *informational_duration*, *productrelated* y *productrelated_duration* representan el número de diferentes tipos de páginas visitadas por el usuario en esa sesión, así como el tiempo (en segundos) invertido en cada una de las categorías. Estos valores derivan de la información URL de las páginas visitadas.

Bouncerates y *exitrates* representan dos métricas medidas por Google Analytics para cada página del sitio del comercio electrónico. ¿Qué diferencia hay entre la tasa de rebote (*bouncerates*) y la de salida (*exitrates*)? Bien, la tasa de rebote hace referencia a aquel porcentaje de visitantes que

ingresan al sitio web, permanecen únicamente unos pocos segundos y se van. Y la tasa de salida muestra cuántas personas salen de una página web en particular, es decir, el porcentaje de abandono de la web.

Otra variable es *specialday* que indica la proximidad del tiempo de visita al sitio a un día especial específico como el Día de la Madre. Así como *weekend* que hace referencia si es fin de semana o no, y *month* que indica el mes de la sesión.

También incluye la variable *operatingsystem*, que hace referencia al sistema operativo desde el que se accede a la web, así como *browser*, que indica el tipo de navegador de acceso.

Como variable demográfica incluye *region.* Y también incluye las variables *traffictype* y *visitortype*. La primera de ellas indica la fuente de tráfico por la que el visitante ha llegado al sitio web, por ejemplo, banner, SMS, de forma directa, etc. Y la segunda hace referencia si se trata de un visitante nuevo (*New_Visitor*), recurrente (*Returning_Visitor*) u otro (*Other*).

3.3.2 Análisis de los datos

Para el análisis de los datos hemos utilizado dos softwares: Stata y Python.

Stata se emplea para describir los datos, obtener los estadísticos descriptivos de las variables numéricas, y obtener la tabla de frecuencia de la variable de salida.

La tabla 2 siguiente recoge los estadísticos descriptivos (media, desviación típica, valor mínimo y valor máximo) de las variables numéricas.

Tabla 2. Estadísticos descriptivos de las variables numéricas. Fuente: elaboración propia

	Media	Desviación típica	Valor mínimo	Valor máximo
Administrative	2,32	3,32	0	27
Administrative_duration	80,82	176,78	0	3.398,8
Informational	0,504	1,26	0	24
Informational_duration	34,47	140,75	0	2.549,4
Productrelated	31,73	44,48	0	705

Productrelated_duration	1.194,75	1.913,67	0	63
Bouncerates	0,02	0,04	0	0,2
Exitrates	0,04	0,05	0	0,2
Pagevalues	5,89	18,57	0	361,8
Specialday	0,06	0,19	0	1

De las 12.330 sesiones que forman el conjunto de datos, 10.422 eran muestras de la clase negativa (*False*) que no terminaban en conversión. El resto, es decir, 1.908 eran muestras de la clase positiva (*True*) que si terminaban en conversión. Estamos ante un problema con datos muy desbalanceados, en el que el 84,53% de las muestras pertenece a la clase negativa.

El análisis de los datos sigue en el Anexo 2, pero aquí utilizamos Python. Con este software vamos a analizar los siguientes apartados:

Valores perdidos

Podemos comprobar que todas las características presentan 12.330 instancias, por lo que nuestra base de datos no presenta valores perdidos.

Histogramas de las variables numéricas y discretas

Ninguna de las características se distribuye normalmente, ya que están muy sesgadas hacia la derecha.

Si nos fijamos en los histogramas referentes a la duración que permanece cada usuario en cada uno de los distintos tipos de páginas, podemos observar que la mayoría de sus valores se encuentran en pocos segundos. La que presenta mayor duración en cuanto a segundos son aquellas páginas relacionadas con el producto, y no las de gestión de cuentas ni informativas.

Matriz de correlación de las variables numéricas

Se considera que existe una correlación alta cuando su coeficiente es mayor que 0,5. A primera vista hay dos variables que parecen estar muy correlacionadas entre ellas: *bouncerates* y *exitrates*. Pero si nos fijamos más detalladamente en la matriz de correlación podemos observar que existen otros tres pares de variables cuyos coeficientes superan al nombrado anteriormente

de 0,5. Tenemos *productrelated* y *productrelated_duration*, *informational* e *informational_duration* así como *administrative* y *administrative_duration*.

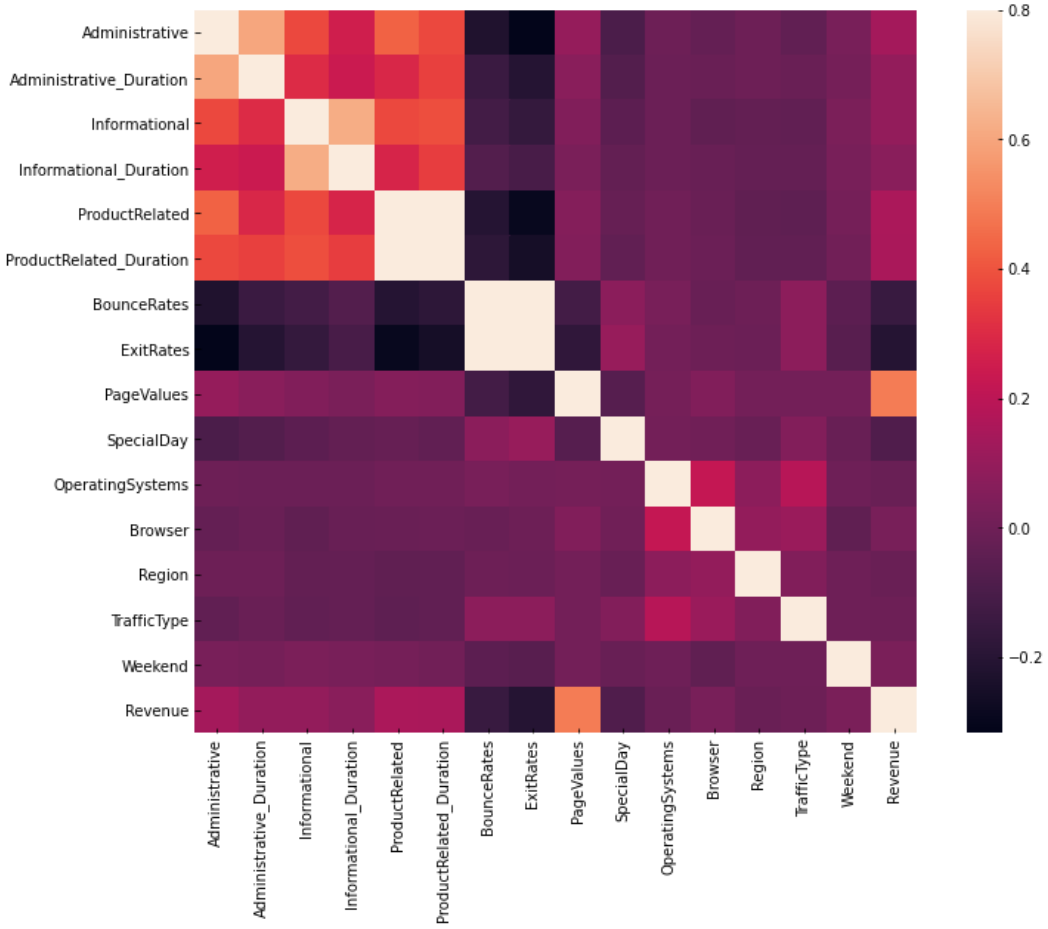


Figura 4. Matriz de correlación de las variables numéricas. Fuente: elaboración propia

Aquellas variables que presentan correlación alta entre ellas están relacionadas con el tipo de página visitada por el usuario y las métricas obtenidas con Google Analytics.

Vamos a ver detalladamente la correlación entre cada par de variables nombradas anteriormente. El coeficiente de correlación entre cada una de ellas se encuentra en la tabla 3 siguiente, y, además, hemos elaborado algunos gráficos de dispersión.

Tabla 3. Coeficiente de correlación de las variables numéricas. Fuente: elaboración propia

Variables	Coeficiente de correlación

Bouncerates y exitrates	0,913
Productrelated y productrelated_duration	0,86
Informational e informational-duration	0,619
Administrative y administrative-duration	0,602

Si nos fijamos en las variables relacionadas con las métricas de Google Analytics, tanto en el coeficiente de correlación como en su gráfico de dispersión que se muestra a continuación, podemos confirmar que existe una correlación muy fuerte y positiva.

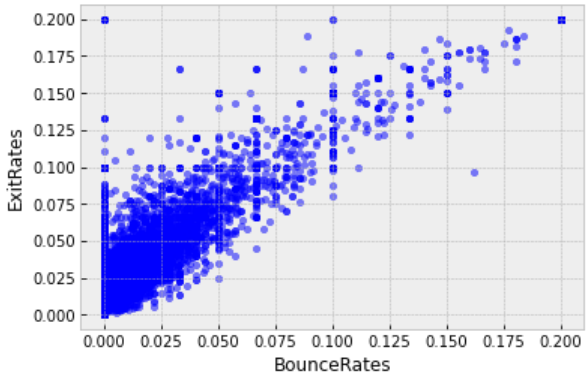


Figura 5. Gráfico de dispersión de las variables numéricas de Google Analytics. Fuente: elaboración propia

La correlación entre *productrelated* y *productrelated_duration* también es alta y positiva, como podemos confirmar con su gráfico de barras.

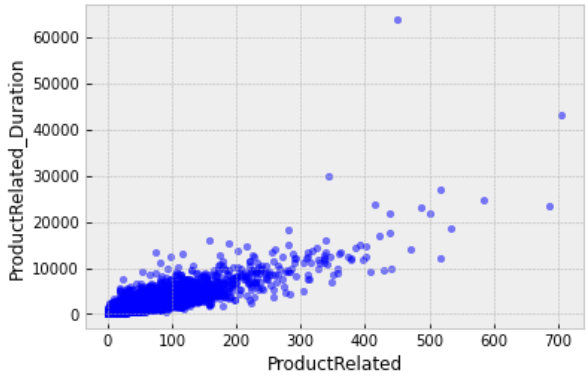


Figura 6. Gráfico de dispersión de las variables numéricas *productrelated* y *productrelated_duration*. Fuente: elaboración propia

Sin embargo, la correlación entre las páginas informativas y su duración, y las páginas relacionadas con la gestión de cuentas y su duración, no es tan alta y fuerte como estas dos, pero si es positiva también.

Hasta ahora hemos hablado de la correlación entre variables, pero también sería conveniente ver la correlación de las variables con respecto a la variable de salida *revenue.* De entre todas las variables, ninguna obtiene un coeficiente de correlación superior al 0,5, haciendo mención especial a la variable *pagevalues*, que casi iguala ese valor. Su coeficiente de correlación es de 0,49. Esto significa que es una de las variables más importantes que presenta y explica el modelo y, por tanto, estaría bien que apareciera en las primeras posiciones de los vectores de predictores óptimos utilizados.

Gráfico de barras de las variables categóricas

Al representar los gráficos de barras de las variables categóricas, es decir, de *visitortype*, *month*, *weekend* y la variable de salida *revenue* podemos extraer las siguientes conclusiones:

- La mayoría de los visitantes que acceden a la Web son visitantes recurrentes, concretamente 10.551 usuarios de los 12.330 de nuestro conjunto de datos, 1.694 son visitantes nuevos y el resto, 85, visitantes pertenecientes a la categoría otros.

- Con respecto a la variable *month,* podemos ver que la mayoría de las visitas se han producido durante el mes de febrero, seguido de marzo y de mayo. Siendo diciembre el mes con menor números de muestras.

- Además, la mayoría de las visitas se producen entre semana, ya que el valor *False* de *weekend* es mucho más elevado que *True*.

- En referencia a la variable de salida *revenue,* ya hemos comentado que la mayoría de las muestras pertenecen al valor *False*, es decir, que no terminan en conversión. Y que estamos ante un conjunto de datos muy desbalanceado.

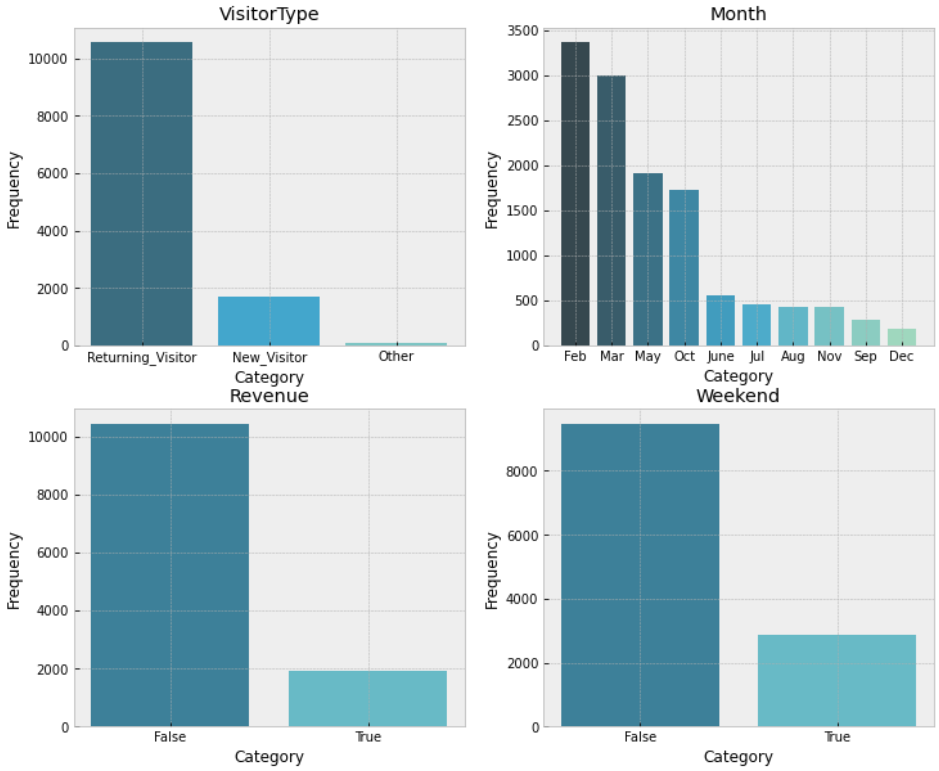


Figura 7. Gráficos de barras de las variables categóricas. Fuente: elaboración propia

Análisis univariante de los datos

Anteriormente hemos comentado que la mayoría de las visitas se producen entre semana, pero ¿la mayoría de los ingresos también se producen entre semana? Para ello hemos representado gráficamente ambas variables: *revenue* y *weekend*, obteniendo el gráfico siguiente:

Figura 8. Gráfico de barras de las variables *revenue* y *weekend*. Fuente: elaboración propia

Efectivamente la mayoría de los ingresos han sido obtenidos entre semana. El que exista el mayor número de visitas entre semana también significa que la mayoría de los ingresos o la mayoría de las compras se realizan entre semana.

¿Ocurre lo mismo con el tipo de visitante? ¿Podemos decir que los visitantes recurrentes son los que generan más ingresos? Como podemos comprobar y confirmar en el gráfico X que mostramos a continuación, los visitantes recurrentes son los que generan el 85,57% de los ingresos.

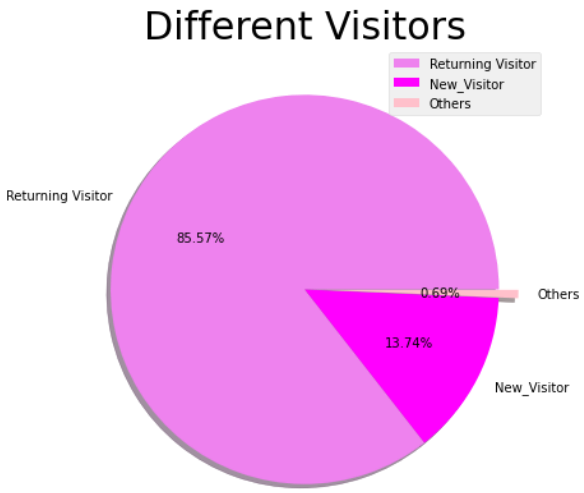


Figura 9. Gráfico de barras de las variables *revenue* y *visitortype*. Fuente: elaboración propia

3.3.3 Selección del conjunto de entrenamiento

En la clasificación supervisada frecuentemente aparecen problemas donde la cantidad de objetos de una clase es significativamente mayor que la cantidad de objetos de otra clase. A este tipo de problemas se les llama problemas con clases desbalanceadas. Los datos desbalanceados afectan a los algoritmos en su proceso de generalización de la información y perjudican a las clases minoritarias, siendo, comúnmente, la que representa el concepto más importante (Castillo, Cabrera, Chávez, & García, 2020).

Como podemos observar en la figura 10, nuestra base de datos está compuesta por 12.330 sesiones de usuarios diferentes, de los 10.422 no finalizaron con compras, frente a 1.908 que si finalizaron. Claramente puede observar el gran desbalanceo existente.

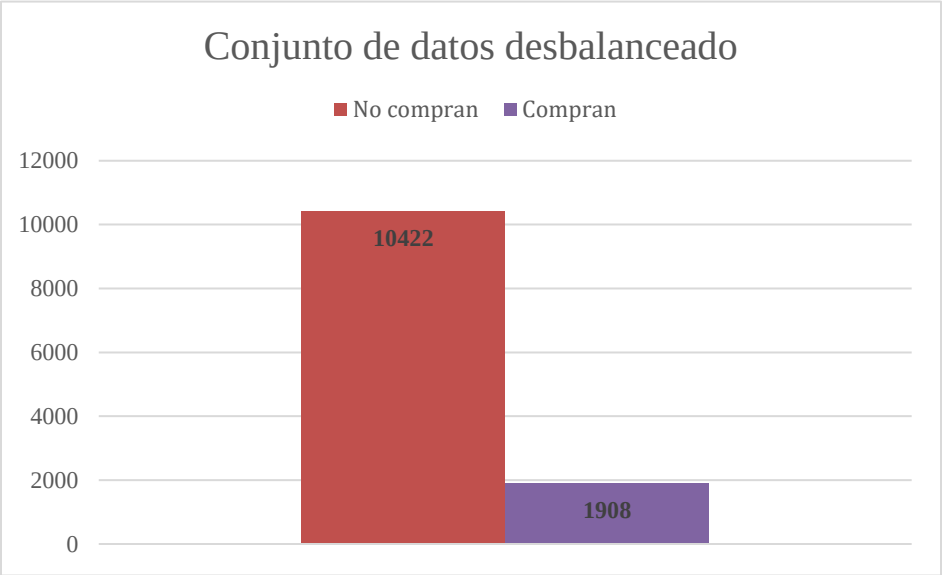


Figura 10. Conjunto de datos desbalanceados. Fuente: elaboración propia

Para hacer frente a este problema existen diversas técnicas, las cuales pueden mejorar la calidad de los datos ayudando a mejorar la precisión y eficiencia de los procesos. Entre todas las técnicas posibles y disponibles, nosotros aplicaremos el submuestreo o *downsampling*. Con esta técnica de muestreo se pretende conseguir una distribución uniforme de las clases eliminando instancias de la clase mayoritaria (*False*, no generan ingresos al no realizar la compra, en nuestro conjunto de datos).

Esta técnica la emplearemos en Matlab, y todos los pasos que explicaremos vienen más detallados en el Anexo 3.

Antes de aplicar el submuestreo, debemos dividir el conjunto de datos ya estandarizado o normalizado en 70% entrenamiento y 30% validación, de forma que, además, en cada conjunto se cumpla la misma proporción de la clase minoritaria, es decir, que, de todas las muestras del conjunto de entrenamiento, el 70% corresponde a la clase minoritaria, y el 30% en el conjunto de test.

Como resultado obtenemos que nuestro conjunto de entrenamiento está formado por 8.631 muestras, de las cuales 7.295 pertenecen a la clase mayoritaria (*False*: no compran, codificada como 1) y 1.336 a la clase minoritaria (*True*: compran, codificada como 2). Y nuestro conjunto de validación presenta 3.699 instancias de las cuales 3.127 pertenecen a la clase mayoritaria y el resto, 572, a la minoritaria.

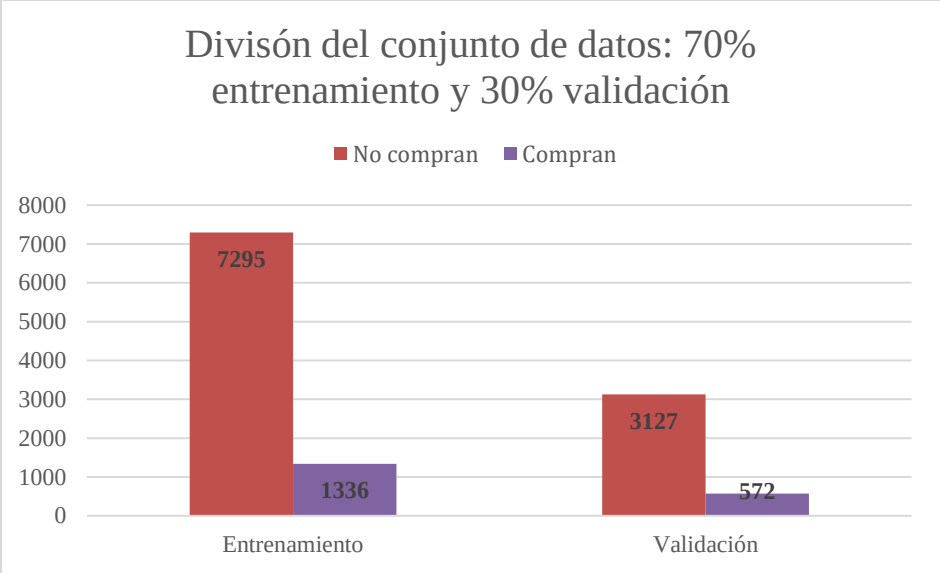


Figura 11. División del conjunto de datos: 70% entrenamiento y 30% validación. Fuente: elaboración propia

Una vez tengamos el conjunto de entrenamiento y validación, y dado el desbalanceo de los datos, el conjunto de entrenamiento fue balanceado mediante submuestreo aleatorio. Del total de muestras de entrenamiento (8.631), obtendremos una muestra menor, ya que nos quedamos con todas las instancias de la clase minoritaria, en nuestro caso que generan ingresos (compran), y eliminamos las muestras de la clase mayoritaria hasta obtener el mismo número de instancias de ambas clases. Es decir, hemos dicho anteriormente que el número de instancias de la clase minoritaria en el conjunto de entrenamiento es 1.336, por tanto, el conjunto de entrenamiento que obtendremos tras el submuestreo está formado por las 1.336 muestras de la clase minoritaria y 1.336 muestras elegidas aleatoriamente entre las 7.295 pertenecientes a la clase mayoritaria, en total, estaría formado por 2.672 instancias.

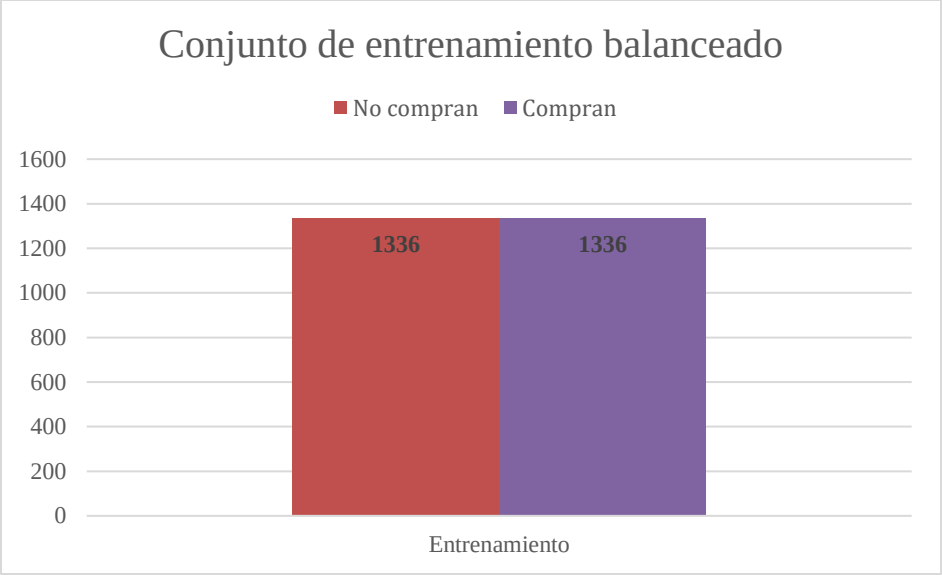


Figura 12. Conjunto de datos balanceados. Fuente: elaboración propia

Este conjunto de datos es el que utilizamos para entrenar los diferentes modelos con las siguientes técnicas de clasificación: máquina de vector soporte (*SVM*) en sus tres *kernels* posibles (*lineal*, *gaussiano* y *polinomial*), árbol de decisión (*Tree*) y bosque aleatorio (*Random Forest*).

3.4 Metodología

3.4.1 Entrenamiento de los modelos

Para entrenar los modelos, utilizamos el conjunto de entrenamiento obtenido tras el submuestreo, y, además, los modelos de clasificación elegidos fueron configurados para trabajar con dos conjuntos de predictores seleccionados tras la aplicación de dos tipos de técnicas de selección: basadas en filtro (MRMR) y métodos envolventes (*Wrapper* o búsqueda escalonada hacia adelante).

En el caso del árbol de decisión vamos a podarlo. Para ello aplicamos *Cost Complexity Pruning* al árbol grande, obteniendo así una secuencia de los mejores subárboles, como una función del parámetro de complejidad alfa. A continuación, evaluamos cada subárbol mediante *K-Fold cross validation* con 10 particiones para elegir el valor de alfa o subárbol asociado. El subárbol seleccionado (o valor de alfa, que es equivalente) es el que presenta el mínimo error medio.

Y con respecto al bosque aleatorio, el número de árboles utilizados es 50.

Una vez entrenado cada modelo, lo aplicamos en el conjunto de validación. En los casos de máquina vector soporte, debe adaptarse la salida del modelo para proporcionar probabilidad de pertenencia a clases en el rango 0-1. Por defecto, los modelos *SVM* ofrecen como salida la distancia de las instancias bajo consideración al hiperplano de separación.

3.4.2 Evaluación de los modelos

La comparación entre los modelos se realiza calculando sus curvas de rendimiento (curva ROC), concretamente nuestra métrica de referencia será el área bajo la curva (*AUC*). A raíz del mejor modelo según *AUC*, vamos a aplicar las métricas de clasificación binaria tasa de acierto (*Acc*), sensibilidad (*Se*) y especificidad (*Sp*) tanto al umbral o punto operativo óptimo, como en otros puntos objetivos de la curva en los que el clasificador puede trabajar. Éstos son para niveles de especificidad y sensibilidad objetivos 65-70-75-80-85-90-95.

Para ello, primero generamos la curva ROC de cada modelo. Ésta devuelve la matriz de umbrales, las coordenadas de cada umbral, es decir, la sensibilidad y la tasa de falsos positivos o correspondientes, así como el área bajo la curva y el punto óptimo.

De las tres métricas de clasificación binaria que obtenemos, la especificidad es la más importante. A continuación, explicamos por qué.

Nuestro objetivo es predecir la intención de compra con la finalidad de conocer a aquellos usuarios cuya tasa de abandono sea alta, y tomar acciones para mejorar la tasa de abandono de la cesta de la compra y de conversión de las compras. Nuestra variable de salida presenta dos valores: *True*, para aquellos usuarios que generan ingresos, es decir, que terminan en conversión, y *False*, para aquellos que no generan ingresos y cuya tasa de abandono será elevada. La primera tiene la condición de clase positiva, y la segunda, de negativa. La especificidad representa el total de observaciones negativas clasificadas correctamente, por lo tanto, nos interesa que esta sea lo más elevada posible, para tomar las acciones promocionales con los usuarios adecuados.

3.5 Resultados

3.5.1 Selección de las características

Como ya comentamos, vamos a seleccionar el mejor vector de predictores según las dos técnicas explicadas. En cada clasificador se van a utilizar dos modelos, uno para cada vector óptimo de cada técnica. Si tenemos los siguientes clasificadores: *SVML*, *SVMG*, *SVMP*, *Tree* y *Random Forest*, obtendremos en total diez modelos.

Para llevar a cabo la primera técnica, deberemos clasificar los predictores en función de su importancia con el algoritmo *MRMR*, obteniendo así un ranking de los predictores.

El vector clasificado obtenido con *MRMR* es: [9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13] lo que significa que la variable 9 es que presenta mejor puntuación y la variable 13 la que menos. Si escribimos el nombre de la variable, en vez de su posición, obtenemos que la variable mejor clasificada es *pagevalues*, la que comentamos anteriormente que presentaba una correlación más fuerte con la variable de salida (0,49). Y la que menor importancia tiene es *browser*, es decir, el navegador utilizado para la visita a la Web.

Nombres_vector_clasificado = [*pagevalues*, *weekend*, *exitrates*, *month*, *administrative*, *operatingsystems*, *productrelated_duration*, *visitortype*, *specialday*, *informational*, *bouncerates*, *traffictype*, *productrelated*, *administrative_duration*, *region*, *informational_duration*, *browser*]

Además, si nos fijamos, *exitrates* y *bouncerates*, están bastantes separadas en cuanto su posición, al igual que *administrative* y *administrative_duration*; *informational* e *informational_duration*; y *productrelated* y *productrelated_duration*, debido a que eran las variables que presentaban correlación fuerte entre ellas.

En cuanto a las puntuaciones obtenidas para cada predictor, aparece representada en la tabla 4.

Tabla 4. Puntuaciones de los predictores tras la clasificación. Fuente: elaboración propia

Variables	Puntuaciones
9 - pagevalues	0,2676
17 - weekend	0,1602

8 – exitrates	0,1522
11 – month	0,1282
1 – administrative	0,1030
12 – operatingsystems	0,0951
6 – productrelated_duration	0,0873
16 – visitortype	0,0619
10 – specialday	0,0496
3 – informational	0,047
7 – bouncerates	0,0466
15 – traffictype	0,0464
5 – producrelated	0,0344
2 – administrative-duration	0,0283
14 – región	0,016
4 – informational_duration	0,0157
13 – browser	0,0055

Representamos gráficamente los predictores clasificados y obtenemos el gráfico de barras que se muestra a continuación.

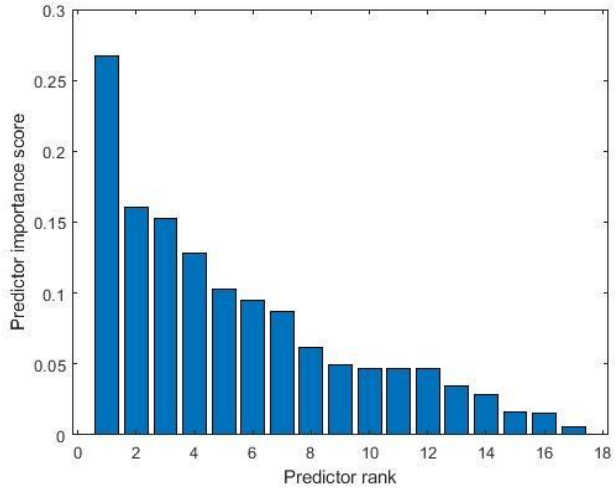


Figura 13. Gráfico de barras de las puntuaciones de los predictores clasificados. Fuente: elaboración propia

La caída de puntuación entre el primer predictor (*pagevalues*) y el segundo (*weekend*) más importante es muy grande. De hecho, es la caída más grande que encontramos en el que pasamos de una puntuación de 0,2676 a 0,1602, en total 0,1074 puntos de diferencia. Esto también reafirma la importancia del predictor *pagevalues*.

Sin embargo, también podemos considerar otras caídas significativas, en comparación con el resto de las caídas. Éstas son la caída del predictor 3 (*exitrates*) al 4 (*month*), así como del 4 (*month*) al 5 (*administrative*) y del 7 (*productrelated_duration*) al 8 (*visitortype*). El resto de las caídas son relativamente pequeñas.

Las grandes caídas implican que el software confía en la selección del predictor más importante, y los descensos pequeños indican que la diferencia entre la importancia de cada predictor no es significativa.

Una vez obtenido el mejor vector clasificado, aplicamos *K-Fold cross validation* con 10 particiones, a cada posible subconjunto de vectores, desde la dimensión 1 a la 17, y seleccionamos como vector óptimo aquel que proporcione mayor valor de tasa de acierto o *Acc*. Este vector óptimo obtenido para cada clasificador es el primero que utilizaremos para entrenar, validar y evaluar cada modelo con cada clasificador. Por ejemplo, para el clasificador máquina de vector soporte con *kernel polinomial*, el procedimiento seguido es el siguiente:

Tabla 5. Procedimiento de selección de predictores técnica 1 (*SVMP*). Fuente: elaboración propia

	Tasa de acierto
[9]	81,92%
[9, 17]	81,92%
[9, 17, 8]	81,81%
[9, 17, 8, 11]	81,1%
[9, 17, 8, 11, 1]	82,03%
[9, 17, 8, 11, 1, 12]	81,51%

[9, 17, 8, 11, 1, 12, 6]	82,15%
[9, 17, 8, 11, 1, 12, 6, 16]	80,39%
[9, 17, 8, 11, 1, 12, 6, 16, 10]	80,09%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3]	79,45%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7]	79%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15]	77,77%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5]	77,36%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2]	76,01%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14]	75,04%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4]	74,81%
[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13]	73,2%

Podemos observar que para el clasificador *SVM* con *kernel polinomial*, el vector de predictores óptimo obtenido con la técnica basada en filtros, es aquel cuya tasa de acierto sea mayor, es decir, el vector formado por 7 variables [9, 17, 8, 11, 1, 12, 6] cuya *Acc* es 82,15%.

Este procedimiento es el empleado para todos los modelos: *SVML*, *SVMG*, *Tree* y *Random Forest*.

La tabla 6 recoge el vector óptimo obtenido con la primera técnica para cada clasificador, así como su tasa de acierto en términos porcentuales.

Tabla 6. Vector de predictores óptimos clasificado y tasa de acierto, según clasificador. Fuente: elaboración propia

	Vector de predictores	Tasa de acierto
SVML	[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13]	81,81%
SVMG	[9]	82,82%
SVMP	[9, 17, 8, 11, 1, 12, 6]	82,15%
Tree	[9, 17]	80,76%
Random Forest	[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4]	85,4%

Si comparamos los porcentajes de *Acc* de cada clasificador, podemos ver que el vector obtenido con *Random Forest* es el que presenta una tasa de acierto mayor, del 85,4%, y es aquel formado por todas las variables exceptuando una (*browser*). Es un vector de dimensión 16.

La tasa de acierto obtenida nos dice que, al emplear dicho vector óptimo formado por 16 predictores, el 85,4% de las instancias o muestras son clasificadas correctamente.

En segundo lugar y haciendo referencia al segundo vector óptimo a utilizar, llevaremos a cabo el método *Wrapper*. Se trata de un método directo que busca el vector de atributos óptimo de cada dimensión (de 1 a 17) a través de la búsqueda escalonada hacia delante, y está relacionado con la técnica de clasificación. La estrategia de entrenamiento-evaluación que se aplica es *K-Fold cross validation* con 10 particiones, como en el caso anterior, con la diferencia que aquí utilizamos el vector de predictores sin un orden establecido, y no el clasificado, El criterio utilizado en este método es el de mayor rendimiento medido en términos de tasa de acierto según cada clasificador o mínimo error de la técnica en un conjunto de validación. El código empleado también aparece en el Anexo 3, junto con el de la técnica 1. Y el procedimiento seguido es parecido a la anterior técnica. Podemos verlo en la tabla 7 siguiente.

Este método devuelve la tasa de error de cada posible vector de predictor de cada dimensión y, a partir de ahí, podemos ver cuál es el óptimo, siendo este el que menor tasa de error presenta (mayor tasa de acierto). Por ejemplo, para el clasificador *SVMG* obtenemos la figura 14 siguiente sobre la tasa de error.

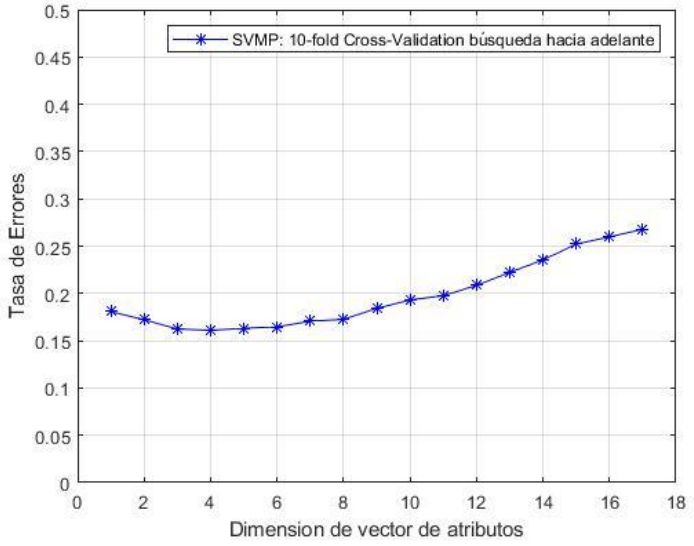


Figura 14. Tasa de error de los vectores de dimensión 1 a 17, utilizando la técnica *Wrapper* y *SVMP* como clasificador. Fuente: elaboración propia

Tabla 7. Procedimiento de selección de predictores técnica 2 (*SVMP*). Fuente: elaboración propia

	Tasa de acierto
[9]	81,92%
[9, 5]	82,78%
[9, 5, 11]	83,76%
[9, 5, 11, 3]	83,91%
[9, 5, 11, 3, 10]	83,68%
[9, 5, 11, 3, 10, 14]	83,53%
[9, 5, 11, 3, 10, 14, 16]	82,93%
[9, 5, 11, 3, 10, 14, 16, 6]	82,74%
[9, 5, 11, 3, 10, 14, 16, 6, 7]	81,58%
[9, 5, 11, 3, 10, 14, 16, 6, 7, 8]	80,69%
[9, 5, 11, 3, 10, 14, 16, 6, 7, 8, 4]	80,24%
[9, 5, 11, 3, 10, 14, 16, 6, 7, 8, 4, 13]	79,15%

[9, 5, 11, 3, 10, 14, 16, 6, 7, 8, 4, 13, 15]	77,77%
[9, 5, 11, 3, 10, 14, 16, 6, 7, 8, 4, 13, 15, 17]	76,46%
[9, 5, 11, 3, 10, 14, 16, 6, 7, 8, 4, 13, 15, 17, 12]	74,78%
[9, 5, 11, 3, 10, 14, 16, 6, 7, 8, 4, 13, 15, 17, 12, 2]	74,03%
[9, 5, 11, 3, 10, 14, 16, 6, 7, 8, 4, 13, 15, 17, 12, 2, 1]	73,2%

En este caso, para el mismo clasificador que antes (*SVMP*), pero empleando la técnica 2 (métodos envolventes), podemos observar que el vector óptimo es aquel formado por 4 variables, concretamente [9, 5, 11, 3] y su tasa de acierto asciende a 83,91%.

Además, en el gráfico anterior también podemos observar que el vector de predictores óptimo para el clasificador *SVMP* utilizando la técnica *Wrapper* es el vector formado por 4 predictores, es decir, de dimensión 4, ya que es el que menor tasa de error presenta (equivalente a una mayor tasa de acierto).

Al igual que para la técnica anterior, mostramos el vector de predictores óptimo y su tasa de acierto para cada clasificador con esta técnica. Se muestra en la tabla 8.

Tabla 8. Vector de predictores óptimos obtenidos con *Wrapper* y tasa de acierto, según clasificador. Fuente: elaboración propia

	Vector de predictores	**Tasa de acierto**
SVML	[9, 5, 11, 8, 4, 15, 10, 12, 3, 14, 2, 17]	81,92%
SVMG	[9, 5, 11, 7]	83,83%
SVMP	[9, 5, 11, 3]	83,91%
Tree	[9, 15, 17]	81,4%
Random Forest	[9, 5, 11, 7, 4, 10, 15, 13, 3, 14, 17, 6, 1]	85,74%

Ocurre de la misma forma que con la técnica 1. Al comparar cada clasificador, podemos ver que el vector obtenido con *Random Forest* es el que presenta una tasa de acierto mayor, del 85,74% y es aquel de dimensión 13, excluyendo las variables: *operatingsystems*, *administrative_duration*, *exitrates* y *visitortype*.

En este caso, el 85,74% de las instancias son clasificadas correctamente cuando se utiliza el vector formado por 13 variables.

Como conclusión de las técnicas de selección de predictores podemos decir que, de las dos técnicas empleadas, el vector de predictores óptimo obtenido con la técnica *Wrapper* presenta una tasa de acierto mayor, con respecto al obtenido con la clasificación, utilizando un menor número de características o predictores.

3.5.2 Curvas ROC

Como comentamos en el apartado de evaluación, utilizaremos la curva ROC (concretamente el valor del área bajo la curva), para comparar cada modelo generador, y las métricas de clasificación binaria: tasa de acierto, sensibilidad y especificidad para evaluar el mejor modelo.

A continuación, mostramos las curvas ROC de cada modelo. Como de cada clasificador tenemos dos modelos, se van a presentar conjuntamente. Por ejemplo, para *SVML* tenemos dos curvas ROC, una para el vector de predictores óptimo de la primera técnica, y otra para el vector óptimo de la segunda técnica.

En la leyenda de cada gráfica aparece el área bajo la curva (*Clasificador_Técnica_AUC*) y el punto de operación óptimo (*Clasificador_Técnica_PUNTO OPERACIÓN ÓPTIMO. UMBRAL*) de cada una de ellas, siendo la de mayor área aquella que explicará mejor nuestro problema.

La primera gráfica que aparece, la figura 15, representa las curvas ROC del modelo entrenado con máquina de vector soporte con *kernel lineal* (*SVML*).

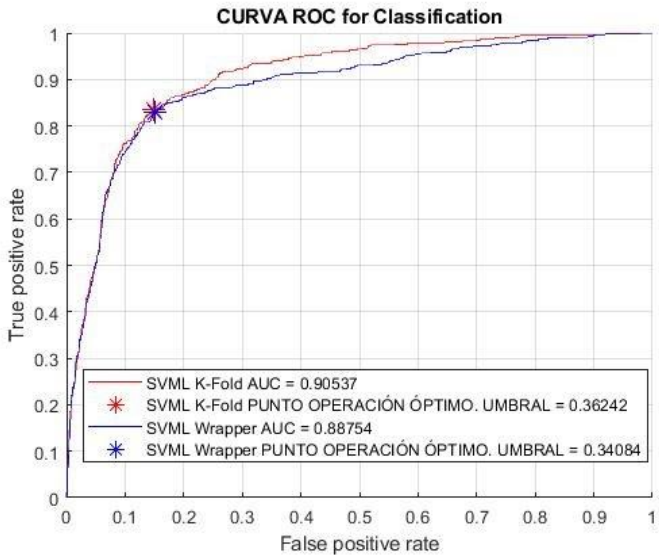


Figura 15. Curva ROC *SVML*. Fuente: elaboración propia

Podemos comprobar como el área bajo la curva, el valor de *AUC*, es mayor cuando se utiliza el vector de predictores óptimo obtenido con la primera técnica. El porcentaje de área bajo la curva es del 90,54%, frente al 88,75% obtenido con la segunda técnica. Si se empleara máquina de vector soporte con *kernel lineal*, es mejor trabajar con el vector de predictores óptimo obtenido con la clasificación *MRMR*.

La figura 16 muestra las curvas ROC del modelo entrenado con máquina de vector soporte con *kernel gaussiano* (*SVMG*).

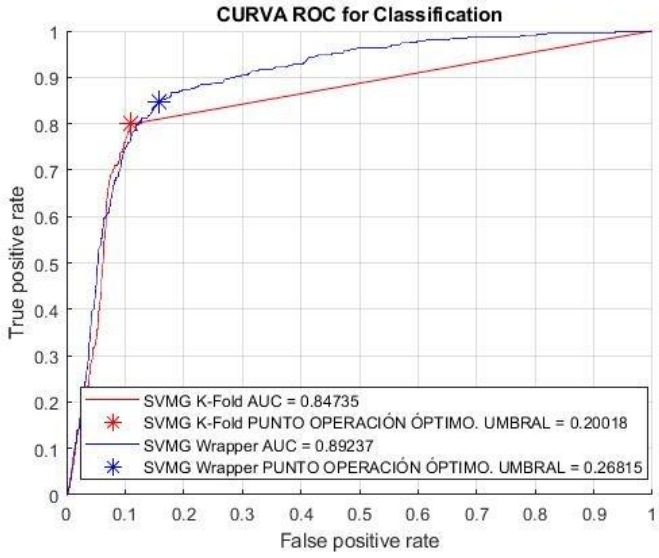


Figura 16. Curva ROC *SVMG*. Fuente: elaboración propia

A simple vista, sin fijarnos en el área bajo la curva de cada modelo, podemos decir que la curva ROC obtenida con el vector óptimo de la segunda técnica es mucho mejor que el obtenido con la primera técnica. Si nos basamos en los datos numéricos, obtenemos la misma conclusión, ya que para la segunda técnica (*Wrapper*) se obtiene un 89,24% de área bajo la curva, y para la primera técnica un 84,74%.

Con la técnica de clasificación de predictores *MRMR*, para *SVMG*, se obtiene que el vector óptimo está formado por una única variable: *pagevalues*. Sin embargo, al incluir tres variables más, vector de predictores óptimo obtenido con *Wrapper*: *productrelated*, *month* y *bouncerates*, se obtiene un área bajo la curva mayor en 5 puntos porcentuales. Se pasa del 84,74% al 89,74%.

Para terminar máquinas de vector soporte, mostramos la curva ROC del modelo entrenado con máquina de vector soporte con *kernel polinomial* (*SVMP*).

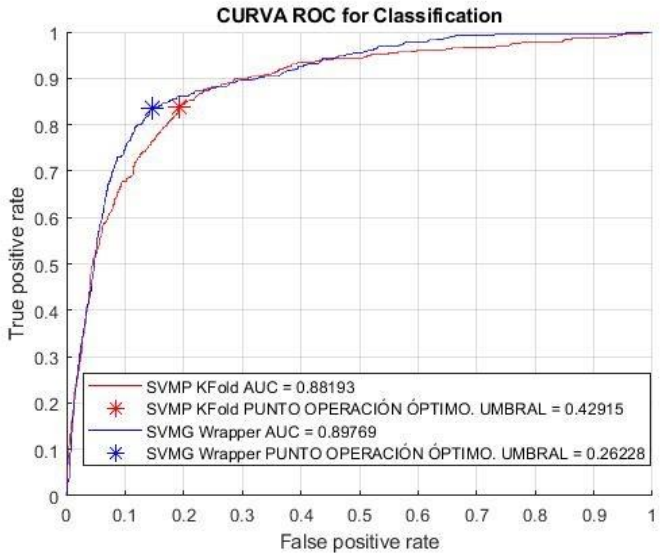


Figura 17. Curva ROC *SVMP*. Fuente: elaboración propia

Para máquinas de vector soporte con *kernel polinomial*, no obtenemos mucha diferencia en las curvas ROC, pero suficiente para saber gráficamente cuál es el mejor. El porcentaje de área bajo la curva de cada una de ellas es de 88,19%, y de 89,77%, según el vector de la técnica 1 o 2, respectivamente. El modelo con mayor área bajo la curva se obtiene cuando se utiliza el vector de predictores óptimo resultante de la técnica *Wrapper*.

A continuación, hemos realizado el mismo procedimiento para árboles de decisión (*Tree*). Recordamos que, en el entrenamiento de los árboles, estos han sido podados.

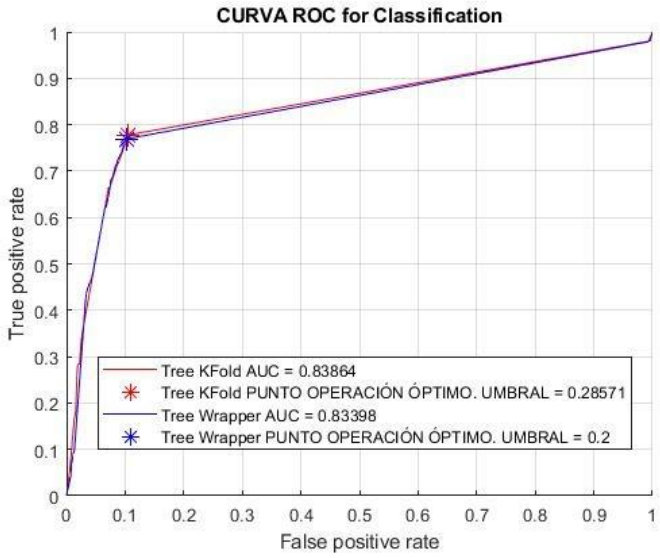


Figura 18. Curva ROC *Tree*. Fuente: elaboración propia

Las curvas ROC obtenidas al emplear el clasificador árbol de decisión son bastantes parecidas, siendo a simple vista difícil saber cuál es la que presenta mayor área bajo la curva. Ya si nos fijamos en la leyenda podemos observar como el árbol de decisión con el vector de predictores óptimo de la primera técnica presenta un área bajo la curva mayor, por poco. Esta asciende a 83,86%. La diferencia radica en la inclusión de un predictor más en la técnica *Wrapper* con respecto a la clasificación, y estaría formado por las siguientes tres variables: *pagevalues*, *traffictype* y *weekend*.

Por último, la figura 19 muestra la curva ROC cuando se emplea la técnica de clasificación *Random Forest*.

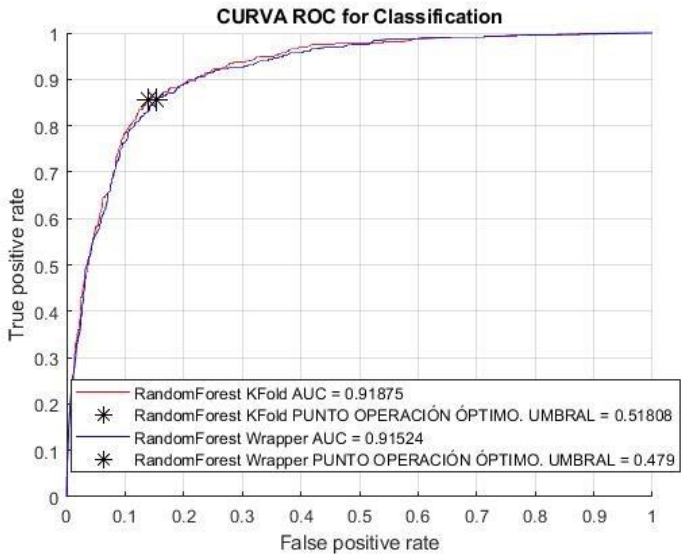


Figura 19. Curva ROC *Random Forest*. Fuente: elaboración propia

Cuando se utiliza el clasificador *Random Forest* se obtiene un mayor porcentaje de área bajo la curva cuando se entrena el conjunto de datos con el vector de predictores óptimo obtenido con la técnica 1 (basada en filtros). Este porcentaje es del 91,87%, frente al 91,52% cuando se utiliza la técnica 2 (método envolvente) de selección de predictores.

Hasta ahora hemos comentado el mejor modelo según cada técnica de clasificación, pero entre todas ellas, ¿cuál es el mejor? La tabla 9, que recoge cada porcentaje de *AUC* y el punto operativo óptimo, nos ayudará a decidir cuál, de los diez modelos, es el que mayor área bajo la curva presenta.

De todas las técnicas, *Random Forest* es la que devuelve mayor porcentaje de área bajo la curva, utilizando cualquiera de los dos predictores óptimos, el de la técnica 1 o el de la técnica 2. Pero entre ellos, *Random Forest* entrenado con el vector de predictores de la técnica 1 devuelve un porcentaje de área de curva mayor que en cualquier otro caso.

Seguido de esta técnica, máquinas de vector soporte lineal (*SVML*) presenta el tercer porcentaje de área bajo la curva mayor, del 90,54%. Tras este también sigue el clasificador máquina de vector soporte, pero no con *kernel lineal*, si no con *polinomial*. *SMVP* con el vector de predictores óptimo de la técnica 2 (*Wrapper*) es el cuarto mejor porcentaje de *AUC*, concretamente 89,77%.

Tabla 9. AUC y umbral óptimo de cada modelo. Fuente: elaboración propia

	Técnica de selección de predictores	Área bajo la curva	Umbral óptimo
SVML	Técnica 1: vector óptimo clasificado	90,54%	0,3624
	Técnica 2: vector óptimo *Wrapper*	88,75%	0,3484
SVMG	Técnica 1: vector óptimo clasificado	84,73%	0,2002
	Técnica 2: vector óptimo *Wrapper*	89,24%	0,2682
SVMP	Técnica 1: vector óptimo clasificado	88,19%	0,4292
	Técnica 2: vector óptimo *Wrapper*	89,77%	0,2623
Tree	Técnica 1: vector óptimo clasificado	83,86%	0,2857
	Técnica 2: vector óptimo *Wrapper*	83,4%	0,2
Random Forest	Técnica 1: vector óptimo clasificado	91,87%	0,5171
	Técnica 2: vector óptimo *Wrapper*	91,52%	0,479

El mejor modelo según el área bajo la curva se obtiene, por tanto, al utilizar el clasificador *Random Forest,* y el vector de predictores óptimo obtenido con la técnica 1. Recordamos que la técnica 1 consistía en obtener el vector de predictores clasificado con *MRMR* y, de entre todos los posibles subconjuntos de predictores de dimensión 1 a 17, quedarnos con el que mayor tasa de acierto reporte. Para este caso, el vector de predictores óptimo a utilizar sería [*pagevalues, weekend, exitrates, month, administrative, operatingsystems, productrelated_duration, visitortype, specialday, informational, bouncerates, traffictype, productrelated, adiministrative_duration, región, informational_duration*] de dimensión 16.

El siguiente paso es obtener las métricas de clasificación binaria en el punto de operación o umbral óptimo de cada modelo.

Cuando aplicamos el modelo, le pedimos también que nos devuelva las probabilidades de pertenencia de la clase negativa y de la positiva. A nosotros solo nos interesa la probabilidad obtenida por el clasificador para la clase positiva, la cual es recogida por la variable *scores*.

Cuando generamos la curva ROC de cada modelo obtuvimos varias variables, entre las que se encuentra el umbral óptimo de cada modelo (*TOP*). Este punto operativo será el que utilizaremos para umbralizar las predicciones de la clase positiva.

Llegados a este punto, nos queda calcular las métricas de clasificación binaria de esas predicciones. Todo el proceso desarrollado aparece en el Anexo 3 del presente estudio.

La tabla 10 recoge la tasa de acierto, sensibilidad y especificidad de todos los modelos.

Tabla 10. Métricas de clasificación binaria en el umbral óptimo de cada modelo. Fuente: elaboración propia

	Técnica de selección de predictores	Tasa de acierto	Sensibilidad	Especificidad
SVML	Técnica 1: vector óptimo clasificado	84,92%	83,57%	85,16%
	Técnica 2: vector óptimo *Wrapper*	84,7%	83,04%	85%
SVMG	Técnica 1: vector óptimo clasificado	87,65%	79,9%	89,06%
	Técnica 2: vector óptimo *Wrapper*	84,35%	84,62%	84,3%
SVMP	Técnica 1: vector óptimo clasificado	81,16%	83,92%	80,65%
	Técnica 2: vector óptimo *Wrapper*	85,13%	83,57%	85,42%
Tree	Técnica 1: vector óptimo clasificado	87,62%	77,8%	89,42%
	Técnica 2: vector óptimo *Wrapper*	87,84%	76,92%	89,83%
Random Forest	Técnica 1: vector óptimo clasificado	86,05%	85,49%	86,15%
	Técnica 2: vector óptimo *Wrapper*	84,75%	85,66%	84,59%

Para el mejor modelo según *AUC,* el punto de la curva que mejor compromiso tiene entre el acierto en la clase positiva (en nuestro caso, la clase "sesiones que generaron ingresos") y los falsos positivos, presenta valores de tasa de acierto, sensibilidad y especificidad del 86,05%, 85,49% y 86,15% respectivamente. Lo que significa que, el 86,05% de las muestras o instancias son clasificadas correctamente por el clasificador y que, el 86,15% de las instancias de la clase negativa, en nuestro caso *False*: no generan ingresos, son clasificadas correctamente como que no generan ingresos.

Por último, siguiendo con el mejor modelo según el área bajo la curva (*Random Forest*), hemos diseñado una tabla final con las métricas anteriores en distintos umbrales objetivos, es decir, en

distintos puntos representativos de la curva. Es como resumir la curva numéricamente a través de los valores de las métricas en distintos puntos de la curva. Se han elegido los mismos niveles de sensibilidad como de especificidad: 65-70-75-80-85-90-95.

La tabla 11 recoge el umbral correspondiente como resultado del nivel de especificidad objetivo y su sensibilidad.

Tabla 11. Niveles de especificidad objetivos y umbral. Fuente: elaboración propia

Especificidad objetivo	Sensibilidad	Umbral
65%	94,93%	0,22
70%	93,71%	0,2632
75%	92,13%	0,32
80%	88,99%	0,3963
85%	86,01%	0,4921
90%	78,67%	0,6333
95%	58,04%	0,7983

Y la tabla 12 recoge lo mismo que la 9, pero con los niveles de sensibilidad objetivos.

Tabla 12. Niveles de sensibilidad objetivos y umbral. Fuente: elaboración propia

Especificidad	Sensibilidad objetivo	Umbral
93,41%	65%	0,7467
91,88%	70%	0,7
91,01%	75%	0,6665
89,19%	80%	0,6033
86,38%	85%	0,52
78,61%	90%	0,376
64,6%	95%	0,2167

Una vez tengamos los niveles de sensibilidad y especificidad objetivos, debemos umbralizar las probabilidades de la clase positiva con dichos umbrales. Y, por último, obtener la tasa de acierto, sensibilidad y especificidad de dichos niveles.

Estas métricas aparecen en la tabla 13 y tabla 14, para niveles de especificidad y de sensibilidad objetivos, respectivamente.

Tabla 13. Métricas de los niveles de especificidad objetivos. Fuente: elaboración propia

Especificidad objetivo	Tasa de acierto	Sensibilidad
65%	69,56%	94,93%
70%	73,67%	93,71%
75%	77,67%	92,13%
80%	81,4%	88,99%
85%	85,16%	86,01%
90%	88,24%	78,67%
95%	89,29%	58,04%

Podemos comprobar que, a medida que el nivel de especificidad objetivo es más alto se obtiene una mayor tasa de aciertos. Si nos fijamos en los extremos de los niveles de especificidad objetivos, podemos decir que:

- Cuando el nivel de especificidad es de 65%, el 69,56% de las instancias son clasificadas correctamente por el clasificador *Random Forest*. Además, el clasificador es capaz de predecir correctamente solamente el desenlace del 65% de las sesiones que finalizaron sin compras, a costa de equivocarse únicamente en el 5% de las sesiones que finalizaron con compras.

- Pero cuando el nivel de especificidad objetivo es mayor, del 95%, el 89,29% de las instancias son clasificadas correctamente por la técnica de clasificación *Random Forest*. En este caso, el clasificador es capaz de predecir correctamente el desenlace de aproximadamente el 90% de las sesiones que finalizaron sin compras, a costa de equivocarse únicamente en torno al 21% de las sesiones que finalizaron con compras.

Estos números indican que el clasificador es capaz de predecir correctamente el desenlace de, aproximadamente, el 90% de las sesiones que finalizaron sin compra (2.814 sesiones de un total de 3.127, según nuestro conjunto de test), a costa de equivocarse en torno al 21% de las que finalizaron con compra. Dado el desbalanceo de las clases, este porcentaje supone tan solo 67 instancias.

Referente a los niveles de sensibilidad objetivos, a medida que ésta aumenta, disminuye la tasa de acierto.

En este caso, si nos fijamos en los extremos de los niveles de sensibilidad objetivos, obtenemos que:

- Cuando el nivel de sensibilidad es el más bajo: 65%, el 89,02% de las instancias son clasificadas correctamente por el clasificador Random Forest. Además, el clasificador es capaz de predecir correctamente el desenlace del 93%, aproximadamente, de las sesiones que finalizaron sin compras, a costa de equivocarse en el 35% de las sesiones que finalizaron con compras.

- Pero cuando el nivel de sensibilidad objetivo es mayor establecido: 95%, el 69,29% de las instancias son clasificadas correctamente por la técnica de clasificación Random Forest. En este caso, el clasificador es capaz de predecir correctamente el desenlace del 78% de las sesiones que finalizaron sin compras, a costa de equivocarse únicamente en el 5% de las sesiones que finalizaron con compras.

Tabla 14. Métricas de los niveles de sensibilidad objetivos.
Fuente: elaboración propia

Sensibilidad objetivo	Tasa de acierto	Especificidad
65%	89,02%	93,41%
70%	88,48%	91,88%
75%	88,54%	91,01%
80%	87,78%	89,19%
85%	86,19%	86,38%
90%	80,37%	78,61%

95%	69,29%	64,6%

3.6 Análisis de los resultados

A continuación, mostramos las conclusiones de los resultados obtenidos:

- Para cada clasificador hemos obtenido dos vectores de predictores óptimos, uno con cada técnica de selección de características empleada. Los vectores óptimos obtenidos con la técnica envolvente presentan una tasa de acierto mayor a los obtenidos basados en filtros. De hecho, el vector de predictor óptimo de todos los obtenidos se obtiene para el clasificador *Random Forest* y la técnica *Wrapper*. Este vector proporciona el 85,74% de las muestras clasificadas correctamente al utilizar las siguientes 13 variables: [*pagevalues, productrelated, month, bouncerates, informational_duration, specialday, traffictype, browser, informational, región, weekend, productrelated_duration, administrative*].

- Según el área bajo la curva (*AUC*), el mayor porcentaje, 91,87%, se obtiene también el clasificador *Random Forest* pero cuando emplea el vector óptimo clasificado con *MRMR*: [*pagevalues, weekend, exitrates, month, administrative, operatingsystems, productrelated_duration, visitortype, specialday, informational, bouncerates, traffictype, productrelated, administrative_duration, región, informational_duration*].

- El punto de la curva que mejor compromiso tiene entre el acierto de la clase positiva (sesiones que generaron ingresos) y los falsos positivos, presenta los siguientes valores de tasa de acierto, sensibilidad y especificidad 86,05%, 85,49% y 86,15%, respectivamente.

- Respecto a los niveles de sensibilidad y especificidad objetivos, podemos concluir diciendo que para niveles de especificidad objetivos elevados (95%) se obtiene que el modelo *Random Forest* anterior, clasifica correctamente el 89,3% de las instancias. Es decir, es un modelo capaz de predecir aproximadamente el 90% de las sesiones que finalizaron sin compra, a costa de equivocarse en torno al 21% de las que finalizaron con compra. Aunque dado el desbalanceo, este porcentaje solo supone 657 instancias.

4 Conclusiones

El objetivo de este trabajo se centraba en la predicción de la intención de compra de los usuarios *online*, para así poder tomar acciones en consecuencias sobre aquellos usuarios que tengan una probabilidad alta de no realizar la conversión. La base de datos empleada está formada por 12.330 sesiones, de forma que cada sesión pertenece a un usuario diferente, y 18 variables. Se dividió el conjunto de datos en entrenamiento (70%) y evaluación (30%), manteniendo la proporción de las clases minoritaria y, dado el desbalanceo de las clases existente, se balanceó mediante submuestreo aleatorio.

Los modelos no fueron entrenados con todas las variables disponibles, si no que los clasificadores utilizados (máquinas de vector soporte, árbol de decisión y bosque aleatorio) fueron configurados para trabajar con dos conjuntos de predictores óptimos obtenidos tras la aplicación de dos técnicas: basadas en filtros (*MRMR*) y métodos envolventes (*Wrapper*). En ambos, se aplicó validación cruzada con 10 particiones sobre el conjunto de entrenamiento ya balanceado.

La comparativa entre los modelos obtenidos se realizó mediante el área bajo la curva (*AUC*) obtenida de la curva ROC. El modelo con mejor rendimiento se obtuvo con *Random Forest* utilizando el vector de predictores óptimo obtenido mediante la técnica basado en filtros (*MRMR*), compuesto por 16 variables. Este porcentaje asciende a 91,87%.

Con respectos a los puntos de la curva, el umbral óptimo presenta que el 86,05% de las instancias son clasificadas correctamente (*Accuracy*) por el clasificador *Random Forest,* así como el 86,15% de las muestras de la clase negativa son clasificadas correctamente como tal (*Specificity*). Pero también encontramos otros puntos en la curva donde el clasificador puede trabajar, por ejemplo, con una tasa de acierto del 88,24%, una sensibilidad de 78,67% y una especificidad de 90%. El clasificador, por tanto, es capaz de predecir correctamente el 90%, aproximadamente, de las sesiones que finalizaron sin compra, a costa de un error en torno al 21% de las que si finalizaron con compras. Dado el desbalanceo de las clases, este porcentaje supone tan sólo 657 instancias.

Finalmente, *Random Forest* puede ser una herramienta efectiva para detectar a los clientes que no tienen intención de comprar, para así tomar acciones promocionales que ayuden a generar la conversión de compra.

5 Referencias

Albert TC, Goes PB, & Gupta A. (2004). A model for design and management of content and Interactivity of customer-centric web sites. *Mis Quarterly*, *28*(2), 161-182.

Alpaydin, E. (2014). *Introduction to machine learning* (3rd ed.). Cambridge: MIT Press.

Baati, K, & Mohsil, M. (2020). Real-time prediction of online shoppers' purchasing intention using Random Forest. *Artificial Intelligence Applications and Innovations*, *583*, 43-51.

Budnikas, G. (2015). *Computerised Recommendations on E-Transaction Finalisation by Means of Machine Learning.* Statistics in Transition.

Calvario, KC. (Abril de 2019). *Análisis de las técnicas de selección de características.* Obtenido de http://ri.uaemex.mx/handle/20.500.11799/99936

Carmona, EJ. (Noviembre de 2016). *ResearchGate.* Obtenido de https://www.researchgate.net/publication/263817587_Tutorial_sobre_Maquinas_de_Vectores_Soporte_SVM

Castillo, A. C., Cabrera, L., Chávez, M. d., & García, M. M. (2020). *Editorial Feijóo.* Obtenido de http://feijoo.cdict.uclv.edu.cu/?monografia=tecnicas-de-pre-procesamiento-en-conjuntos-de-datos-desbalanceados-para-mejorar-la-clasificacion

Chaitow, L. (2004). Accuracy. *Journal of Bodywork and Movement Therapies*, *8*, 158-159.

Chandrashekar, G, & Sahin, F. (2014). A survey on feature selection methods. *Computers & Electrical Engineering*, *40*, 16-28.

Cho CH, Kang J, & Cheon HJ. (2006). Online shopping hesitation. *CyberPsychology & Behavior*, *9*(3), 261-274.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letter*, *27*, 861-874.

González, L. (23 de Marzo de 2018). *aprendeIA*. Obtenido de https://aprendeia.com/aprendizaje-supervisado-random-forest-classification/

Heras, JM. (18 de Septiembre de 2020). *IArtificial.net*. Obtenido de https://www.iartificial.net/random-forest-bosque-aleatorio/

Kau AK, Tang YE, & Ghose S. (2003). Typology of online shoppers. *Journal of Consumer Marketing, 20*(2), 139-156.

Kurniawan, I, Abdussomad, Akbar, MF, Azis, MS, Saepudin, DF, & Tabrani, M. (2020). Improving the effectiveness of classification using the data level approach and feature selection techniques in online shoppers purchasing intention prediction. *Journal of Physics: Conference series, 1641*.

Mathworks Inc. (2021). *Statistics and Machine Learning Toolbox.* Obtenido de https://es.mathworks.com/help/stats/index.html?s_tid=CRUX_lftnav

Medina, R. F., & Ñique, C. I. (2017). Bosques aleatorios como extensión de los árboles de clasificación con los programas R y Python. *Dialnet*(10), 165-189.

Mobasher B, Dai H, Luo T, & Nakagawa M. (2002). Discovery and evaluation of aggregate usage profiles for web personalization. *Data Mining and Knowledge Discovery*, 61-82.

Moe WW. (2003). Buying, searching, or browsing: Differentiating between online shoppers using in-store navigational clickstream. *Journal of Consumer Psychology, 13*, 29-39.

Poggi, N, Moreno, T, Berral, JL, Gavaldà. R, & Torres, J. (2007). Web customer modeling for automated session prioritization on high traffic sites. *International Conference on User Modeling. Springer, 4511*, 450-454.

Rajamma RK, Paswan AK, & Hossain MM. (2009). ¿Por qué los compradores abandonan la cesta de la compra? Tiempo de espera percibido, riesgo e inconvenientes de la transacción. *Journal of Product ¬ Brand Management, 18*(3), 188-197.

Rodrigo, A. (Febrero de 2017). *RPubs.* Obtenido de https://rpubs.com/Joaquin_AR/255596

Rodrigo, AJ. (Octubre de 2020). *RPubs.* Obtenido de https://rpubs.com/Joaquin_AR/255596

Sakar, CO, Polat, SO, Katircioglu, M, & Kastro, Y. (2019). Real-time prediction of online shoppers' purchasing intention using multilayer perceptron and LSTM recurrent neural networks. *Neural Computing and Applications, 31*, 6893-6908.

Suchacka G, Skolimowska-Kulig, M, & Potempa, A. (2015). Classification of e-customer sessions based on support vector machine. *ECMS*, 594-600.

Suchacka, G, Skolimowska-Kulig, M, & Potempa, A. (2015). A k.nearest neighbors method for classifying user sessions in e-commerce scenario. *Journal of Telecommunications and Information Technology, 3*, 64-69.

Suchacka, G., & Chodak, G. (2017). Using association rules to assess purchase probability in online stores. *Information Systems and e-Business Management, 15*, 751-780.

Valdovinos, R. (23 de Junio de 2006). *TDX*. Obtenido de Tesis Doctorals en Xarxa: https://www.tdx.cat/handle/10803/10478#page=2

Anexo 1

Descripción de los datos

Contains data
  obs: 12,330
 vars: 18

variable name	storage type	display format	value label	variable label
administrative	byte	%8.0g		Administrative
administrativ~n	float	%9.0g		Administrative_Duration
informational	byte	%8.0g		Informational
informational~n	float	%9.0g		Informational_Duration
productrelated	int	%8.0g		ProductRelated
productrelate~n	float	%9.0g		ProductRelated_Duration
bouncerates	float	%9.0g		BounceRates
exitrates	float	%9.0g		ExitRates
pagevalues	float	%9.0g		PageValues
specialday	float	%9.0g		SpecialDay
month	str4	%9s		Month
operatingsyst~s	byte	%8.0g		OperatingSystems
browser	byte	%8.0g		Browser
region	byte	%8.0g		Region
traffictype	byte	%8.0g		TrafficType
visitortype	str17	%17s		VisitorType
weekend	str5	%9s		Weekend
revenue	str5	%9s		Revenue

Sorted by:
     Note: Dataset has changed since last saved.

Estadísticos descriptivos de las variables numéricas

Variable	Obs	Mean	Std. Dev.	Min	Max
administra~e	12,330	2.315166	3.321784	0	27
administra~n	12,330	80.81861	176.7791	0	3398.75
informatio~l	12,330	.5035685	1.270156	0	24
informatio~n	12,330	34.4724	140.7493	0	2549.375
productrel~d	12,330	31.73147	44.4755	0	705
productrel~n	12,330	1194.746	1913.669	0	63973.52
bouncerates	12,330	.0221914	.0484883	0	.2
exitrates	12,330	.0430728	.0485965	0	.2
pagevalues	12,330	5.889258	18.56844	0	361.7637
specialday	12,330	.0614274	.1989173	0	1
month	0				
operatings~s	12,330	2.124006	.9113248	1	8
browser	12,330	2.357097	1.717277	1	13
region	12,330	3.147364	2.401591	1	9
traffictype	12,330	4.069586	4.025169	1	20
visitortype	0				
weekend	0				
revenue	0				

Tabla de frecuencia de la variable de salida *revenue*

Revenue	Freq.	Percent	Cum.
FALSE	10,422	84.53	84.53
TRUE	1,908	15.47	100.00
Total	12,330	100.00	

Anexo 2

Anexo 2.1. Importación de las librerías

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy import stats
import warnings
warnings.filterwarnings('ignore') # Para evitar los molestos avisos.
%matplotlib inline
plt.style.use('bmh')
```

Anexo 2.2. Lectura del fichero

In [2]:
```
df = pd.read_csv('online_shoppers_intention.csv')
df.head()
```

Out[2]:

	Administrative	Administrative_Duration	Informational	Informational_Duration	ProductRelated	ProductRelated_Duration	BounceRates	ExitRates	PageValues
0	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
1	0	0.0	0	0.0	2	64.000000	0.00	0.10	0.0
2	0	0.0	0	0.0	1	0.000000	0.20	0.20	0.0
3	0	0.0	0	0.0	2	2.666667	0.05	0.14	0.0
4	0	0.0	0	0.0	10	627.500000	0.02	0.05	0.0

In [3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12330 entries, 0 to 12329
Data columns (total 18 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Administrative           12330 non-null  int64
 1   Administrative_Duration  12330 non-null  float64
 2   Informational            12330 non-null  int64
 3   Informational_Duration   12330 non-null  float64
 4   ProductRelated           12330 non-null  int64
 5   ProductRelated_Duration  12330 non-null  float64
 6   BounceRates              12330 non-null  float64
 7   ExitRates                12330 non-null  float64
 8   PageValues               12330 non-null  float64
 9   SpecialDay               12330 non-null  float64
 10  Month                    12330 non-null  object
 11  OperatingSystems         12330 non-null  int64
 12  Browser                  12330 non-null  int64
 13  Region                   12330 non-null  int64
 14  TrafficType              12330 non-null  int64
 15  VisitorType              12330 non-null  object
 16  Weekend                  12330 non-null  bool
 17  Revenue                  12330 non-null  bool
dtypes: bool(2), float64(7), int64(7), object(2)
memory usage: 1.5+ MB
```

In [4]:
```
df.columns
```

Out[4]:
```
Index(['Administrative', 'Administrative_Duration', 'Informational',
       'Informational_Duration', 'ProductRelated', 'ProductRelated_Duration',
       'BounceRates', 'ExitRates', 'PageValues', 'SpecialDay', 'Month',
       'OperatingSystems', 'Browser', 'Region', 'TrafficType', 'VisitorType',
       'Weekend', 'Revenue'],
      dtype='object')
```

Anexo 2.3. Valores perdidos

Como podemos comprobar no presenta valores perdidos ya que al aplicar isna().any() a nuestro fichero, nos devuelve en todas las características el valor de *False* que indica la inexistencia de valores perdidos.

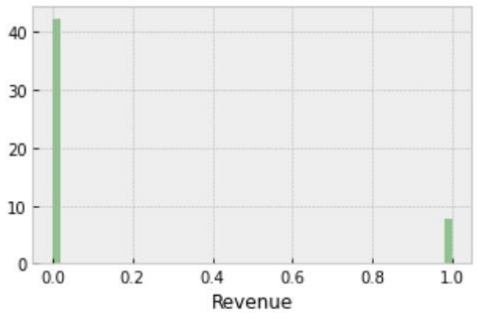
```
In [5]:
df.isna().any()
```

```
Out[5]:

Administrative             False
Administrative_Duration    False
Informational              False
Informational_Duration     False
ProductRelated             False
ProductRelated_Duration    False
BounceRates                False
ExitRates                  False
PageValues                 False
SpecialDay                 False
Month                      False
OperatingSystems           False
Browser                    False
Region                     False
TrafficType                False
VisitorType                False
Weekend                    False
Revenue                    False
dtype: bool
```

Anexo 2.4. Variable de salida *revenue*

```
In [6]:
print(df['Revenue'].describe())
plt.figure(figsize=(5,3))
sns.distplot(df['Revenue'], color='g');
```

```
count     12330
unique        2
top       False
freq      10422
Name: Revenue, dtype: object
```



Con respecto a la variable de salida *revenue* podemos comprar que presenta dos valores posibles (unique: 2) y que 10.422 instancias (freq) pertenecen a la clase *False* (top).

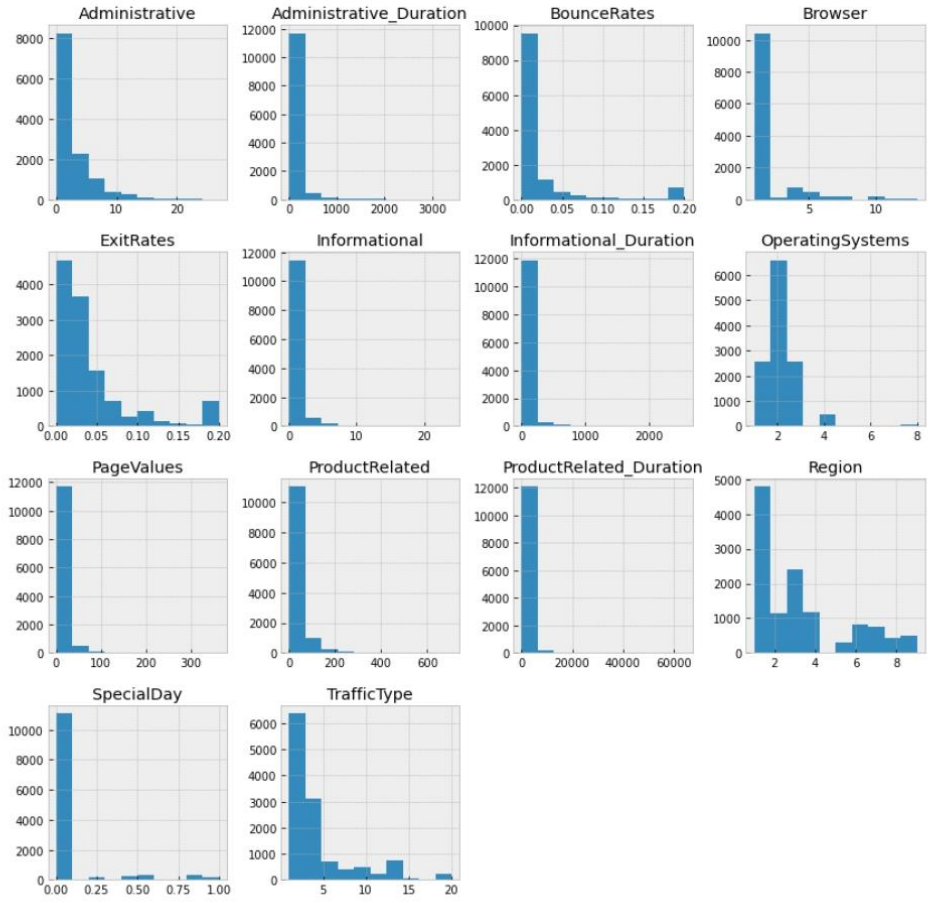
Anexo 2.5. Valores únicos de las variables discretas

```
In [8]:
for col in df.describe().columns:
    if df[col].dtype=='int64':
        unico=df[col].nunique()
        print(f'{col}:',unico)
```

```
Administrative: 27
Informational: 17
ProductRelated: 311
OperatingSystems: 8
Browser: 13
Region: 9
TrafficType: 20
```

Anexo 2.6. Histogramas de las variables numéricas y discretas

```
In [10]:
df[df.describe().columns].hist(figsize=(14,14));
```



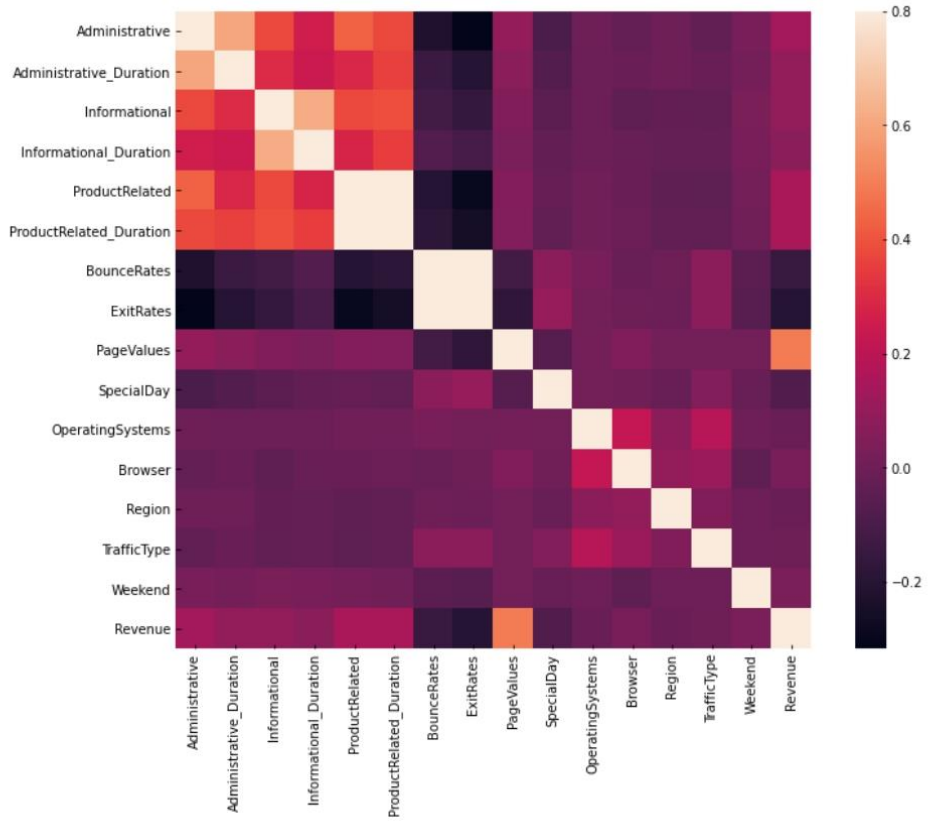
Como podemos comprobar, ninguna de las variables sigue una distribución normal.

Características que llaman la atención por su histograma: *specialday*. Pensaba que era una variable binaria de 0 y 1, pero se trata de una variable continua con 6 posibles valores. Cuanto más cercano al 1 sea e valor, más próximo a un día especial se encuentra.

```
In [11]:
df.SpecialDay.value_counts()
Out[11]:
0.0    11079
0.6      351
0.8      325
0.4      243
0.2      178
1.0      154
Name: SpecialDay, dtype: int64
```

Anexo 2.7. Correlación de las variables numéricas

```
In [14]:
corrmat = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corrmat, vmax=.8, square=True);
```



A primera vista hay dos variables que parecen estar muy correlacionadas: *bouncerates* y *exitrates*. Con respecto a las correlaciones de la variable de salida *revenue* destaca *pagevalues* como podemos comprobar en la matriz de correlación anterior.

A continuación, vemos el coeficiente de correlación de todas las variables con respecto a la variable de salida para conocer el coeficiente de la variable más relacionada: *pagevalues*.

In [16]:

```
corr = df.corr()
corr[['Revenue']].sort_values(by = 'Revenue',ascending = False).style.background_gradient()
```

Out[16]:

	Revenue
Revenue	1.000000
PageValues	0.492569
ProductRelated	0.158538
ProductRelated_Duration	0.152373
Administrative	0.138917
Informational	0.095200
Administrative_Duration	0.093587
Informational_Duration	0.070345
Weekend	0.029295
Browser	0.023984
TrafficType	-0.005113
Region	-0.011595
OperatingSystems	-0.014668
SpecialDay	-0.082305
BounceRates	-0.150673
ExitRates	-0.207071

También analizamos la correlación de las variables *bouncerates* y *exitrates*; *administrative* y *administrative_duration*; *informational* e *informational_duration; productrelated* y *productrelated_duration.* Cada una de ellas aparecen en los siguientes gráficos respectivamente.

Podemos ver como el coeficiente de correlación entre *bouncerates* y *exitrates* es de 0,91. Y su gráfico de dispersión muestra esa correlación fuerte y lineal.

In [17]:

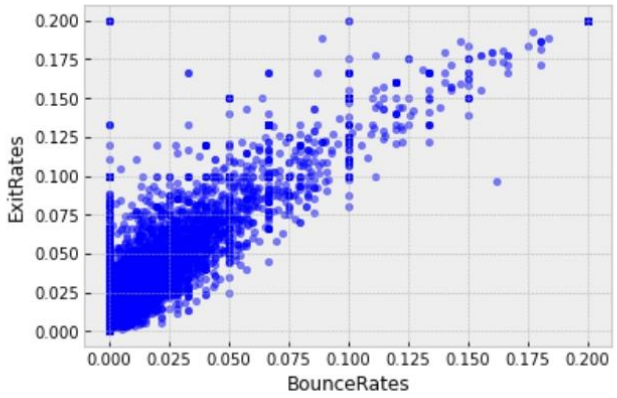
```
corr2 = df.corr()
corr2[['ExitRates']].sort_values(by = 'ExitRates',ascending = False).style.background_gradi
```

Out[17]:

	ExitRates
ExitRates	1.000000
BounceRates	0.913004
SpecialDay	0.102242
TrafficType	0.078616
OperatingSystems	0.014567
Browser	-0.004442
Region	-0.008907
Weekend	-0.062587
Informational_Duration	-0.105276
Informational	-0.163666
PageValues	-0.174498
Administrative_Duration	-0.205798
Revenue	-0.207071
ProductRelated_Duration	-0.251984
ProductRelated	-0.292526
Administrative	-0.316483

In [18]:

```
var = 'BounceRates'
data = pd.concat([df['ExitRates'], df[var]], axis=1)
data.plot.scatter(x=var, y='ExitRates', alpha = 0.5);
```



Con respecto a *administrative* y *administrative_duration*, el coeficiente de correlación es de 0,602.

In [23]:

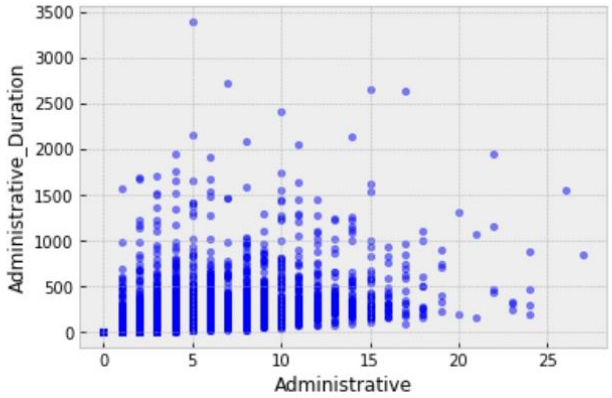
```
corr5 = df.corr()
corr5[['Administrative']].sort_values(by = 'Administrative',ascending = False).style.backgr
```

Out[23]:

	Administrative
Administrative	1.000000
Administrative_Duration	0.601583
ProductRelated	0.431119
Informational	0.376850
ProductRelated_Duration	0.373939
Informational_Duration	0.255848
Revenue	0.138917
PageValues	0.098990
Weekend	0.026417
Region	-0.005487
OperatingSystems	-0.006347
Browser	-0.025035
TrafficType	-0.033561
SpecialDay	-0.094778
BounceRates	-0.223563
ExitRates	-0.316483

In [24]:

```
var = 'Administrative'
data = pd.concat([df['Administrative_Duration'], df[var]], axis=1)
data.plot.scatter(x=var, y='Administrative_Duration', alpha = 0.5);
```



La correlación entre *informational* e *informational_duration* es de 0,619.

In [21]:

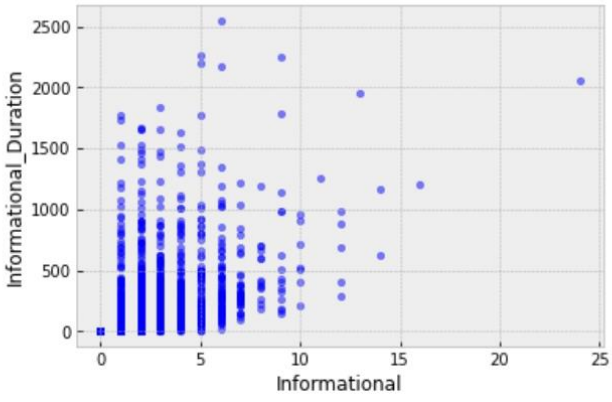
```
corr4 = df.corr()
corr4[['Informational']].sort_values(by = 'Informational',ascending = False).style.backgrou
```

Out[21]:

	Informational
Informational	1.000000
Informational_Duration	0.618955
ProductRelated_Duration	0.387505
Administrative	0.376850
ProductRelated	0.374164
Administrative_Duration	0.302710
Revenue	0.095200
PageValues	0.048632
Weekend	0.035785
OperatingSystems	-0.009527
Region	-0.029169
TrafficType	-0.034491
Browser	-0.038235
SpecialDay	-0.048219
BounceRates	-0.116114
ExitRates	-0.163666

In [22]:

```
var = 'Informational'
data = pd.concat([df['Informational_Duration'], df[var]], axis=1)
data.plot.scatter(x=var, y='Informational_Duration', alpha = 0.5);
```



Así como el coeficiente entre *productrelated* y *productrelated_duration* es de 0,861.

In [19]:

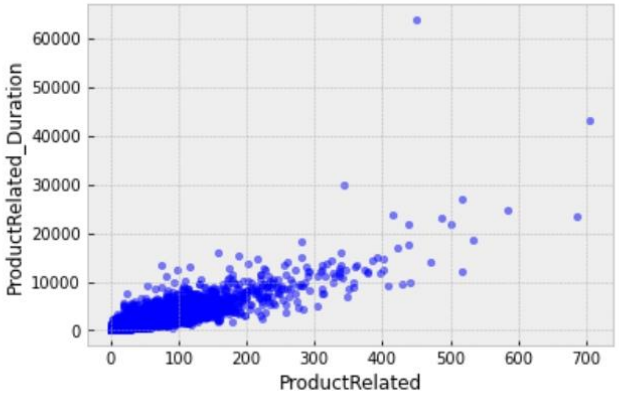
```
corr3 = df.corr()
corr3[['ProductRelated']].sort_values(by = 'ProductRelated',ascending = False).style.backgr
```

Out[19]:

	ProductRelated
ProductRelated	1.000000
ProductRelated_Duration	0.860927
Administrative	0.431119
Informational	0.374164
Administrative_Duration	0.289087
Informational_Duration	0.280046
Revenue	0.158538
PageValues	0.056282
Weekend	0.016092
OperatingSystems	0.004290
Browser	-0.013146
SpecialDay	-0.023958
Region	-0.038122
TrafficType	-0.043064
BounceRates	-0.204578
ExitRates	-0.292526

In [20]:

```
var = 'ProductRelated'
data = pd.concat([df['ProductRelated_Duration'], df[var]], axis=1)
data.plot.scatter(x=var, y='ProductRelated_Duration', alpha = 0.5);
```



Anexo 2.8. Gráfico de barras de las variables categóricas

In [25]:

```
non_num_cols = list(set(df.columns) - set(df.describe().columns))
print('Non-numerical features: ',non_num_cols)
```

```
Non-numerical features:  ['Revenue', 'Weekend', 'VisitorType', 'Month']
```

In [26]:

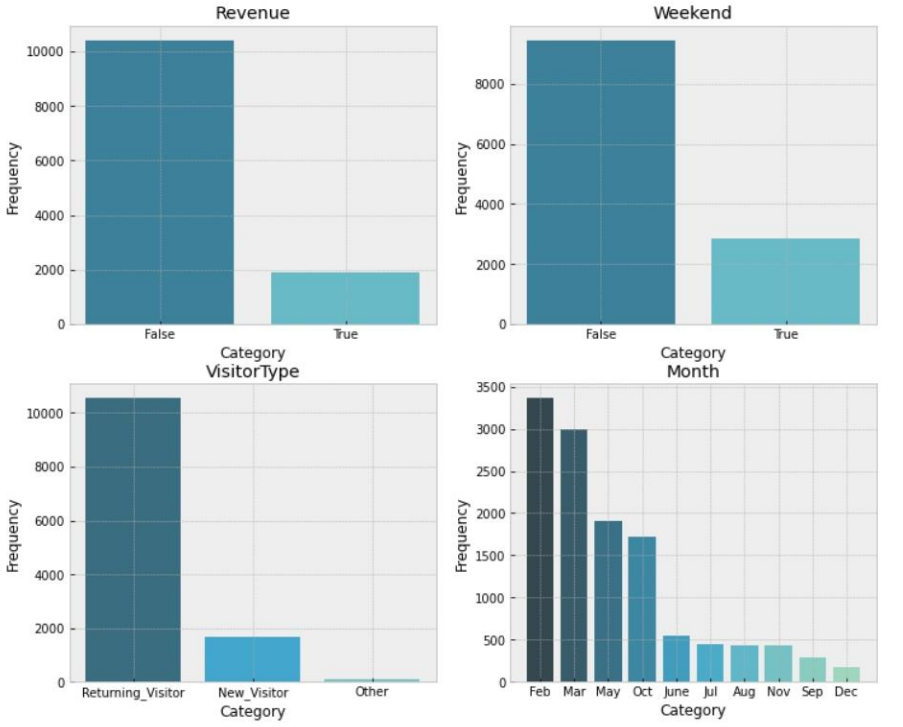
```
df.describe(include=['object','bool'])
```

Out[26]:

	Month	VisitorType	Weekend	Revenue
count	12330	12330	12330	12330
unique	10	3	2	2
top	May	Returning_Visitor	False	False
freq	3364	10551	9462	10422

In [27]:

```
fig, axs = plt.subplots(2, 2,figsize=(12,10))
# Gráfico de barras variables categóricas
for i, col in enumerate(non_num_cols):
    # Variables de ayuda para asignar los ejes correctos a cada uno de los 4 gráficos.
    j=i//2
    k=i%2
    color = sns.color_palette("GnBu_d",n_colors=len(df[col].unique()))

    axs[j,k].bar(df[col].astype('str').unique(), df[col].value_counts(), color= color)
    axs[j,k].set_title(col)
    axs[j,k].set_ylabel('Frequency')
    axs[j,k].set_xlabel('Category')
plt.show();
```



Anexo 2.9. Gráficos pares de variables

Weekend* – *revenue: la mayoría de los ingresos no se obtienen durante el fin de semana.

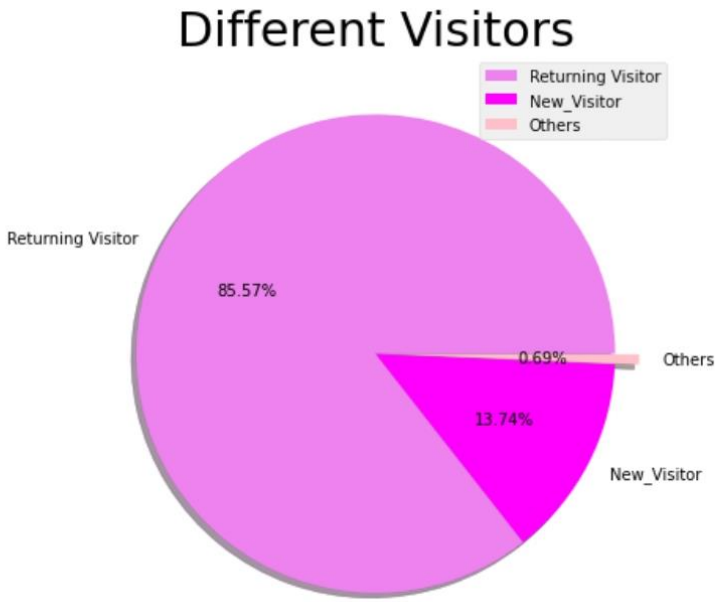
In [30]:

```
plt.rcParams['figure.figsize'] = (7, 7)
sns.countplot(data['Weekend'], palette = 'inferno')
plt.title('Purchase on Weekends', fontsize = 30)
plt.xlabel('Weekend or not', fontsize = 15)
plt.ylabel('count', fontsize = 15)

plt.show()
```

Visitortype* – *revenue: la mayoría de los ingresos (85,57%) se obtiene por los visitantes que vuelven, seguido de los visitantes nuevos (13,74%) y terminando con otros usuarios.

In [32]:

```
plt.rcParams['figure.figsize'] = (18, 7)
size = [10551, 1694, 85]
colors = ['violet', 'magenta', 'pink']
labels = "Returning Visitor", "New_Visitor", "Others"
explode = [0, 0, 0.1]
plt.pie(size, colors = colors, labels = labels, explode = explode, shadow = True, autopct =
plt.title('Different Visitors', fontsize = 30)
plt.axis('off')
plt.legend()
plt.show()
```



Anexo 3

Anexo 3.1. Lectura del fichero de datos

```
addpath("DatosEntrada")
addpath("Funciones")

load online_shoppers.mat
```

Anexo 3.2. Generación del conjunto de datos X-Y, definición de las variables representativas y estandarización del conjunto de datos

```
X = datosNum(:,1:end-1);
% X: me quedo con todas las filas y con todas las columnas menos la última,
% que es la variable de salida
Y = datosNum(:,end);
% Me quedo con todas las filas y con la última columna
varNamesX = varNames(1:end-1);
varNamesY = varNames(end);
[numDatos , numAtributos] = size(X);
valoresCodifClases = unique(Y);
numClases = length(valoresCodifClases);
nombreClases = cell(1, numClases);
nombreClases{1} = 'No generan ingresos';
nombreClases{2} = 'Generan ingresos';
% Podemos confirmar que el conjunto de datos está formado por 12330
% instancias y 17 predictores sin tener en cuenta la variable de salida
X=normalize(X);
```

Anexo 3.3. División del conjunto de datos en entrenamiento y validación

```
porcentajeTrain=70;

[XTrain, YTrain, XTest, YTest] = ...
         funcion_particiona_datos_holdout_v2(X, Y, porcentajeTrain);

% Consiste en dividir el conjunto de datos en 70% entrenamiento - 30%
% validación, de forma que en cada conjunto se cumpla la misma proporción
% de la clase minoritaria, es decir, que de todas las muestras del conjunto
% de entrenamiento, el 70% corresponda a la clase minoritaria, y el 30% en
% el conjunto de test

% El 70% de la clase minoritaria en XTrain es de 1336 instancias. Y el 30%
% de la clase minoritaria en XTest es de 572.

%  Xtrain - YTrain
% 8631x17 - 8631x1
% 8631 → 70% 12330 → 0.7 Clase Mayoritaria + 0.7 Clase Minoritaria = 7295 + 1336
```

```
%  Xtest  - YTest
% 3699x17 - 3699x1
% 3699 → 30% de 12330 → 0.3 Clase Mayoritaria + 0.3 Clase Minoritaria = 3127 + 572
```

Anexo 3.4. Submuestreo del conjunto de entrenamiento

```
% El submuestreo es una técnica que se emplea cuando los datos están muy
% desbalanceados, como es nuestro caso, y se aplica sobre el conjunto de
entrenamiento.
% Del total de muestras de entrenamiento, obtendríamos una muestra menor,
% ya que nos quedamos con todas las instancias de la clase minoritaria,
% en nuestro caso es que generan ingresos,
% y eliminamos las muestras de la clase mayoritaria hasta obtener el mismo número de
instancias.
% Es decir, hemos dicho anteriormente que el número de instancias de la
% clase minoritaria en XTrain era de 1336 (XTrain: 1336 + 7295 = 8631), por tanto
% el conjunto de entrenamiento que obtendríamos tras el submuestreo está
% formado por 1336 instancias de la clase minoritaria y 1336 instancias
% elegidas aleatoriamente entre las 7295 de la clase mayoritaria.
codifClaseMinoritaria = 2;
numDatos = sum(YTrain == codifClaseMinoritaria);
% suma de cuantos Y=2 hay en YTrain.

codifClaseMayoritaria = 1;
indicesClaseMayoritaria = find(YTrain==codifClaseMayoritaria);
% find devuelve un vector que contiene los ÍNDICES LINEALES de cada
% elemento de YTrain en los que YTrain=1

instanciasElegidasAleatoriamente = randsample(indicesClaseMayoritaria, numDatos);
% Devuelve 1336 valores entre los (7295x1)
XTrainSub = [XTrain(instanciasElegidasAleatoriamente,:) ; XTrain(YTrain==2,:)];
YTrainSub = [YTrain(instanciasElegidasAleatoriamente) ; YTrain(YTrain==2)];

% XTrainSub = (2672x17) → 1336 Clase Minoritaria + 1336 Clase Mayoritaria
% YTrainSub = (2672x1) → 1336 Clase Minoritaria + 1336 Clase Mayoritaria
```

Anexo 3.5. Clasificación de los predictores

```
%% Clasificar los predictores en función de su importancia
[idx,scores] = fscmrmr(XTrainSub,YTrainSub);
% fscmrmr clasifica los predictores en XTrain utilizando la variable de respuesta
YTrain
% - idx devuelve un vector de predictores con la mejor clasificación de los
% predictores, siendo el primero el mejor de todos.
% - scores devuelve un vector con las puntuaciones de los predictores. Un valor de
puntuación
% grande indica que el predictor correspondiente es importante.

% El orden del mejor al predictor es el siguiente:
% [ pagevalues, weekend, exitrates, month, administrative, ...
% operatingsystems, productrelated_duration, visitortype, ...
% specialday, informational, bouncerates, traffictype, productrelated, ...
```

```
% administrative_duration, region, informational_duration, browser ]

% idx = [9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13]

%% Crear un gráfico de barras de las puntuaciones de importancia de los predictores
bar(scores(idx))
xlabel('Predictor rank')
ylabel('Predictor importance score')

scores(idx)

% GRÁFICO - Anexo 4: Puntuaciones de los predictores clasificados

% La caída de puntuación entre el primer y el segundo predictor más importante es
muy grande.
% Otras caídas que podemos considerar significantes, en comparación con todas
% las caídas en general son del predictor del 3 al 4, así como del 4 al 5 y del 7 al
8.
% Las caídas restantes son relativamente pequeñas.
% - Las grandes caídas implican que el software confía en la selección del predictor
más importante y
% - los descensos pequeños indican que la diferencia entre la importancia del
predictor no es significativa.
```

Anexo 3.6. Máquina vector soporte con *kernel lineal*

```
%% Selección del vector de predictores óptimo
kFold=10;
[numDatos,numAtributos]=size(XTrainSub);
c=cvpartition(numDatos,'kFold',kFold);

C=1;
tipoClasificador='SVML';

%% TÉCNICA 1
% Vectores de atributos:
% 1
vectorPredictores=[9];
posPredCat=[];
[Acc_SVML_1, Se_SVML_1, Sp_SVML_1] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 2
vectorPredictores=[9, 17];
posPredCat=[2];
[Acc_SVML_2, Se_SVML_2, Sp_SVML_2] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 3
vectorPredictores=[9, 17, 8];
posPredCat=[2];
[Acc_SVML_3, Se_SVML_3, Sp_SVML_3] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 4
```

```
vectorPredictores=[9, 17, 8, 11];
posPredCat=[2, 4];
[Acc_SVML_4, Se_SVML_4, Sp_SVML_4] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 5
vectorPredictores=[9, 17, 8, 11, 1];
posPredCat=[2, 4];
[Acc_SVML_5, Se_SVML_5, Sp_SVML_5] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 6
vectorPredictores=[9, 17, 8, 11, 1, 12];
posPredCat=[2, 4];
[Acc_SVML_6, Se_SVML_6, Sp_SVML_6] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 7
vectorPredictores=[9, 17, 8, 11, 1, 12, 6];
posPredCat=[2, 4];
[Acc_SVML_7, Se_SVML_7, Sp_SVML_7] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 8
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16];
posPredCat=[2, 4, 8];
[Acc_SVML_8, Se_SVML_8, Sp_SVML_8] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 9
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10];
posPredCat=[2, 4, 8];
[Acc_SVML_9, Se_SVML_9, Sp_SVML_9] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 10
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3];
posPredCat=[2, 4, 8];
[Acc_SVML_10, Se_SVML_10, Sp_SVML_10] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 11
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7];
posPredCat=[2, 4, 8];
[Acc_SVML_11, Se_SVML_11, Sp_SVML_11] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 12
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15];
posPredCat=[2, 4, 8];
[Acc_SVML_12, Se_SVML_12, Sp_SVML_12] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 13
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5];
posPredCat=[2, 4, 8];
```

```
[Acc_SVML_13, Se_SVML_13, Sp_SVML_13] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 14
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2];
posPredCat=[2, 4, 8];
[Acc_SVML_14, Se_SVML_14, Sp_SVML_14] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 15
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14];
posPredCat=[2, 4, 8];
[Acc_SVML_15, Se_SVML_15, Sp_SVML_15] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 16
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4];
posPredCat=[2, 4, 8];
[Acc_SVML_16, Se_SVML_16, Sp_SVML_16] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 17
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13];
posPredCat=[2, 4, 8];
[Acc_SVML_17, Se_SVML_17, Sp_SVML_17] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)

% ------------------------------------------
% RESUMEN RESULTADOS vpSVML:
% ------------------------------------------
% Tasa de acierto:
%   1          2        3          4        5          6         7
% 0.7938    0.7938    0.7938    0.8091    0.8099    0.8091    0.8129
%   8          9        10         11        12         13        14
% 0.8121    0.8117    0.8144    0.8144    0.8155    0.8155    0.8166
%   15        16        17
% 0.8166    0.8170    0.8181

% Sensibilidad:
%   1          2        3          4        5          6         7
% 0.6611    0.6611    0.6611    0.6949    0.7008    0.7001    0.7283
%   8          9        10         11        12         13        14
% 0.7262    0.7254    0.7330    0.7330    0.7344    0.7345    0.7338
%   15        16        17
% 0.7354    0.7360    0.7375

% Especificidad:
%   1          2        3          4        5          6         7
% 0.9269    0.9269    0.9269    0.9238    0.9193    0.9186    0.8974
%   8          9        10         11        12         13        14
% 0.8982    0.8982    0.8960    0.8960    0.8967    0.8970    0.9000
%   15        16        17
% 0.8985    0.8985    0.8992

% El porcentaje más alto de aciertos es de 81,81% para el vector formado por
```

```
% todas las variables.
% vpSVML_KFold = [9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13];


%% TÉCNICA 2
% Wrapper o procedimiento de selección de forma escalonada hacia delante
% En este pasos se van a seleccionar vectores de atributos de cada dimensión
% posible (de 1 a 17) con el criterio de mayor rendimiento medido en
% términos de tasa de acierto según cada tipo de clasificador. La
% estrategia de entrenamiento-evaluación que se aplica es validación
% cruzada con 10 particiones, igual que en el caso anterior, pero con la
% diferencia de que ahora se utiliza el vector de predictores original:
% [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17] y no el
% vector de predictores clasificado.
C=1;
tipoClasificador='SVML';
indPredicCat=[11, 16, 17]; % Cuáles son las variables categóricas.

vectoresAtributosMaxAcc = cell(1,numAtributos);
valoresAccVectoresAtributosMaxAcc = zeros(1,numAtributos);
nombreVectoresAtributosMaxAcc = cell(1,numAtributos);
vectorAtributosMaxAcc = [];

for dim=1:numAtributos

    % La función genera el vector de dimensión dim de mayor Acc,
    % incorporando un predictor al vector obtenido en la iteración
    % de dimensión dim-1

    [vectorAtributosMaxAcc, Acc]...
        =funcion_genera_vectorAtr_haciaDelante_crossVal(...
            vectorAtributosMaxAcc, XTrainSub, YTrainSub, c, indPredicCat, ...
            tipoClasificador, C);

    vectoresAtributosMaxAcc{dim} = vectorAtributosMaxAcc';

    valoresAccVectoresAtributosMaxAcc(dim) = Acc;

    nombreVectoresAtributosMaxAcc{dim} = ...
        varNamesX(vectorAtributosMaxAcc)';
end

% vectoresAtributosMaxAcc --> guarda todos los vectores de todas las
% dimensiones
% valoresAccVectoresAtributosMaxAcc --> Acc de cada vector de cada
% dimensión
% Acc -->  Acc del último vector (dimensión=17)

% Representación gráfica de la tasa de errores
figure,
plot(1-valoresAccVectoresAtributosMaxAcc,'*-b')
xlabel('Dimension de vector de atributos')
ylabel('Tasa de Errores')
axis([0 numAtributos+1 0 0.5])
grid on
leyenda = cell('')
```

```
leyenda{1}=('SVML: 10-fold Cross-Validation búsqueda hacia adelante')
legend(leyenda)

% GRÁFICO – Anexo 4: Tasa de error de los vectores de dimensión 1 a 17 utilizando la
técnica Wrapper y SVML como clasificador

%Guardamos los resultados de mayor Acc
[AccMaxSVML, indiceAccMaxSVML]= max(valoresAccVectoresAtributosMaxAcc)
vectorPredictoresSVML = vectoresAtributosMaxAcc{indiceAccMaxSVML}
nombresVectorPredictoresSVML = varNamesX(vectorPredictoresSVML)'

%.....................................................................
% GUARDAMOS RESULTADO EN VARIABLE CON LA NOTACIÓN QUE SE UTILIZARÁ
%.....................................................................

vpSVML_wrapper = vectorPredictoresSVML;

% vpSVML_wrapper = [9, 5, 11, 8, 4, 15, 10, 12, 3, 14, 2, 17]

%% MODELO 1
%% Mejor vector según Acc utilizando K-Fold (K=10)
%% y el vector de atributos mejor clasificado por fscmrmr (vpSVML_KFold)


%% 1 - Entrenamiento del modelo en XTrainSub - YTrainSub
% ----------------------------------------------
vpSVML_KFold = [9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13];
varNamesX(vpSVML_KFold);

indPredicCat = [17, 11, 16]; % Contempla poder tratar con variables categóricas
C=1;
ZTrainSVML = XTrainSub(:,vpSVML_KFold);
posPredCatSVML = find(ismember(vpSVML_KFold,indPredicCat));
modelSVML = fitcsvm(ZTrainSVML, YTrainSub,...
                            'CategoricalPredictors',posPredCatSVML,...
                            'BoxConstraint',C,...
                            'KernelFunction','linear');


%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ----------------------------------------------
% Adaptación de la salida del modelo en el rango 0-1
modelSVML=fitPosterior(modelSVML)
% Aplicación
ZTestFilt = XTest(:,vpSVML_KFold);
[YTest_SVML, pSVML_KFold] = predict(modelSVML,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ----------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'SVML KFold','-*r','k*'};
scores = pSVML_KFold(:,indiceClasPos);
```

```
[SENS_SVML_KFold, FP_SVML_KFold, UMBRALES_SVML_KFold, TOP_SVML_KFold,
AUC_SVML_KFold, leyenda_SVML_KFold, X_SVML_Acc,Y_SVML_KFold, T_SVML_KFold,
OPTROCPT_SVML_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRÁFICO – Anexo 4: Curva ROC SVML con el vector de predictores óptimo clasificado

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pSVML_KFold(:,indiceClasPos);
YClasificador_Binaria=scores>=TOP_SVML_KFold;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformarla:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_SVML_KFold = tablaRes(:,[1 3 5])
```

```
%                          Acc         Se          Sp
% SVM Lineal K-Fold     0.84915     0.83566     0.85161


%% MODELO 2
%% Mejor vector según Acc utilizando el método Wrapper (con validación
%% cruzada con 10 particiones) y el vector de predictores original,
%% no el clasificado.
% ------------------------------------------


%% 1 - Entrenamiento del modelo en XTrainSub - YTrainSub
% ------------------------------------------
varNamesX(vpSVML_wrapper); % Utiliza todos los atributos

indPredicCat = [17]; % Contempla poder tratar con variables categóricas
C=1;
ZTrainSVML = XTrainSub(:,vpSVML_wrapper);
posPredCatSVML = find(ismember(vpSVML_wrapper,indPredicCat));
modelSVML = fitcsvm(ZTrainSVML, YTrainSub,...
                            'CategoricalPredictors',posPredCatSVML,...
                            'BoxConstraint',C,...
                            'KernelFunction','linear');


%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ------------------------------------------
% Adaptación de la salida del modelo en el rango 0-1
modelSVML=fitPosterior(modelSVML)
% Aplicación
ZTestFilt = XTest(:,vpSVML_wrapper);
[YTest_SVML, pSVML_wrapper] = predict(modelSVML,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'SVML Wrapper','-*r','k*'};
scores = pSVML_wrapper(:,indiceClasPos);
[SENS_SVML_wrapper, FP_SVML_wrapper, UMBRALES_SVML_wrapper, TOP_SVML_wrapper, ...
    AUC_SVML_wrapper, leyenda_SVML_wrapper, X_SVML_wrapper, Y_SVML_wrapper, ...
    T_SVML_wrapper, OPTROCPT_SVML_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4:Curva ROC SVML con el vector de predictores óptimo obtenido con
Wrapper

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
```

```
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pSVML_wrapper(:,indiceClasPos);

YClasificador_Binaria=scores>=TOP_SVML_wrapper;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo al umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_SVML_wrapper = tablaRes(:,[1 3 5])

%                         Acc         Se          Sp
% SVM Lineal Wrapper    0.84699     0.83042     0.85002


%% 5 - Comparación curva ROC ambos modelos
% ------------------------------------------
% Para poder comparar varias curvas ROC, infoGrafica debe tener leyenda
% Llamamos a la función con la opción de actualizar la leyenda - vamos
% a representar en la misma gráfica múltiples curvas de distintos
% clasificadores
```

```
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

leyenda = cell('');
infoGrafica={'SVML K-Fold','-r','r*', leyenda};
scores = pSVML_KFold(:,indiceClasPos);
[SENS_SVML_KFold, FP_SVML_KFold, UMBRALES_SVML_KFold, TOP_SVML_KFold,
AUC_SVML_KFold, ...
    leyenda, X_SVML_Acc,Y_SVML_KFold, T_SVML_KFold, OPTROCPT_SVML_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

infoGrafica={'SVML Wrapper','-b','b*', leyenda};
scores = pSVML_wrapper(:,indiceClasPos);
[SENS_SVML_wrapper, FP_SVML_wrapper, UMBRALES_SVML_wrapper, TOP_SVML_wrapper, ...
    AUC_SVML_wrapper, leyenda_SVML_wrapper, X_SVML_wrapper, Y_SVML_wrapper, ...
    T_SVML_wrapper, OPTROCPT_SVML_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);


%% 6 - Tabla comparativa métricas clasificación binaria de los umbrales
% ---------------------------------------------
% Creamos una tabla conjunta de las métricas Acc, Se y Sp obtenidas en los
% umbrales óptimo de cada modelo
tablaResCompletaTest_SVML = [tabla_umbral_SVML_KFold; ...
    tabla_umbral_SVML_wrapper];
tablaResCompletaTest_SVML.Properties.RowNames = {'SVM Lineal K-Fold', ...
    'SVM Lineal Wrapper'}

% ---------------------------------------------
% RESULTADOS:
% ---------------------------------------------
%                          Acc         Se          Sp
% SVM Lineal K-Fold     0.84915     0.83566     0.85161
% SVM Lineal Wrapper    0.84699     0.83042     0.85002

% Si utilizamos SVM Lineal, obtendremos una mejor tasa de acierto, de
% sensibilidad y de especificidad para el vector de atributos óptimo
% obtenido tras la clasificación
```

Anexo 3.7. Máquina vector soporte con *kernel gaussiano*

```
%% Selección del vector de predictores óptimo

%% TÉCNICA 1
% Vectores de atributos:
% 1
vectorPredictores=[9];
posPredCat=[];
[Acc_SVMG_1, Se_SVMG_1, Sp_SVMG_1] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 2
```

```
vectorPredictores=[9, 17];
posPredCat=[2];
[Acc_SVMG_2, Se_SVMG_2, Sp_SVMG_2] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 3
vectorPredictores=[9, 17, 8];
posPredCat=[2];
[Acc_SVMG_3, Se_SVMG_3, Sp_SVMG_3] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 4
vectorPredictores=[9, 17, 8, 11];
posPredCat=[2, 4];
[Acc_SVMG_4, Se_SVMG_4, Sp_SVMG_4] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 5
vectorPredictores=[9, 17, 8, 11, 1];
posPredCat=[2, 4];
[Acc_SVMG_5, Se_SVMG_5, Sp_SVMG_5] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 6
vectorPredictores=[9, 17, 8, 11, 1, 12];
posPredCat=[2, 4];
[Acc_SVMG_6, Se_SVMG_6, Sp_SVMG_6] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 7
vectorPredictores=[9, 17, 8, 11, 1, 12, 6];
posPredCat=[2, 4];
[Acc_SVMG_7, Se_SVMG_7, Sp_SVMG_7] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 8
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16];
posPredCat=[2, 4, 8];
[Acc_SVMG_8, Se_SVMG_8, Sp_SVMG_8] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 9
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10];
posPredCat=[2, 4, 8];
[Acc_SVMG_9, Se_SVMG_9, Sp_SVMG_9] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 10
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3];
posPredCat=[2, 4, 8];
[Acc_SVMG_10, Se_SVMG_10, Sp_SVMG_10] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 11
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7];
posPredCat=[2, 4, 8];
```

```
[Acc_SVMG_11, Se_SVMG_11, Sp_SVMG_11] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 12
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15];
posPredCat=[2, 4, 8];
[Acc_SVMG_12, Se_SVMG_12, Sp_SVMG_12] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 13
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5];
posPredCat=[2, 4, 8];
[Acc_SVMG_13, Se_SVMG_13, Sp_SVMG_13] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 14
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2];
posPredCat=[2, 4, 8];
[Acc_SVMG_14, Se_SVMG_14, Sp_SVMG_14] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 15
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14];
posPredCat=[2, 4, 8];
[Acc_SVMG_15, Se_SVMG_15, Sp_SVMG_15] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 16
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4];
posPredCat=[2, 4, 8];
[Acc_SVMG_16, Se_SVMG_16, Sp_SVMG_16] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 17
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13];
posPredCat=[2, 4, 8];
[Acc_SVMG_17, Se_SVMG_17, Sp_SVMG_17] = ...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador, C)

% ------------------------------------------
% RESUMEN RESULTADOS vpSVMG:
% ------------------------------------------
% Tasa de acierto:
%   1          2         3          4         5          6         7
% 0.8282    0.8275    0.8271    0.8106    0.8162    0.8039    0.8039
%   8          9         10         11        12         13        14
% 0.7941    0.7900    0.7803    0.7769    0.7687    0.7612    0.7530
%   15         16        17
% 0.7313    0.7290    0.7155

% Sensibilidad:
%   1          2         3          4         5          6         7
% 0.7705    0.7644    0.7636    0.8063    0.8081    0.8112    0.8213
%   8          9         10         11        12         13        14
% 0.8199    0.8131    0.8215    0.8200    0.8317    0.8394    0.8529
```

```
%   15        16        17
% 0.8648    0.8708    0.8733

% Especificidad:
%   1         2         3         4         5         6         7
% 0.8865    0.8910    0.8911    0.8155    0.8248    0.7969    0.7871
%   8         9        10        11        12        13        14
% 0.7686    0.7671    0.7402    0.7348    0.7065    0.6839    0.6543
%   15        16        17
% 0.5992    0.5887    0.5594

% El porcentaje más alto de aciertos que se ha obtenido es de 82,82% para
% el vector formado por un único predictor: 'pagevalues'.
% vpSVMG_KFold = [9]

%% TÉCNICA 2
% Wrapper o procedimiento de selección de forma escalonada hacia delante
% En este pasos se van a seleccionar vectores de atributos de cada dimensión
% posible (de 1 a 17) con el criterio de mayor rendimiento medido en
% términos de tasa de acierto según cada tipo de clasificador. La
% estrategia de entrenamiento-evaluación que se aplica es validación
% cruzada con 10 particiones, igual que en el caso anterior, pero con la
% diferencia de que ahora se utiliza el vector de predictores original:
% [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17] y no el
% vector de predictores clasificado.

C=1;
tipoClasificador='SVMG';
indPredicCat=[11, 16, 17]; % Cuáles son las variables categóricas.

vectoresAtributosMaxAcc = cell(1,numAtributos);
valoresAccVectoresAtributosMaxAcc = zeros(1,numAtributos);
nombreVectoresAtributosMaxAcc = cell(1,numAtributos);
vectorAtributosMaxAcc = [];

for dim=1:numAtributos

    % La función genera el vector de dimensión dim de mayor Acc,
    % incorporando un predictor al vector obtenido en la iteración
    % de dimensión dim-1

    [vectorAtributosMaxAcc, Acc]...
        =funcion_genera_vectorAtr_haciaDelante_crossVal(...
            vectorAtributosMaxAcc, XTrainSub, YTrainSub, c, indPredicCat, ...
            tipoClasificador, C);

    vectoresAtributosMaxAcc{dim} = vectorAtributosMaxAcc';

    valoresAccVectoresAtributosMaxAcc(dim) = Acc;

    nombreVectoresAtributosMaxAcc{dim} = ...
        varNamesX(vectorAtributosMaxAcc)';
end

% vectoresAtributosMaxAcc --> guarda todos los vectores de todas las
% dimensiones
```

```
% valoresAccVectoresAtributosMaxAcc --> Acc de cada vector de cada
% dimensión
% Acc -->  Acc del último vector (dimensión=17)

% Representación gráfica de la tasa de errores
figure,
plot(1-valoresAccVectoresAtributosMaxAcc,'*-b')
xlabel('Dimension de vector de atributos')
ylabel('Tasa de Errores')
axis([0 numAtributos+1 0 0.5])
grid on
leyenda = cell('')
leyenda{1}=('SVMG: 10-fold Cross-Validation búsqueda hacia adelante')
legend(leyenda)

% GRAFICO – Anexo 4: Tasa de error de los vectores de dimensión 1 a 17 utilizando la
técnica Wrapper y SVMG como clasificador

%Guardamos los resultados de mayor Acc
[AccMaxSVMG, indiceAccMaxSVMG]= max(valoresAccVectoresAtributosMaxAcc)
vectorPredictoresSVMG = vectoresAtributosMaxAcc{indiceAccMaxSVMG}
nombresVectorPredictoresSVMG = varNamesX(vectorPredictoresSVMG)'

%.....................................................................
% GUARDAMOS RESULTADO EN VARIABLE CON LA NOTACIÓN QUE SE UTILIZARÁ
%.....................................................................

vpSVMG_wrapper = vectorPredictoresSVMG;

% vpSVMG_wrapper = [9, 5, 11, 7]


%% MODELO 1
%% Mejor vector según Acc utilizando K-Fold (K=10)
%% y el vector de atributos mejor clasificado por fscmrmr (vpSVMG_KFold)
% ------------------------------------------

%% 1 - Entrenamiento del modelo en XTrainSub - YTrainSub
% ------------------------------------------
vpSVMG_KFold = [9]; varNamesX(vpSVMG_KFold);

indPredicCat = []; % Contempla poder tratar con variables categóricas
C=1;
ZTrainSVMG = XTrainSub(:,vpSVMG_KFold);
posPredCatSVMG = find(ismember(vpSVMG_KFold,indPredicCat));
modelSVMG = fitcsvm(ZTrainSVMG, YTrainSub,...
                            'CategoricalPredictors',posPredCatSVMG,...
                            'BoxConstraint',C,...
                            'KernelFunction','gaussian'); %'rbf'


%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ------------------------------------------
% Adaptación de la salida del modelo en el rango 0-1
modelSVMG=fitPosterior(modelSVMG)
% Aplicación
```

```
ZTestFilt = XTest(:,vpSVMG_KFold);
[YTest_SVMG, pSVMG_KFold] = predict(modelSVMG,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'SVMG KFold','-*r','k*'};
scores = pSVMG_KFold(:,indiceClasPos);
[SENS_SVMG_KFold, FP_SVMG_KFold, UMBRALES_SVMG_KFold, TOP_SVMG_KFold,
AUC_SVMG_KFold, leyenda_SVMG_KFold, X_SVMG_Acc,Y_SVMG_KFold, T_SVMG_KFold,
OPTROCPT_SVMG_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4: Curva ROC SVMG con el vector de predictores óptimo clasificado

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pSVMG_KFold(:,indiceClasPos);
YClasificador_Binaria=scores>=TOP_SVMG_KFold;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
```

```
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_SVMG_KFold = tablaRes(:,[1 3 5])

%                          Acc         Se          Sp
% SVM Gaussiano K-Fold   0.87645    0.79895     0.89063


%% MODELO 2
%% Mejor vector según Acc utilizando el método Wrapper (con validación
%% cruzada con 10 particiones) y el vector de predictores original,
%% no el clasificado.
% ------------------------------------------------


%% 1 - Entrenamiento del modelo en XTrainSub - YTrainSub
% ------------------------------------------------
varNamesX(vpSVMG_wrapper);

indPredicCat = [17]; % Contempla poder tratar con variables categóricas, al igual
que el arbol
C=1;
ZTrainSVMG = XTrainSub(:,vpSVMG_wrapper);
posPredCatSVMG = find(ismember(vpSVMG_wrapper,indPredicCat));
modelSVMG = fitcsvm(ZTrainSVMG, YTrainSub,...
                            'CategoricalPredictors',posPredCatSVMG,...
                            'BoxConstraint',C,...
                            'KernelFunction','gaussian'); %'rbf'


%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ------------------------------------------------
% Adaptación de la salida del modelo en el rango 0-1
modelSVMG=fitPosterior(modelSVMG)
% Aplicación
ZTestFilt = XTest(:,vpSVMG_wrapper);
[YTest_SVMG, pSVMG_wrapper] = predict(modelSVMG,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ------------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'SVMG Wrapper','-*r','k*'};
scores = pSVMG_wrapper(:,indiceClasPos);
```

```
[SENS_SVMG_wrapper, FP_SVMG_wrapper, UMBRALES_SVMG_wrapper, TOP_SVMG_wrapper, ...
    AUC_SVMG_wrapper, leyenda_SVMG_wrapper, X_SVMG_wrapper, Y_SVMG_wrapper, ...
    T_SVMG_wrapper, OPTROCPT_SVMG_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO -Anexo 4: Curva ROC SVMG con el vector de predictores óptimo obtenido con
Wrapper

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pSVMG_wrapper(:,indiceClasPos);

YClasificador_Binaria=scores>=TOP_SVMG_wrapper;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
```

```
    codifClasePos, false)
tabla_umbral_SVMG_wrapper = tablaRes(:,[1 3 5])

%                          Acc         Se          Sp
% SVM Gaussiano Wrapper  0.84347    0.84615     0.84298


%% 5 - Comparación curva ROC ambos modelos
% ------------------------------------------
% Para poder comparar varias curvas ROC, infoGrafica debe tener leyenda
% Llamamos a la función con la opción de actualizar la leyenda - vamos
% a representar en la misma gráfica múltiples curvas de distintos
% clasificadores
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

leyenda = cell('');
infoGrafica={'SVMG K-Fold','-r','r*', leyenda};
scores = pSVMG_KFold(:,indiceClasPos);
[SENS_SVMG_KFold, FP_SVMG_KFold, UMBRALES_SVMG_KFold, TOP_SVMG_KFold,
AUC_SVMG_KFold, ...
    leyenda, X_SVMG_Acc,Y_SVMG_KFold, T_SVMG_KFold, OPTROCPT_SVMG_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

infoGrafica={'SVMG Wrapper','-b','b*', leyenda};
scores = pSVMG_wrapper(:,indiceClasPos);
[SENS_SVMG_wrapper, FP_SVMG_wrapper, UMBRALES_SVMG_wrapper, TOP_SVMG_wrapper, ...
    AUC_SVMG_wrapper, leyenda, X_SVMG_wrapper, Y_SVMG_wrapper, ...
    T_SVMG_wrapper, OPTROCPT_SVMG_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);



%% 6 - Tabla comparativa métricas clasificación binaria de los umbrales
% ------------------------------------------
% Creamos una tabla conjunta de las métricas Acc, Se y Sp obtenidas en los
% umbrales óptimo de cada modelo
tablaResCompletaTest_SVMG = [tabla_umbral_SVMG_KFold; ...
    tabla_umbral_SVMG_wrapper];
tablaResCompletaTest_SVMG.Properties.RowNames = {'SVM Gaussiano K-Fold', ...
    'SVM Gaussiano Wrapper'}

% ------------------------------------------
% RESULTADOS:
% ------------------------------------------
%                          Acc         Se          Sp
% SVM Gaussiano K-Fold     0.87645  0.79895     0.89063
% SVM Gaussiano Wrapper    0.84347  0.84615     0.84298

% Si utilizamos SVM Gaussiano, obtendremos una mejor tasa de acierto y de
% especificidad para el vector de atributos óptimo obtenido tras la
% clasificación. Sin embargo, la mayor sensibilidad se obtiene para el
% vector de predictores óptimo obtenido con la técnica Wrapper
```

Anexo 3.8. Máquina vector soporte con *kernel polinomial*

```
%% Selección del vector de predictores óptimo
kFold=10;
[numDatos,numAtributos]=size(XTrainSub);
c=cvpartition(numDatos,'kFold',kFold);

C=1;
tipoClasificador='SVMP';

%% TÉCNICA 1
% Vectores de atributos:
% 1
vectorPredictores=[9];
posPredCat=[];
[Acc_SVMP_1, Se_SVMP_1, Sp_SVMP_1] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 2
vectorPredictores=[9, 17];
posPredCat=[2];
[Acc_SVMP_2, Se_SVMP_2, Sp_SVMP_2] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 3
vectorPredictores=[9, 17, 8];
posPredCat=[2];
[Acc_SVMP_3, Se_SVMP_3, Sp_SVMP_3] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 4
vectorPredictores=[9, 17, 8, 11];
posPredCat=[2, 4];
[Acc_SVMP_4, Se_SVMP_4, Sp_SVMP_4] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 5
vectorPredictores=[9, 17, 8, 11, 1];
posPredCat=[2, 4];
[Acc_SVMP_5, Se_SVMP_5, Sp_SVMP_5] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 6
vectorPredictores=[9, 17, 8, 11, 1, 12];
posPredCat=[2, 4];
[Acc_SVMP_6, Se_SVMP_6, Sp_SVMP_6] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 7
vectorPredictores=[9, 17, 8, 11, 1, 12, 6];
posPredCat=[2, 4];
[Acc_SVMP_7, Se_SVMP_7, Sp_SVMP_7] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 8
```

```
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16];
posPredCat=[2, 4, 8];
[Acc_SVMP_8, Se_SVMP_8, Sp_SVMP_8] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 9
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10];
posPredCat=[2, 4, 8];
[Acc_SVMP_9, Se_SVMP_9, Sp_SVMP_9] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 10
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3];
posPredCat=[2, 4, 8];
[Acc_SVMP_10, Se_SVMP_10, Sp_SVMP_10] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 11
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7];
posPredCat=[2, 4, 8];
[Acc_SVMP_11, Se_SVMP_11, Sp_SVMP_11] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 12
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15];
posPredCat=[2, 4, 8];
[Acc_SVMP_12, Se_SVMP_12, Sp_SVMP_12] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 13
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5];
posPredCat=[2, 4, 8];
[Acc_SVMP_13, Se_SVMP_13, Sp_SVMP_13] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 14
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2];
posPredCat=[2, 4, 8];
[Acc_SVMP_14, Se_SVMP_14, Sp_SVMP_14] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 15
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14];
posPredCat=[2, 4, 8];
[Acc_SVMP_15, Se_SVMP_15, Sp_SVMP_15] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 16
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4];
posPredCat=[2, 4, 8];
[Acc_SVMP_16, Se_SVMP_16, Sp_SVMP_16] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador,C)
% 17
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13];
posPredCat=[2, 4, 8];
```

```
[Acc_SVMP_17, Se_SVMP_17, Sp_SVMP_17] = ...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador, C)

% ------------------------------------------
% RESUMEN RESULTADOS vpSVML:
% ------------------------------------------

% Tasa de acierto:
%   1          2         3          4         5          6         7
% 0.8192    0.8192    0.8181    0.8110    0.8203    0.8151    0.8215
%   8          9         10         11        12         13        14
% 0.8039    0.8009    0.7945    0.7900    0.7777    0.7736    0.7601
%   15         16        17
% 0.7504    0.7481    0.732

% Sensibilidad:
%   1          2         3          4         5          6         7
% 0.7366    0.7373    0.7373    0.8057    0.7805    0.7926    0.8197
%   8          9         10         11        12         13        14
% 0.8026    0.7980    0.7776    0.7788    0.7789    0.7741    0.7602
%   15         16        17
% 0.7571    0.7598    0.7478

% Especificidad:
%   1          2         3          4         5          6         7
% 0.9022    0.9014    0.8992    0.8167    0.8615    0.8371    0.8233
%   8          9         10         11        12         13        14
% 0.8050    0.8037    0.8114    0.8004    0.7761    0.7723    0.7589
%   15         16        17
% 0.7434    0.7359    0.7156


% El mayor porcentaje de aciertos se obtiene para el vector de predictores
% formado por siete variables (82,15%).
% vpSVMP_KFold = [9, 17, 8, 11, 1, 12, 6]



%% TÉCNICA 2
% Wrapper o procedimiento de selección de forma escalonada hacia delante
% En este pasos se van a seleccionar vectores de atributos de cada dimensión
% posible (de 1 a 17) con el criterio de mayor rendimiento medido en
% términos de tasa de acierto según cada tipo de clasificador. La
% estrategia de entrenamiento-evaluación que se aplica es validación
% cruzada con 10 particiones
C=1;
tipoClasificador='SVMP';
indPredicCat=[11, 16, 17]; % Cuáles son las variables categóricas.

vectoresAtributosMaxAcc = cell(1,numAtributos);
valoresAccVectoresAtributosMaxAcc = zeros(1,numAtributos);
nombreVectoresAtributosMaxAcc = cell(1,numAtributos);
vectorAtributosMaxAcc = [];

for dim=1:numAtributos
```

```
    % La función genera el vector de dimensión dim de mayor Acc,
    % incorporando un predictor al vector obtenido en la iteración
    % de dimensión dim-1

    [vectorAtributosMaxAcc, Acc]...
        =funcion_genera_vectorAtr_haciaDelante_crossVal(...
            vectorAtributosMaxAcc, XTrainSub, YTrainSub, c, indPredicCat, ...
            tipoClasificador, C)

    vectoresAtributosMaxAcc{dim} = vectorAtributosMaxAcc';

    valoresAccVectoresAtributosMaxAcc(dim) = Acc;

    nombreVectoresAtributosMaxAcc{dim} = ...
        varNamesX(vectorAtributosMaxAcc)';
end

% vectoresAtributosMaxAcc --> guarda todos los vectores de todas las
% dimensiones
% valoresAccVectoresAtributosMaxAcc --> Acc de cada vector de cada
% dimensión
% Acc --> Acc del último vector (dimensión=17)

% Representación gráfica de la tasa de errores
figure,
plot(1-valoresAccVectoresAtributosMaxAcc,'*-b')
xlabel('Dimension de vector de atributos')
ylabel('Tasa de Errores')
axis([0 numAtributos+1 0 0.5])
grid on
leyenda = cell('')
leyenda{1}=('SVMP: 10-fold Cross-Validation búsqueda hacia adelante')
legend(leyenda)

% GRAFICO – Anexo 4: Tasa de error de los vectores de dimensión 1 a 17 utilizando la
técnica Wrapper y SVMP como clasificador

%Guardamos los resultados de mayor Acc
[AccMaxSVMP, indiceAccMaxSVMP]= max(valoresAccVectoresAtributosMaxAcc)
vectorPredictoresSVMP = vectoresAtributosMaxAcc{indiceAccMaxSVMP}
nombresVectorPredictoresSVMP = varNamesX(vectorPredictoresSVMP)'

%.....................................................................
% GUARDAMOS RESULTADO EN VARIABLE CON LA NOTACIÓN QUE SE UTILIZARÁ
%.....................................................................

vpSVMP_wrapper = vectorPredictoresSVMP;

% vpSVMP_wrapper = [9, 5, 11, 3]


%% MODELO 1
%% Mejor vector según Acc utilizando K-Fold (K=10)
%% y el vector de atributos mejor clasificado por fscmrmr (vpSVMP_KFold)
% ------------------------------------------
```

```
%% 1 - Entrenamiento del modelo en XTrainSub - YTrainSub
% ------------------------------------------
vpSVMP_KFold = [9, 17, 8, 11, 1, 12, 6]; varNamesX(vpSVMP_KFold); % Utiliza todos
los atributos

indPredicCat = [17, 11]; % Contempla poder tratar con variables categóricas, al
igual que el arbol
C=1;
ZTrainSVMP = XTrainSub(:,vpSVMP_KFold);
posPredCatSVMP = find(ismember(vpSVMP_KFold,indPredicCat));
modelSVMP = fitcsvm(ZTrainSVMP, YTrainSub,...
                            'CategoricalPredictors',posPredCatSVMP,...
                            'BoxConstraint',C,...
                            'PolynomialOrder',3,...
                            'KernelFunction','polynomial');



%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ------------------------------------------
% Adaptación de la salida del modelo en el rango 0-1
modelSVMP=fitPosterior(modelSVMP);
% Aplicación
ZTestFilt = XTest(:,vpSVMP_KFold);
[YTest_SVMP, pSVMP_KFold] = predict(modelSVMP,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'SVMP KFold','-*r','k*'};
scores = pSVMP_KFold(:,indiceClasPos);
[SENS_SVMP_KFold, FP_SVMP_KFold, UMBRALES_SVMP_KFold, TOP_SVMP_KFold,
AUC_SVMP_KFold, leyenda_SVMP_KFold, X_SVMP_KFold,Y_SVMP_KFold, T_SVMP_KFold,
OPTROCPT_SVMP_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4: Curva ROC SVMP con el vector de predictores óptimo clasificado

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
```

```
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pSVMP_KFold(:,indiceClasPos);
YClasificador_Binaria=scores>=TOP_SVMP_KFold;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_SVMP_KFold = tablaRes(:,[1 3 5])

%                            Acc         Se          Sp
% SVM Polinomial K-Fold     0.81157     0.83916     0.80652



%% MODELO 2
%% Mejor vector según Acc utilizando el método Wrapper (con validación
%% cruzada con 10 particiones) y el vector de predictores original,
%% no el clasificado.
% ------------------------------------------


%% 1 - Entrenamiento del modelo en XTrainSub - YTrainSub
% ------------------------------------------
varNamesX(vpSVMP_wrapper) % Utiliza todos los atributos

indPredicCat = []; % Contempla poder tratar con variables categóricas, al igual que
el arbol
C=1;
```

```
ZTrainSVMP = XTrainSub(:,vpSVMP_wrapper);
posPredCatSVMP = find(ismember(vpSVMP_wrapper,indPredicCat));
modelSVMP = fitcsvm(ZTrainSVMP, YTrainSub,...
                            'CategoricalPredictors',posPredCatSVMP,...
                            'BoxConstraint',C,...
                            'PolynomialOrder',3,...
                            'KernelFunction','polynomial');


%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ---------------------------------------------
% Adaptación de la salida del modelo en el rango 0-1
modelSVMP=fitPosterior(modelSVMP)
% Aplicación
ZTestFilt = XTest(:,vpSVMP_wrapper);
[YTest_SVMP, pSVMP_wrapper] = predict(modelSVMP,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ---------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'SVMP Wrapper','-*r','k*'};
scores = pSVMP_wrapper(:,indiceClasPos);
[SENS_SVMP_wrapper, FP_SVMP_wrapper, UMBRALES_SVMP_wrapper, TOP_SVMP_wrapper, ...
    AUC_SVMP_wrapper, leyenda_SVMP_wrapper, X_SVMP_wrapper, Y_SVMP_wrapper, ...
    T_SVMP_wrapper, OPTROCPT_SVMP_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4: Curva ROC SVMP con el vector de predictores óptimo obtenido con
Wrapper

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
```

```
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pSVMP_wrapper(:,indiceClasPos);

YClasificador_Binaria=scores>=TOP_SVMP_wrapper;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_SVMP_wrapper = tablaRes(:,[1 3 5])

%                             Acc         Se          Sp
% SVM Polinomial Wrapper    0.85131     0.83566     0.85417

%% 5 - Comparación curva ROC ambos modelos
% ------------------------------------------------
% Para poder comparar varias curvas ROC, infoGrafica debe tener leyenda
% Llamamos a la función con la opción de actualizar la leyenda - vamos
% a representar en la misma gráfica múltiples curvas de distintos
% clasificadores
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

leyenda = cell('');
infoGrafica={'SVMP KFold','-r','r*', leyenda};
scores = pSVMP_KFold(:,indiceClasPos);
[SENS_SVMP_KFold, FP_SVMP_KFold, UMBRALES_SVMP_KFold, TOP_SVMP_KFold,
AUC_SVMP_KFold, ...
    leyenda, X_SVMP_KFold,Y_SVMP_KFold, T_SVMP_KFold, OPTROCPT_SVMP_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

infoGrafica={'SVMG Wrapper','-b','b*', leyenda};
scores = pSVMP_wrapper(:,indiceClasPos);
[SENS_SVMP_wrapper, FP_SVMP_wrapper, UMBRALES_SVMP_wrapper, TOP_SVMP_wrapper, ...
    AUC_SVMP_wrapper, leyenda, X_SVMP_wrapper, Y_SVMP_wrapper, ...
    T_SVMP_wrapper, OPTROCPT_SVMP_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);
```

```
%% 6 - Tabla comparativa métricas clasificación binaria de los umbrales
% ------------------------------------------------
% Creamos una tabla conjunta de las métricas Acc, Se y Sp obtenidas en los
% umbrales óptimo de cada modelo
tablaResCompletaTest_SVMP = [tabla_umbral_SVMP_KFold; ...
    tabla_umbral_SVMP_wrapper];
tablaResCompletaTest_SVMP.Properties.RowNames = {'SVM Polinomial K-Fold', ...
    'SVM Polinomial Wrapper'}

Ruta_Nombre = 'DatosGenerados\Metricas_umbral_SVMP.mat';
save(Ruta_Nombre, 'tablaResCompletaTest_SVMP')

% ------------------------------------------------
% RESULTADOS:
% ------------------------------------------------
%                             Acc         Se          Sp
% SVM Polinomial K-Fold     0.81157     0.83916     0.80652
% SVM Polinomial Wrapper    0.85131     0.83566     0.85417

% Si utilizamos SVM Polinomial, obtendremos una mejor tasa de acierto y de
% especificidad para el vector de atributos óptimo obtenido con la técnica
% Wrapper. Sin embargo, la mayor sensibilidad la encontramos para el vector
% de predictores óptimo según vector clasificado.
```

Anexo 3.9. Árbol de decisión

```
%% Selección del vector de predictores óptimo
tipoClasificador='Tree';

%% TÉCNICA 1
% Vectores de atributos:
% 1
vectorPredictores=[9];
posPredCat=[];
[Acc_Tree_1, Se_Tree_1, Sp_Tree_1] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 2
vectorPredictores=[9, 17];
posPredCat=[2];
[Acc_Tree_2, Se_Tree_2, Sp_Tree_2] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 3
vectorPredictores=[9, 17, 8];
posPredCat=[2];
[Acc_Tree_3, Se_Tree_3, Sp_Tree_3] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 4
vectorPredictores=[9, 17, 8, 11];
posPredCat=[2, 4];
```

```
[Acc_Tree_4, Se_Tree_4, Sp_Tree_4] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 5
vectorPredictores=[9, 17, 8, 11, 1];
posPredCat=[2, 4];
[Acc_Tree_5, Se_Tree_5, Sp_Tree_5] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 6
vectorPredictores=[9, 17, 8, 11, 1, 12];
posPredCat=[2, 4];
[Acc_Tree_6, Se_Tree_6, Sp_Tree_6] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 7
vectorPredictores=[9, 17, 8, 11, 1, 12, 6];
posPredCat=[2, 4];
[Acc_Tree_7, Se_Tree_7, Sp_Tree_7] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 8
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16];
posPredCat=[2, 4, 8];
[Acc_Tree_8, Se_Tree_8, Sp_Tree_8] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 9
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10];
posPredCat=[2, 4, 8];
[Acc_Tree_9, Se_Tree_9, Sp_Tree_9] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 10
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3];
posPredCat=[2, 4, 8];
[Acc_Tree_10, Se_Tree_10, Sp_Tree_10] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 11
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7];
posPredCat=[2, 4, 8];
[Acc_Tree_11, Se_Tree_11, Sp_Tree_11] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 12
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15];
posPredCat=[2, 4, 8];
[Acc_Tree_12, Se_Tree_12, Sp_Tree_12] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 13
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5];
posPredCat=[2, 4, 8];
[Acc_Tree_13, Se_Tree_13, Sp_Tree_13] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
```

```
                        YTrainSub,c,posPredCat,tipoClasificador)
% 14
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2];
posPredCat=[2, 4, 8];
[Acc_Tree_14, Se_Tree_14, Sp_Tree_14] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 15
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14];
posPredCat=[2, 4, 8];
[Acc_Tree_15, Se_Tree_15, Sp_Tree_15] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 16
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4];
posPredCat=[2, 4, 8];
[Acc_Tree_16, Se_Tree_16, Sp_Tree_16] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 17
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13];
posPredCat=[2, 4, 8];
[Acc_Tree_17, Se_Tree_17, Sp_Tree_17] = ...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)

% ------------------------------------------
% RESUMEN RESULTADOS vpTree:
% ------------------------------------------
% Tasa de acierto:
%   1          2        3          4        5          6         7
% 0.8039    0.8076    0.7777    0.7855    0.7878    0.7945    0.7945
%   8          9        10         11        12         13        14
% 0.7964    0.7971    0.7956    0.7998    0.8054    0.8013    0.7997
%   15         16        17
% 0.8024    0.7979    0.8035

% Sensibilidad:
%   1          2        3          4        5          6         7
% 0.7211    0.7276    0.7525    0.7841    0.7861    0.7950    0.7943
%   8          9        10         11        12         13        14
% 0.7936    0.7943    0.7959    0.8030    0.8144    0.8137    0.7970
%   15         16        17
% 0.8002    0.7961    0.8015

% Especificidad:
%   1          2        3          4        5          6         7
% 0.8878    0.8886    0.8028    0.7873    0.7894    0.7939    0.7944
%   8          9        10         11        12         13        14
% 0.7988    0.7996    0.7955    0.7971    0.7970    0.7886    0.8023
%   15         16        17
% 0.8045    0.7992    0.8052


% El porcentaje más alto de especificidad que se ha obtenido es de 80,76%,
% para el vector de predictores formado por dos variables.
```

```
% vpTree_KFold = [9, 17]


%% TÉCNICA 2
% Wrapper o procedimiento de selección de forma escalonada hacia delante
% En este pasos e van a seleccionar vectores de atributos de cada dimensión
% posible (de 1 a 17) con el criterio de mayor rendimiento medido en
% términos de tasa de acierto según cada tipo de clasificador. La
% estrategia de entrenamiento-evaluación que se aplica es validación
% cruzada con 10 particiones, igual que en el caso anterior, pero con la
% diferencia de que ahora se utiliza el vector de predictores original:
% [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17] y no el
% vector de predictores clasificado.
tipoClasificador='Tree';
indPredicCat=[11, 16, 17]; % Cuáles son las variables categóricas.

vectoresAtributosMaxAcc = cell(1,numAtributos);
valoresAccVectoresAtributosMaxAcc = zeros(1,numAtributos);
nombreVectoresAtributosMaxAcc = cell(1,numAtributos);
vectorAtributosMaxAcc = [];

for dim=1:numAtributos

    % La función genera el vector de dimensión dim de mayor Acc,
    % incorporando un predictor al vector obtenido en la iteración
    % de dimensión dim-1

    [vectorAtributosMaxAcc, Acc]...
        =funcion_genera_vectorAtr_haciaDelante_crossVal(...
            vectorAtributosMaxAcc, XTrainSub, YTrainSub, c, indPredicCat, ...
            tipoClasificador);

    vectoresAtributosMaxAcc{dim} = vectorAtributosMaxAcc';

    valoresAccVectoresAtributosMaxAcc(dim) = Acc;

    nombreVectoresAtributosMaxAcc{dim} = ...
        varNamesX(vectorAtributosMaxAcc)';
end

% vectoresAtributosMaxAcc --> guarda todos los vectores de todas las
% dimensiones
% valoresAccVectoresAtributosMaxAcc --> Acc de cada vector de cada
% dimensión
% Acc -->  Acc del último vector (dimensión=17)

% Representación gráfica de la tasa de errores
figure,
plot(1-valoresAccVectoresAtributosMaxAcc,'*-b')
xlabel('Dimension de vector de atributos')
ylabel('Tasa de Errores')
axis([0 numAtributos+1 0 0.5])
grid on
leyenda = cell('')
leyenda{1}=('Tree: 10-fold Cross-Validation búsqueda hacia adelante')
legend(leyenda)
```

```
% GRAFICO -Anexo 4: Tasa de error de los vectores de dimensión 1 a 17 utilizando la
técnica Wrapper y Tree como clasificador

%Guardamos los resultados de mayor Acc
[AccMaxTree, indiceAccMaxTree]= max(valoresAccVectoresAtributosMaxAcc)
vectorPredictoresTree = vectoresAtributosMaxAcc{indiceAccMaxTree}
nombresVectorPredictoresTree = varNamesX(vectorPredictoresTree)'

%.....................................................................
% GUARDAMOS RESULTADO EN VARIABLE CON LA NOTACIÓN QUE SE UTILIZARÁ
%.....................................................................

vpTree_wrapper = vectorPredictoresTree;

% vpTree_wrapper = [9, 15, 17]


%% MODELO 1
%% Mejor vector según Acc utilizando K-Fold (K=10)
%% y el vector de atributos mejor clasificado por fscmrmr (vpSVML_KFold)
% ------------------------------------------

%% 1 - Generación del árbol grande
% ------------------------------------------
vpTree_KFold = [9, 17];
indPredicCat = [17];

ZTrainTree = XTrainSub(:,vpTree_KFold);
modelTree = fitctree(ZTrainTree, YTrainSub);
view(modelTree,'Mode','graph')


%% 2 - Subárbol mejor
% ------------------------------------------
% 2.1. Aplicamos Cost Complexity Pruning al árbol generado en el paso anterior
% para obtener una secuencia de los mejores subárboles candidatos,
% como una función del parámetro de complejidad alfa
valoresAlfa= modelTree.PruneAlpha;

% 2.2. Evaluamos cada subárbol generado mediante K-Fold cross validation para
% elegir el valor de alfa o subárbol asociado
% K = 10
numValoresAlfa = length(valoresAlfa);
erroresAlfa = zeros(numValoresAlfa,1);

for i=1:numValoresAlfa

    subtree = prune(modelTree,'alpha',valoresAlfa(i));%
    erroresAlfa(i)=cvloss(subtree,'kfold',10); % error promedio

end

% 2.3. Seleccionamos el subárbol (o de forma equivalente, el valor de ALPHA)
% que presenta el error medio minimo
[errorMinimo, indiceErrorMinimo] = min(erroresAlfa);
```

```
valorAlfaSeleccionado = valoresAlfa(indiceErrorMinimo);

treeSelected = prune(modelTree, 'alpha', valorAlfaSeleccionado);
view(treeSelected,'Mode','graph')

modelTree_KFold = treeSelected;

% GRAFICO – Anexo 4: Árbol podado utilizando el vector óptimo clasificado

%% 3 - Aplicación del modelo en el conjunto de validación XTest
% ---------------------------------------------
ZTestFilt = XTest(:,vpTree_KFold);
[YTest_Tree, pTree_KFold] = predict(modelTree_KFold,ZTestFilt);


%% 4 - Evaluación mediante curva ROC
% ---------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'Tree KFold','-*r','k*'};
scores = pTree_KFold(:,indiceClasPos);
[SENS_Tree_KFold, FP_Tree_KFold, UMBRALES_Tree_KFold, TOP_Tree_KFold,
AUC_Tree_KFold, leyenda_Tree_KFold, X_Tree_KFold,Y_Tree_KFold, T_Tree_KFold,
OPTROCPT_Tree_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4: Curva ROC Tree con el vector de predictores óptimo clasificado

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC
```

```
%% 5 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pTree_KFold(:,indiceClasPos);
YClasificador_Binaria=scores>=TOP_Tree_KFold;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_Tree_KFold = tablaRes(:,[1 3 5])

%                   Acc         Se          Sp
% Tree K-Fold     0.87618     0.77797     0.89415


%% MODELO 2
%% Mejor vector según Acc utilizando el método Wrapper (con validación
%% cruzada con 10 particiones) y el vector de predictores original,
%% no el clasificado.
% ------------------------------------------


%% 1 - Generación del árbol grande
% ------------------------------------------
indPredicCat = [17];

ZTrainTree = XTrainSub(:,vpTree_wrapper);
modelTree = fitctree(ZTrainTree, YTrainSub);
view(modelTree,'Mode','graph')

%% 2 - Subárbol mejor
% ------------------------------------------
% 2.1. Aplicamos Cost Complexity Pruning al árbol generado en el paso anterior
% para obtener una secuencia de los mejores subárboles candidatos,
% como una función del parámetro de complejidad alfa
valoresAlfa= modelTree.PruneAlpha;

% 2.2. Evaluamos cada subárbol generado mediante K-Fold cross validation para
% elegir el valor de alfa o subárbol asociado
% K = 10
numValoresAlfa = length(valoresAlfa);
erroresAlfa = zeros(numValoresAlfa,1);
```

```
for i=1:numValoresAlfa

    subtree = prune(modelTree,'alpha',valoresAlfa(i));%
    erroresAlfa(i)=cvloss(subtree,'kfold',10); % error promedio

end

% 2.3. Seleccionamos el subárbol (o de forma equivalente, el valor de ALPHA)
% que presenta el error medio minimo
[errorMinimo, indiceErrorMinimo] = min(erroresAlfa);
valorAlfaSeleccionado = valoresAlfa(indiceErrorMinimo);

treeSelected = prune(modelTree, 'alpha', valorAlfaSeleccionado);
view(treeSelected,'Mode','graph')

modelTree_wrapper = treeSelected;

% GRAFICO – Anexo 4: Árbol podado utilizando el vector óptimo obtenido con Wrapper

%% 3 - Aplicación del modelo en el conjunto de validación XTest
% ---------------------------------------------
ZTestFilt = XTest(:,vpTree_wrapper);
[YTest_Tree, pTree_wrapper] = predict(modelTree_wrapper,ZTestFilt);


%% 4 - Evaluación mediante curva ROC
% ---------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'Tree Wrapper','-*r','k*'};
scores = pTree_wrapper(:,indiceClasPos);
[SENS_Tree_wrapper, FP_Tree_wrapper, UMBRALES_Tree_wrapper, TOP_Tree_wrapper, ...
    AUC_Tree_wrapper, leyenda_Tree_wrapper, X_Tree_wrapper, Y_Tree_wrapper, ...
    T_Tree_wrapper, OPTROCPT_Tree_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4: Curva ROC Tree con el vector de predictores óptimo obtenido con
Wrapper

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
```

```
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 5 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pTree_wrapper(:,indiceClasPos);
YClasificador_Binaria=scores>=TOP_Tree_wrapper;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_Tree_wrapper = tablaRes(:,[1 3 5])

%                   Acc         Se          Sp
% Tree Wrapper    0.87835     0.76923     0.89831


%% 6 - Comparación curva ROC ambos modelos
% ------------------------------------------------
% Para poder comparar varias curvas ROC, infoGrafica debe tener leyenda
% Llamamos a la función con la opción de actualizar la leyenda - vamos
% a representar en la misma gráfica múltiples curvas de distintos
% clasificadores
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

leyenda = cell('');
infoGrafica={'Tree KFold','-r','r*', leyenda};
scores = pTree_KFold(:,indiceClasPos);
[SENS_Tree_KFold, FP_Tree_KFold, UMBRALES_Tree_KFold, TOP_Tree_KFold,
AUC_Tree_KFold, ...
    leyenda, X_Tree_KFold,Y_Tree_KFold, T_Tree_KFold, OPTROCPT_Tree_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);
```

```
infoGrafica={'Tree Wrapper','-b','b*', leyenda};
scores = pTree_wrapper(:,indiceClasPos);
[SENS_Tree_wrapper, FP_Tree_wrapper, UMBRALES_Tree_wrapper, TOP_Tree_wrapper, ...
    AUC_Tree_wrapper, leyenda, X_Tree_wrapper, Y_Tree_wrapper, ...
    T_Tree_wrapper, OPTROCPT_Tree_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);


%% 7 - Tabla comparativa métricas clasificación binaria de los umbrales
% ---------------------------------------------
% Creamos una tabla conjunta de las métricas Acc, Se y Sp obtenidas en los
% umbrales óptimo de cada modelo
tablaResCompletaTest_Tree = [tabla_umbral_Tree_KFold; ...
    tabla_umbral_Tree_wrapper];
tablaResCompletaTest_Tree.Properties.RowNames = {'Tree K-Fold', ...
    'Tree Wrapper'}

Ruta_Nombre = 'DatosGenerados\Metricas_umbral_Tree.mat';
save(Ruta_Nombre, 'tablaResCompletaTest_Tree')

% ---------------------------------------------
% RESULTADOS:
% ---------------------------------------------
%                    Acc          Se          Sp
% Tree K-Fold     0.87618     0.77797     0.89415
% Tree Wrapper    0.87835     0.76923     0.89831

% Si utilizamos Tree (árbol de decisión), obtendremos una mejor tasa de
% acierto de especificidad para el vector de atributos óptimo
% obtenido con la técnica Wrapper. Por el contrario, la mayor sensibilidad
% se obtiene para el vector de predictores óptimo obtenido tras la
% clasificación
```

Anexo 3.10. Bosque aleatorio

```
%% Selección del vector de predictores óptimo
tipoClasificador='RandomForest';

%% TÉCNICA 1
% Vectores de atributos:
% 1
vectorPredictores=[9];
posPredCat=[];
[Acc_RForest_1, Se_RForest_1, Sp_RForest_1] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 2
vectorPredictores=[9, 17];
posPredCat=[2];
[Acc_RForest_2, Se_RForest_2, Sp_RForest_2] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 3
```

```
vectorPredictores=[9, 17, 8];
posPredCat=[2];
[Acc_RForest_3, Se_RForest_3, Sp_RForest_3] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 4
vectorPredictores=[9, 17, 8, 11];
posPredCat=[2, 4];
[Acc_RForest_4, Se_RForest_4, Sp_RForest_4] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 5
vectorPredictores=[9, 17, 8, 11, 1];
posPredCat=[2, 4];
[Acc_RForest_5, Se_RForest_5, Sp_RForest_5] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 6
vectorPredictores=[9, 17, 8, 11, 1, 12];
posPredCat=[2, 4];
[Acc_RForest_6, Se_RForest_6, Sp_RForest_6] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 7
vectorPredictores=[9, 17, 8, 11, 1, 12, 6];
posPredCat=[2, 4];
[Acc_RForest_7, Se_RForest_7, Sp_RForest_7] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 8
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16];
posPredCat=[2, 4, 8];
[Acc_RForest_8, Se_RForest_8, Sp_RForest_8] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 9
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10];
posPredCat=[2, 4, 8];
[Acc_RForest_9, Se_RForest_9, Sp_RForest_9] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 10
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3];
posPredCat=[2, 4, 8];
[Acc_RForest_10, Se_RForest_10, Sp_RForest_10] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 11
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7];
posPredCat=[2, 4, 8];
[Acc_RForest_11, Se_RForest_11, Sp_RForest_11] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 12
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15];
posPredCat=[2, 4, 8];
```

```
[Acc_RForest_12, Se_RForest_12, Sp_RForest_12] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 13
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5];
posPredCat=[2, 4, 8];
[Acc_RForest_13, Se_RForest_13, Sp_RForest_13] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 14
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2];
posPredCat=[2, 4, 8];
[Acc_RForest_14, Se_RForest_14, Sp_RForest_14] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 15
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14];
posPredCat=[2, 4, 8];
[Acc_RForest_15, Se_RForest_15, Sp_RForest_15] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 16
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4];
posPredCat=[2, 4, 8];
[Acc_RForest_16, Se_RForest_16, Sp_RForest_16] =...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub,c,posPredCat,tipoClasificador)
% 17
vectorPredictores=[9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4, 13];
posPredCat = [2, 4, 8]
[Acc_RForest_17, Se_RForest_17, Sp_RForest_17] = ...
    funcion_calcula_kfold_validacion_cruzada(XTrainSub(:,(vectorPredictores)),...
                        YTrainSub, c,posPredCat,tipoClasificador)

% ------------------------------------------
%% RESUMEN RESULTADOS vpRForest:
% ------------------------------------------
% Tasa de acierto:
%   1          2        3          4        5          6         7
% 0.7960    0.8028    0.8192    0.8248    0.8222    0.8275    0.8424
%   8          9        10         11        12         13        14
% 0.8495    0.8469    0.8458    0.8436    0.8492    0.8439    0.8477
%   15         16        17
% 0.8462    0.8540    0.8450

% Sensibilidad:
%   1          2        3          4        5          6         7
% 0.7052    0.7229    0.7622    0.8194    0.8311    0.8361    0.8504
%   8          9        10         11        12         13        14
% 0.8593    0.8549    0.8538    0.8472    0.8539    0.8512    0.8540
%   15         16        17
% 0.8468    0.8605    0.8515

% Especificidad:
%   1          2        3          4        5          6         7
% 0.8879    0.8835    0.8766    0.8306    0.8140    0.8194    0.8347
```

```
%   8          9         10         11         12         13         14
% 0.8399    0.8392    0.8375    0.8398    0.8443    0.8368    0.8416
%   15         16         17
% 0.8460    0.8481    0.8392

% El porcentaje más alto de aciertos se obtiene para el vector formado por
% 16 predictores, y es del 85,4%
% vpRForest_KFold = [9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4]


%% TÉCNICA 2
% Wrapper o procedimiento de selección de forma escalonada hacia delante
% En este pasos e van a seleccionar vectores de atributos de cada dimensión
% posible (de 1 a 17) con el criterio de mayor rendimiento medido en
% términos de tasa de acierto según cada tipo de clasificador. La
% estrategia de entrenamiento-evaluación que se aplica es validación
% cruzada con 10 particiones, igual que en el caso anterior, pero con la
% diferencia de que ahora se utiliza el vector de predictores original:
% [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17] y no el
% vector de predictores clasificado.
% Utilizamos la misma partición de c en todos los casos
tipoClasificador='RandomForest';
indPredicCat=[11, 16, 17]; % Cuáles son las variables categóricas.

vectoresAtributosMaxAcc = cell(1,numAtributos);
valoresAccVectoresAtributosMaxAcc = zeros(1,numAtributos);
nombreVectoresAtributosMaxAcc = cell(1,numAtributos);
vectorAtributosMaxAcc = [];

for dim=1:numAtributos

    % La función genera el vector de dimensión dim de mayor Acc,
    % incorporando un predictor al vector obtenido en la iteración
    % de dimensión dim-1

    [vectorAtributosMaxAcc, Acc]...
        =funcion_genera_vectorAtr_haciaDelante_crossVal(...
            vectorAtributosMaxAcc, XTrainSub, YTrainSub, c, indPredicCat, ...
            tipoClasificador);

    vectoresAtributosMaxAcc{dim} = vectorAtributosMaxAcc';

    valoresAccVectoresAtributosMaxAcc(dim) = Acc;

    nombreVectoresAtributosMaxAcc{dim} = ...
        varNamesX(vectorAtributosMaxAcc)';
end

% vectoresAtributosMaxAcc --> guarda todos los vectores de todas las
% dimensiones
% valoresAccVectoresAtributosMaxAcc --> Acc de cada vector de cada
% dimensión
% Acc -->  Acc del último vector (dimensión=17)

% Representación gráfica de la tasa de errores
figure,
```

```
plot(1-valoresAccVectoresAtributosMaxAcc,'*-b')
xlabel('Dimension de vector de atributos')
ylabel('Tasa de Errores')
axis([0 numAtributos+1 0 0.5])
grid on
leyenda = cell('')
leyenda{1}=('Random Forest: 10-fold Cross-Validation búsqueda hacia adelante')
legend(leyenda)

% GRAFICO – Anexo 4: Tasa de error de los vectores de dimensión 1 a 17 utilizando la
técnica Wrapper y Random Forest como clasificador

%Guardamos los resultados de mayor Acc
[AccMaxRForest, indiceAccMaxRForest]= max(valoresAccVectoresAtributosMaxAcc)
vectorPredictoresRForest = vectoresAtributosMaxAcc{indiceAccMaxRForest}
nombresVectorPredictoresRForest = varNamesX(vectorPredictoresRForest)'

%.....................................................................
% GUARDAMOS RESULTADO EN VARIABLE CON LA NOTACIÓN QUE SE UTILIZARÁ
%.....................................................................

vpRForest_wrapper = vectorPredictoresRForest;

% vpRForest_wrapper = [9, 5, 11, 7, 4, 10, 15, 13, 3, 14, 17, 6, 1]


%% MODELO 1
%% Mejor vector según Acc utilizando K-Fold (K=10)
%% y el vector de atributos mejor clasificado por fscmrmr (vpRForest_KFold)
% ------------------------------------------

%% 1 -Entrenamiento del modelo en XTrainSub - YTrainSub
% ------------------------------------------
vpRForest_KFold = [9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2, 14, 4];
varNamesX(vpRForest_KFold); % Utiliza todos los atributos

indPredicCat = [17, 11, 16]; % Contempla poder tratar con variables categóricas, al
igual que el arbol
NumTrees = 50;
ZTrainRForest = XTrainSub(:,vpRForest_KFold);
posPredCatRForest = find(ismember(vpRForest_KFold,indPredicCat));
modelRForest = TreeBagger(NumTrees, ZTrainRForest,YTrainSub,...
                        'CategoricalPredictors',posPredCatRForest, ...
                        'Method','classification');
% view(modelRForest.Trees{1},'Mode','graph')


%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ------------------------------------------
ZTestFilt = XTest(:,vpRForest_KFold);
[YTest_RForest, pRForest_KFold] = predict(modelRForest,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
```

```
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'RandomForest KFold','-*r','k*'};
scores = pRForest_KFold(:,indiceClasPos);
[SENS_RForest_KFold, FP_RForest_KFold, UMBRALES_RForest_KFold, TOP_RForest_KFold,
AUC_RForest_KFold, leyenda_RForest_KFold, X_RForest_KFold, Y_RForest_KFold,
T_RForest_KFold, OPTROCPT_RForest_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4: Curva ROC Random Forest con el vector de predictores óptimo
clasificado

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pRForest_KFold(:,indiceClasPos);
YClasificador_Binaria=scores>=TOP_RForest_KFold;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
```

```
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_RForest_KFold = tablaRes(:,[1 3 5])

%                            Acc         Se          Sp
% Random Forest K-Fold     0.8605      0.8549      0.86153


%% MODELO 2
%% Mejor vector según Acc utilizando el método Wrapper (con validación
%% cruzada con 10 particiones) y el vector de predictores original,
%% no el clasificado.
% ---------------------------------------------

%% 1 - Entrenamiento del modelo en XTrainSub - YTrainSub
% ---------------------------------------------
varNamesX(vpRForest_wrapper); % Utiliza todos los atributos

indPredicCat = []; % Contempla poder tratar con variables categóricas
NumTrees = 50;
ZTrainRForest = XTrainSub(:,vpRForest_wrapper);
posPredCatRForest = find(ismember(vpRForest_wrapper,indPredicCat));
modelRForest = TreeBagger(NumTrees, ZTrainRForest,YTrainSub,...
                          'CategoricalPredictors',posPredCatRForest, ...
                          'Method','classification');
% view(modelRForest.Trees{1},'Mode','graph')


%% 2 - Aplicación del modelo en el conjunto de validación XTest
% ---------------------------------------------
ZTestFilt = XTest(:,vpRForest_wrapper);
[YTest_RForest, pRForest_wrapper] = predict(modelRForest,ZTestFilt);


%% 3 - Evaluación mediante curva ROC
% ---------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

infoGrafica={'RandomForest Wrapper','-*r','k*'};
scores = pRForest_wrapper(:,indiceClasPos);
[SENS_RForest_wrapper, FP_RForest_wrapper, UMBRALES_RForest_wrapper, ...
    TOP_RForest_wrapper, AUC_RForest_wrapper, leyenda_RForest_wrapper, ...
    X_RForest_wrapper, Y_RForest_wrapper, T_RForest_wrapper,
OPTROCPT_RForest_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

% GRAFICO – Anexo 4: Curva ROC Random Forest con el vector de predictores óptimo
obtenido con Wrapper

% Variables de entrada
    % scores --> vector de predicciones del clasificador: probabilidad
```

```
    % obtenida por el clasificador para la clase positiva

    % labels --> vector de clases reales: etiquetas de clase verdaderas,
    % especificadas como vector numérico, vector lógico, matriz de
    % caracteres, matriz de cadenas, matriz de celdas de vectores de
    % caracteres o matriz categórica.

    % posclass/codifClassPos --> etiqueta de la clase positica:
    % codificacion de la clase que se trata como positiva

    % infoGrafica: array de 3 o 3 celdas con información para la representación
    % Si esta variable no se pasa como argumento de entrada, la función no
    % representa la curva. Opciones:
        % - Representación de múltiples curvas:
            % infoGrafica admite una cuarta celda con información para
            % actualización de la leyenda
        % - Representación de una única curva:
            % infoGrafica, no debe incluir contenido en la cuarta celda

% Variables de salida
    % X - Y → coordenadas de la curva ROC
    % T → matriz de umbrales en las puntuaciones del clasificador
    % AUC → área bajo la curva
    % OPTROCPT → punto operativo óptimo de la curva ROC


%% 4 - Métricas Acc, Se y Sp en el punto de operación óptimo
% ------------------------------------------------
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pRForest_wrapper(:,indiceClasPos);
YClasificador_Binaria=scores>=TOP_RForest_wrapper;
% YClasificador_Binaria: predicción de la clase de cada instancia de test
% de acuerdo con el umbral. Codificado como 1 binario la clase positiva y 0
% binario la negativa
% Como nuestras clases reales (YTest) con las que tenemos que comparar para
% obtener las métricas tienen una codificación de 1 (para la clase
% negativa) y 2 (la positiva), hay que transformar YTest:
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false)
tabla_umbral_RForest_wrapper = tablaRes(:,[1 3 5])

%                            Acc         Se          Sp
% Random Forest Wrapper    0.84753     0.85664     0.84586


%% 5 - Comparación curva ROC ambos modelos
% ------------------------------------------------
% Para poder comparar varias curvas ROC, infoGrafica debe tener leyenda
% Llamamos a la función con la opción de actualizar la leyenda - vamos
```

```
% a representar en la misma gráfica múltiples curvas de distintos
% clasificadores
codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);

leyenda = cell('');
infoGrafica={'RandomForest KFold','-r','k*', leyenda};
scores = pRForest_KFold(:,indiceClasPos);
[SENS_RForest_KFold, FP_RForest_KFold, UMBRALES_RForest_KFold, TOP_RForest_KFold,
...
    AUC_RForest_KFold, leyenda, X_RForest_KFold, Y_RForest_KFold, ...
    T_RForest_KFold, OPTROCPT_RForest_KFold] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);

infoGrafica={'RandomForest Wrapper','-b','k*', leyenda};
scores = pRForest_wrapper(:,indiceClasPos);
[SENS_RForest_wrapper, FP_RForest_wrapper, UMBRALES_RForest_wrapper, ...
    TOP_RForest_wrapper, AUC_RForest_wrapper, leyenda, ...
    X_RForest_wrapper, Y_RForest_wrapper, T_RForest_wrapper,
OPTROCPT_RForest_wrapper] = ...
        funcion_genera_curva_ROC (YTest,scores,codifClasePos,infoGrafica);


%% 6 - Tabla comparativa métricas clasificación binaria de los umbrales
% ---------------------------------------------
% Creamos una tabla conjunta de las métricas Acc, Se y Sp obtenidas en los
% umbrales óptimo de cada modelo
tablaResCompletaTest_RForest = [tabla_umbral_RForest_KFold; ...
    tabla_umbral_RForest_wrapper];
tablaResCompletaTest_RForest.Properties.RowNames = {'Random Forest K-Fold', ...
    'Random Forest Wrapper'}

Ruta_Nombre = 'DatosGenerados\Metricas_umbral_RForest.mat';
save(Ruta_Nombre, 'tablaResCompletaTest_RForest')

% ---------------------------------------------
% RESULTADOS:
% ---------------------------------------------
%                            Acc         Se          Sp
% Random Forest K-Fold     0.8605      0.8549      0.86153
% Random Forest Wrapper    0.84753     0.85664     0.84586

% Si utilizamos Random Forest, obtendremos una mejor tasa de acierto y de
% especificidad para el vector de atributos óptimo obtenido tras la
% clasificación. La mayor tasa de sensbilidad, sin embargo, se obtiene para
% el vector de predictores óptimo obtenido con la técnica Wrapper.


%% 7 - Métricas Acc, Se y Sp en puntos objetivos de la curva ROC del
%% mejor modelo (mayor AUC)
% ---------------------------------------------
% El modelo con mayor área bajo la curva es Random Forest utilizando el
% vector de predictores [9, 17, 8, 11, 1, 12, 6, 16, 10, 3, 7, 15, 5, 2,
% 14, 4] de dimensión 16.
%% a) NIVELES DE ESPECIFICIDAD OBJETIVOS
```

```
% Se elegirán distintos niveles de especificidad (65-70-75-80-85-90-95)
% Para ello debemos seleccionar distintos puntos de la curva, nos fijamos
% en el eje de las x que representa 1 - especificidad, por lo que
% seleccionaremos los siguientes valores:
% 1-0,65 = 0,35 aproximadamente
% 1-0,7 = 0,3 aproximadamente
% 1-0,75 = 0,25 aproximadamente
% 1-0,8 = 0,2 aproximadamente
% 1-0,85 = 0,15 aproximadamente
% 1-0,9 = 0,1 aproximadamente
% 1-0,95 = 0,05 aproximadamente

X_Y_Umbral_RForest = [X_RForest_KFold, Y_RForest_KFold, UMBRALES_RForest_KFold];

% Los valores de X más cercanos a estos valores son:
%     X       Y     UMBRAL  FILA
%   0,3508  0,9493  0,22    1000
%   0,3     0,9371  0,2632  885
%   0,2498  0,9213  0,32    760
%   0,1999  0,8899  0,3963  620
%   0,15    0,8601  0,4924  491
%   0,1001  0,7867  0,6333  353
%   0,0499  0,5804  0,7983  174

% Establecemos los niveles objetivos de especificidad
umbral_objetivo_Sp_65 = UMBRALES_RForest_KFold(1000,1);
umbral_objetivo_Sp_70 = UMBRALES_RForest_KFold(885,1);
umbral_objetivo_Sp_75 = UMBRALES_RForest_KFold(760,1);
umbral_objetivo_Sp_80 = UMBRALES_RForest_KFold(620,1);
umbral_objetivo_Sp_85 = UMBRALES_RForest_KFold(491,1);
umbral_objetivo_Sp_90 = UMBRALES_RForest_KFold(353,1);
umbral_objetivo_Sp_95 = UMBRALES_RForest_KFold(174,1);

codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pRForest_KFold(:,indiceClasPos);

%% Sp = 65
YClasificador_Binaria=scores>=umbral_objetivo_Sp_65;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Sp_65 = tablaRes(:,[1 3 5])

%% Sp = 70
YClasificador_Binaria=scores>=umbral_objetivo_Sp_70;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
```

```
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Sp_70 = tablaRes(:,[1 3 5])

%% SP = 75
YClasificador_Binaria=scores>=umbral_objetivo_Sp_75;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Sp_75 = tablaRes(:,[1 3 5])

%% Sp = 80
YClasificador_Binaria=scores>=umbral_objetivo_Sp_80;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Sp_80 = tablaRes(:,[1 3 5])

%% Sp = 85
YClasificador_Binaria=scores>=umbral_objetivo_Sp_85;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Sp_85 = tablaRes(:,[1 3 5])

%% Sp = 90
YClasificador_Binaria=scores>=umbral_objetivo_Sp_90;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Sp_90 = tablaRes(:,[1 3 5])

%% Sp = 95
YClasificador_Binaria=scores>=umbral_objetivo_Sp_95;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
```

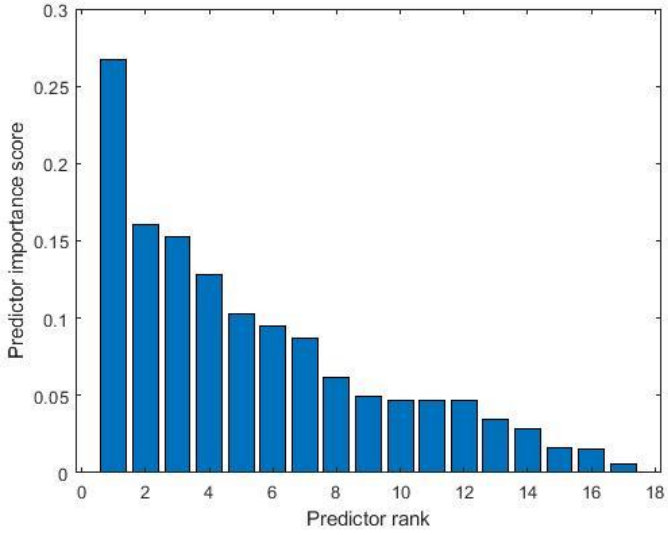
```
    codifClasePos, false);
tabla_umbral_RForest_KFold_Sp_95 = tablaRes(:,[1 3 5])

% Tabla completa
tablaResCompletaTest_RForest_Sp = [tabla_umbral_RForest_KFold_Sp_65; ...
    tabla_umbral_RForest_KFold_Sp_70; tabla_umbral_RForest_KFold_Sp_75; ...
    tabla_umbral_RForest_KFold_Sp_80; tabla_umbral_RForest_KFold_Sp_85; ...
    tabla_umbral_RForest_KFold_Sp_90; tabla_umbral_RForest_KFold_Sp_95];
tablaResCompletaTest_RForest_Sp.Properties.RowNames = {'Especificidad - 65', ...
    'Especificidad - 70', 'Especificidad - 75', 'Especificidad - 80', ...
    'Especificidad - 85', 'Especificidad - 90', 'Especificidad - 95'}

%% b) NIVELES DE SENSIBILIDAD OBJETIVOS
% Se elegirán distintos niveles de sensibilidad (65-70-75-80-85-90-95)
% Para ello debemos seleccionar distintos puntos de la curva, nos fijamos
% en el eje de las y que representa la sensibilidad.

X_Y_Umbral_RForest = [X_RForest_KFold, Y_RForest_KFold, UMBRALES_RForest_KFold];

% Los valores de Y más cercanos a estos valores son:
%     X       Y     UMBRAL  FILA
%   0.0659  0.6503  0.7467  230
%   0.0812  0.6993  0.7     279
%   0.0899  0.75    0.6665  320
%   0.1081  0.8007  0.6033  376
%   0.1362  0.8514  0.52    457
%   0.2139  0.9003  0.376   661
%   0.354   0.9493  0.2167  1010

% Establecemos los niveles objetivos de sensibilidad
umbral_objetivo_Se_65 = UMBRALES_RForest_KFold(230,1);
umbral_objetivo_Se_70 = UMBRALES_RForest_KFold(279,1);
umbral_objetivo_Se_75 = UMBRALES_RForest_KFold(320,1);
umbral_objetivo_Se_80 = UMBRALES_RForest_KFold(376,1);
umbral_objetivo_Se_85 = UMBRALES_RForest_KFold(457,1);
umbral_objetivo_Se_90 = UMBRALES_RForest_KFold(661,1);
umbral_objetivo_Se_95 = UMBRALES_RForest_KFold(1010,1);

codifClasePos = 2; % Elegimos la clase positiva la codificada con 2
codifClases = unique(YTest);
indiceClasPos = find(codifClases == codifClasePos);
scores = pRForest_KFold(:,indiceClasPos);

%% Se = 65
YClasificador_Binaria=scores>=umbral_objetivo_Se_65;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Se_65 = tablaRes(:,[1 3 5])

%% Se = 70
YClasificador_Binaria=scores>=umbral_objetivo_Se_70;
```

```
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Se_70 = tablaRes(:,[1 3 5])

%% Se = 75
YClasificador_Binaria=scores>=umbral_objetivo_Se_75;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Se_75 = tablaRes(:,[1 3 5])

%% Se = 80
YClasificador_Binaria=scores>=umbral_objetivo_Se_80;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Se_80 = tablaRes(:,[1 3 5])

%% 85
YClasificador_Binaria=scores>=umbral_objetivo_Se_85;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Se_85 = tablaRes(:,[1 3 5])

%% Se = 90
YClasificador_Binaria=scores>=umbral_objetivo_Se_90;
YTestClasificador=double(YClasificador_Binaria)+1;
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Se_90 = tablaRes(:,[1 3 5])

%% Se = 95
YClasificador_Binaria=scores>=umbral_objetivo_Se_95;
YTestClasificador=double(YClasificador_Binaria)+1;
```

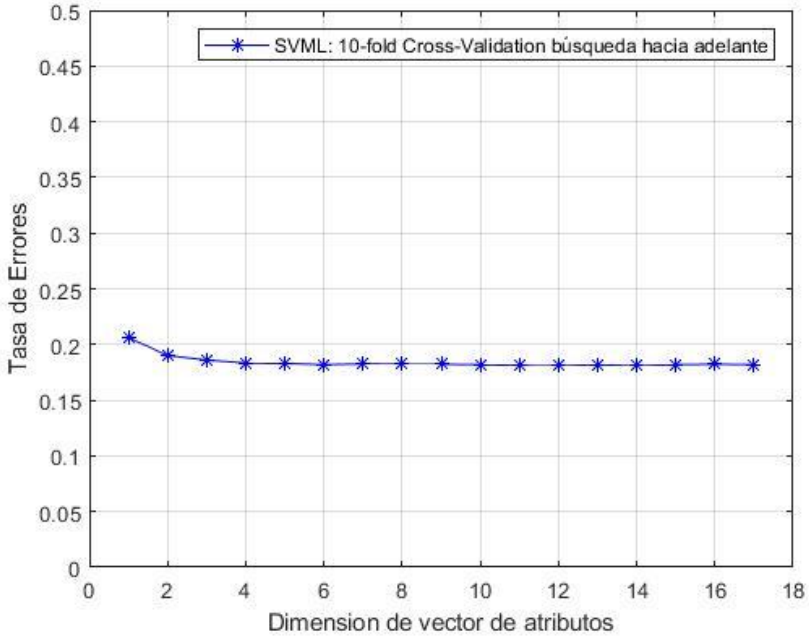
```
% Ahora lo que estaba codificado como 0 tiene un 1, y lo que estaba
% codificado con 1 tiene 2, igual que la clase real YTest.

[C, Acc, ER, Se,FNR, Sp, FPR, P, F, varNames, tablaRes] = ...
    funcion_calcula_metricas_clasificacion_binaria(YTest, YTestClasificador,...
    codifClasePos, false);
tabla_umbral_RForest_KFold_Se_95 = tablaRes(:,[1 3 5])

% Tabla completa
tablaResCompletaTest_RForest_Se = [tabla_umbral_RForest_KFold_Se_65; ...
    tabla_umbral_RForest_KFold_Se_70; tabla_umbral_RForest_KFold_Se_75; ...
    tabla_umbral_RForest_KFold_Se_80; tabla_umbral_RForest_KFold_Se_85; ...
    tabla_umbral_RForest_KFold_Se_90; tabla_umbral_RForest_KFold_Se_95];
tablaResCompletaTest_RForest_Se.Properties.RowNames = {'Sensibilidad - 65', ...
    'Sensibilidad - 70', 'Sensibilidad - 75', 'Sensibilidad - 80', ...
    'Sensibilidad - 85', 'Sensibilidad - 90', 'Sensibilidad - 95'}
```

Anexo 4

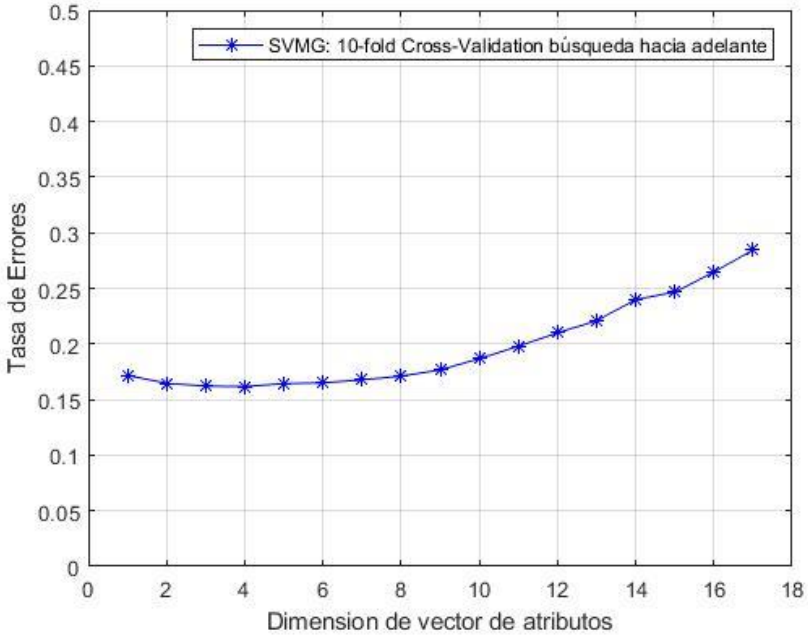
Puntuaciones de los predictores clasificados



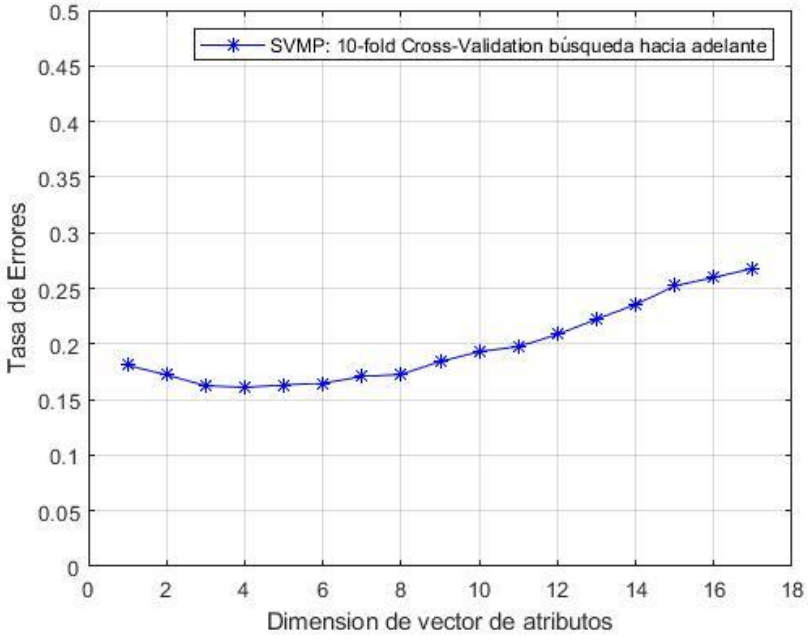
Tasa de error de los vectores de dimensión 1 a 17 utilizando la técnica *Wrapper* y *SVM*L como clasificador



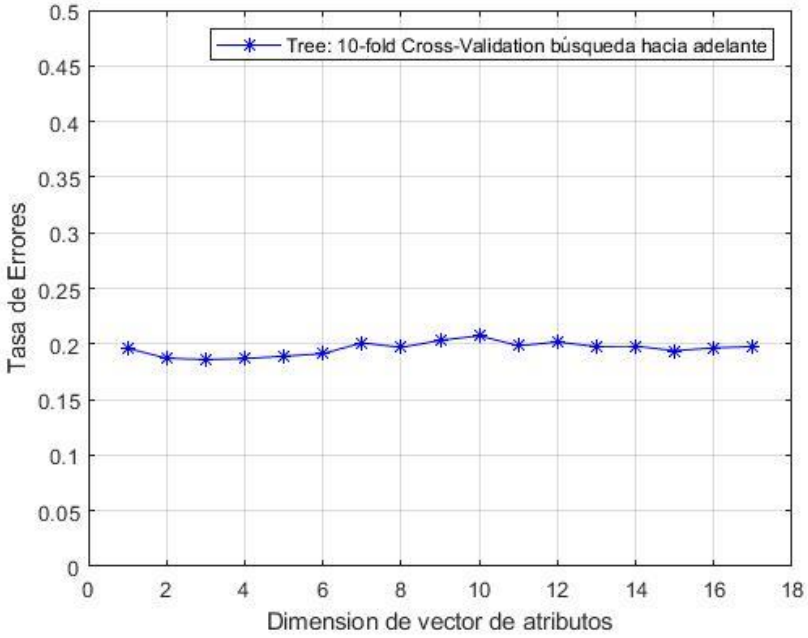
Tasa de error de los vectores de dimensión 1 a 17 utilizando la técnica *Wrapper* y *SVM*G como clasificador



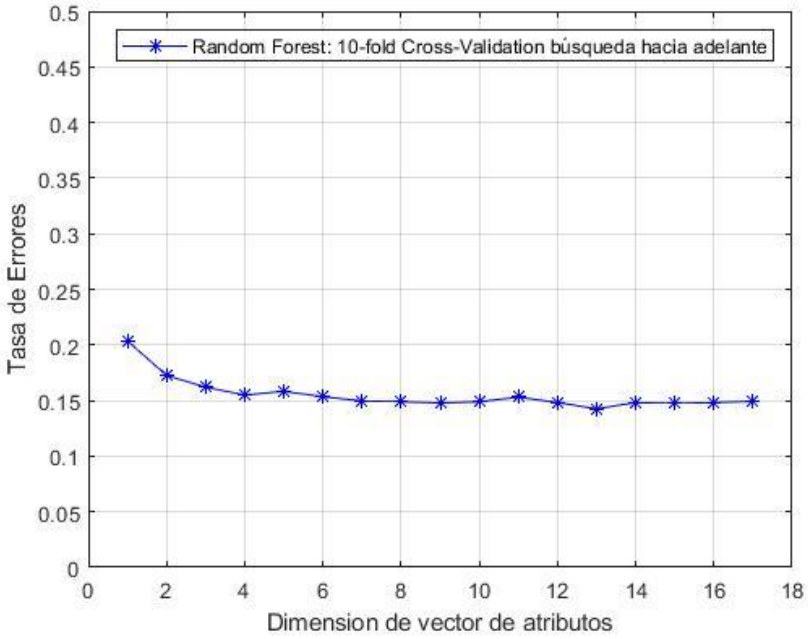
Tasa de error de los vectores de dimensión 1 a 17 utilizando la técnica *Wrapper* y *SVM*P como clasificador



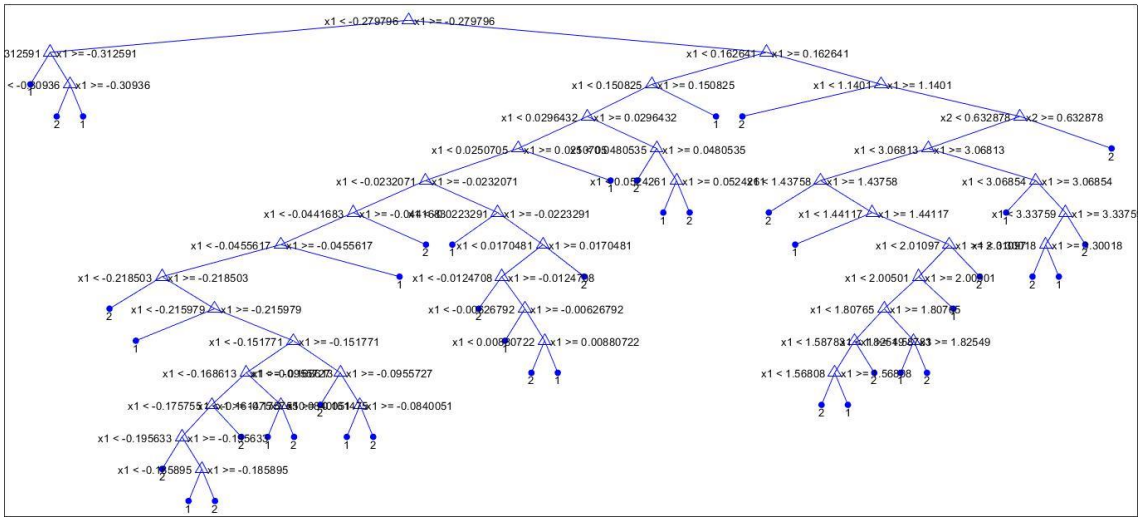
Tasa de error de los vectores de dimensión 1 a 17 utilizando la técnica *Wrapper* y *Tree* como clasificador



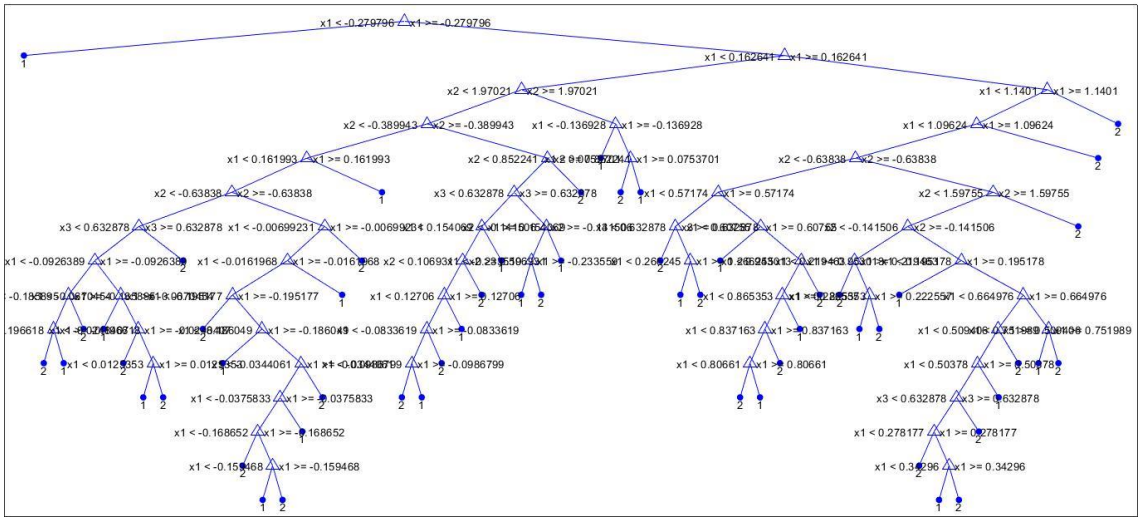
Tasa de error de los vectores de dimensión 1 a 17 utilizando la técnica *Wrapper* y *Random Forest* como clasificador



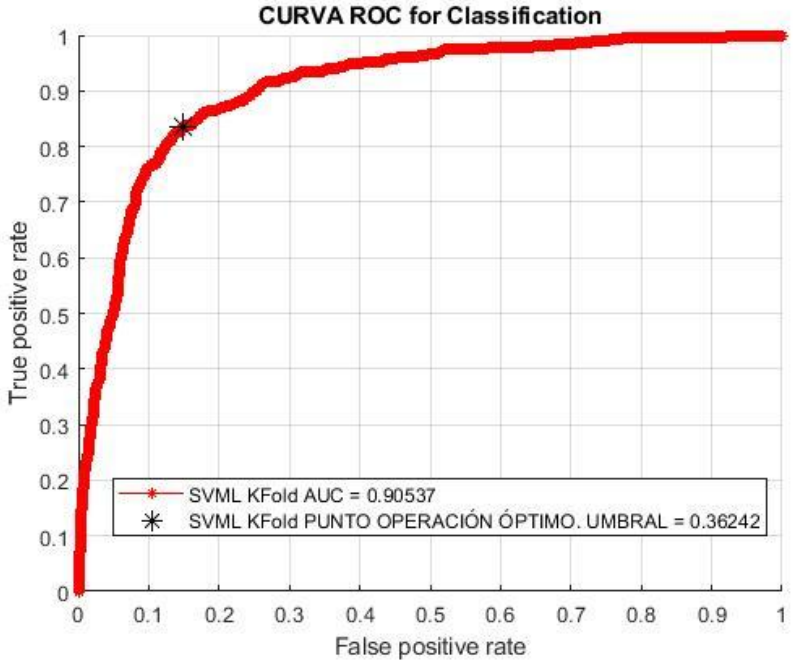
Árbol podado utilizando el vector óptimo obtenido clasificado



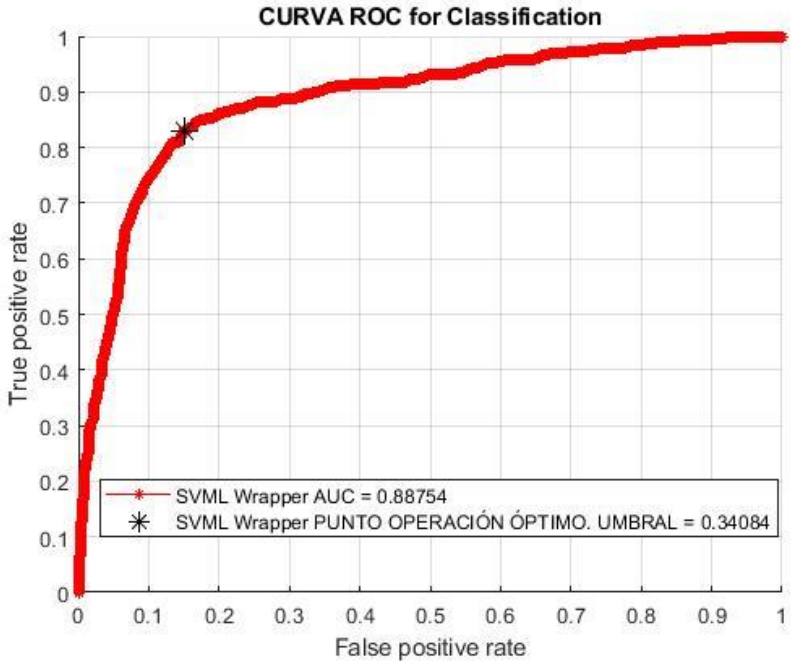
Árbol podado utilizando el vector óptimo obtenido con *Wrapper*



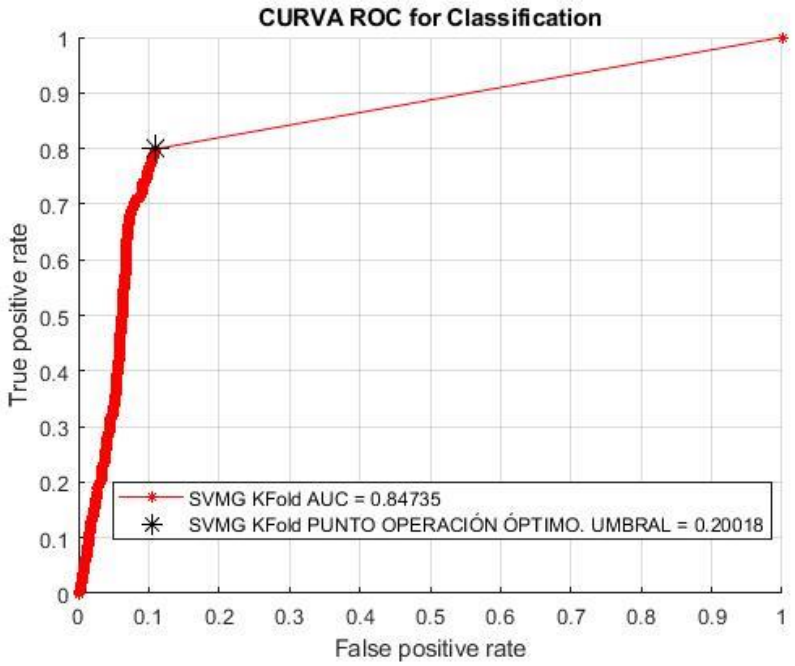
Curva ROC *SVML* con el vector de predictores óptimo clasificado



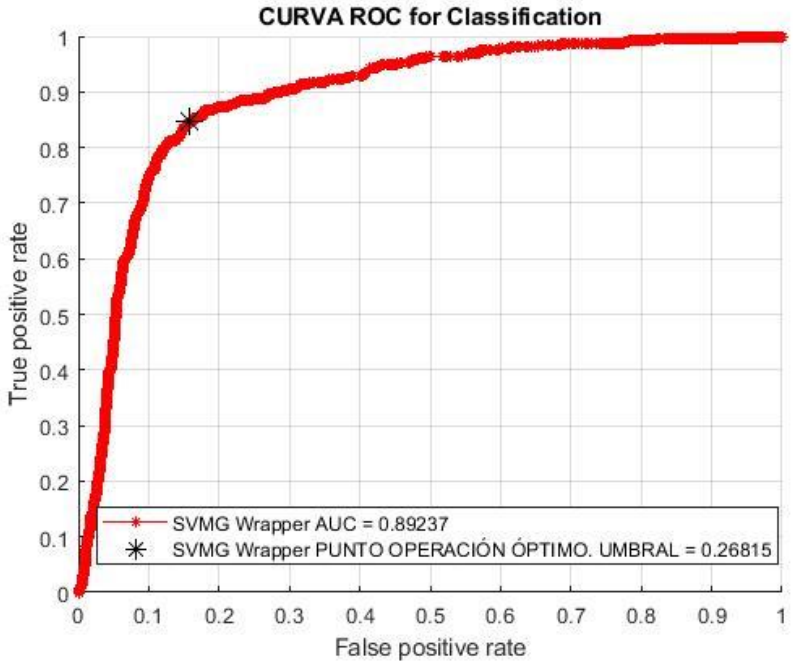
Curva ROC *SVML* con el vector de predictores óptimo obtenido con *Wrapper*



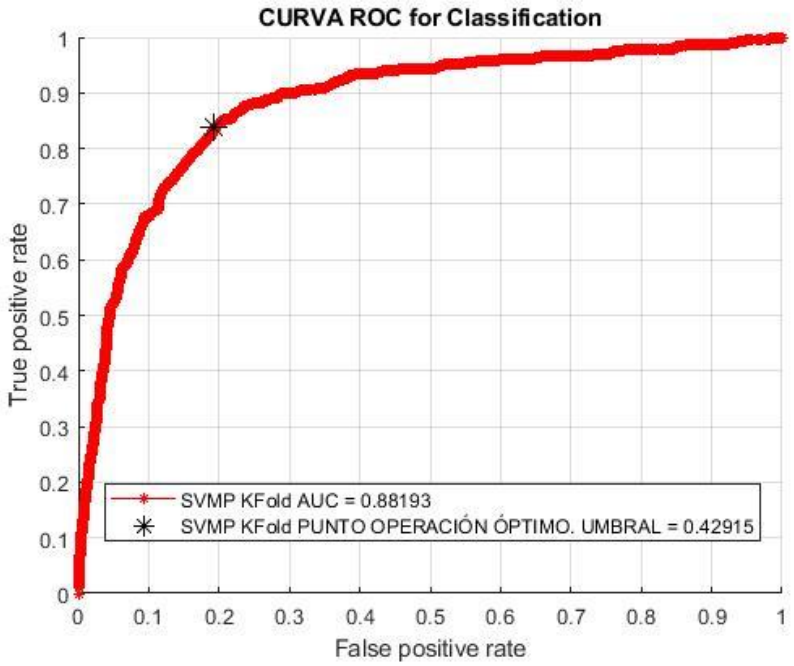
Curva ROC *SVMG* con el vector de predictores óptimo clasificado



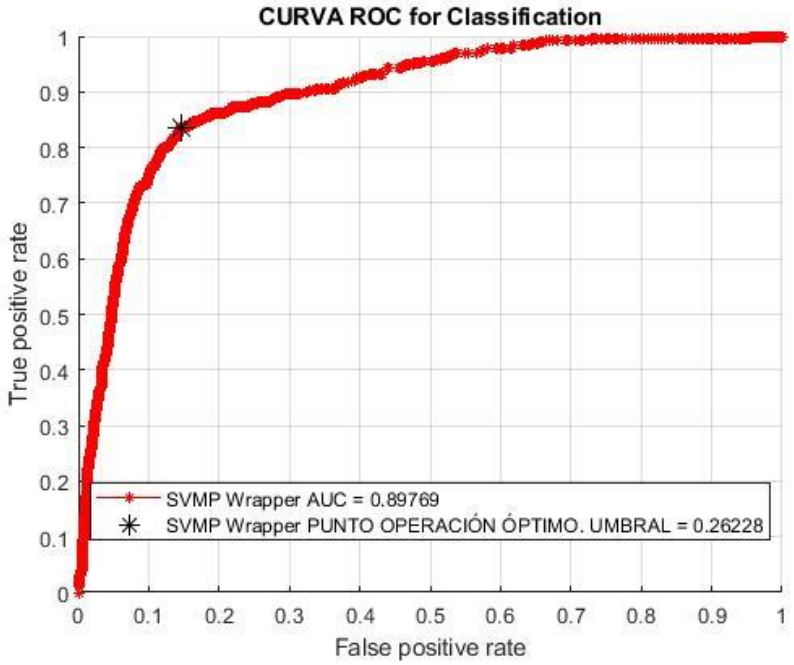
Curva ROC *SVMG* con el vector de predictores óptimo obtenido con *Wrapper*



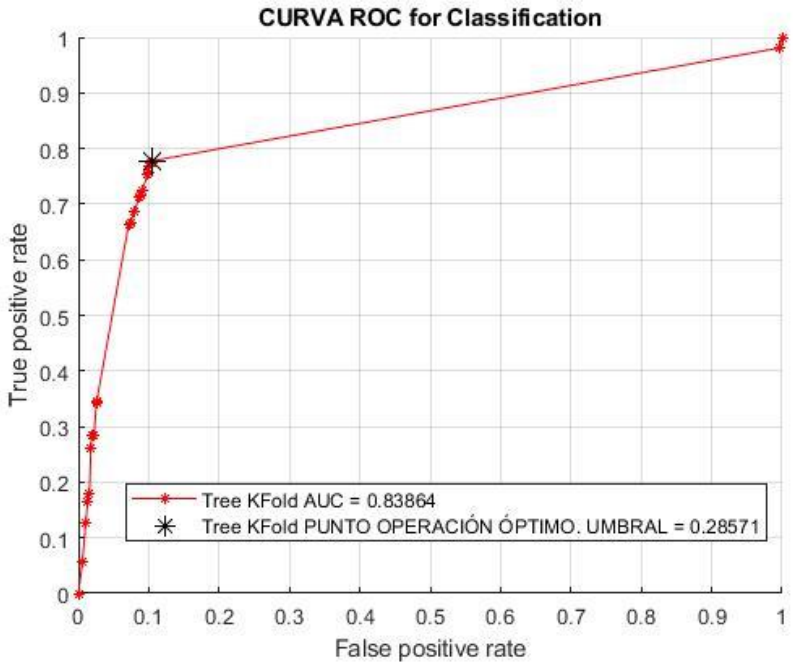
Curva ROC *SVMP* con el vector de predictores óptimo clasificado



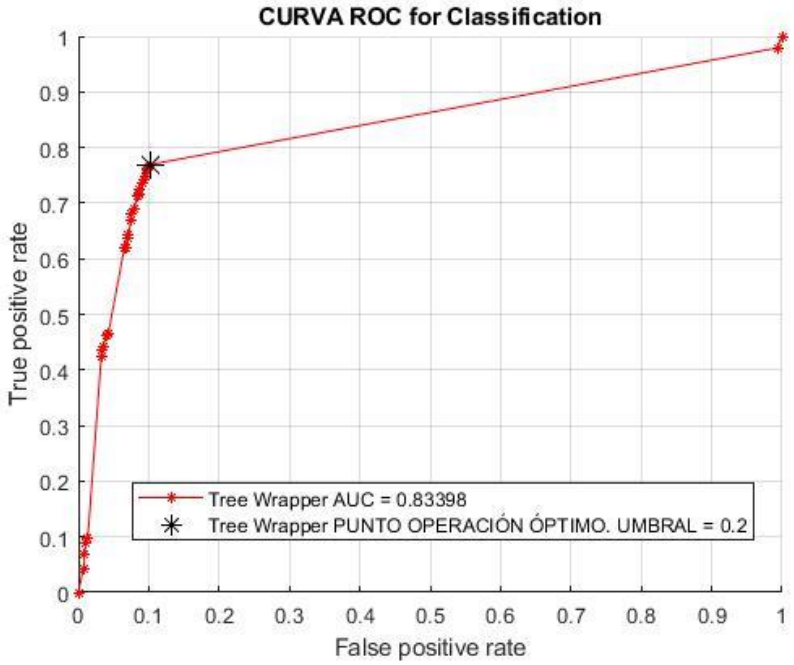
Curva ROC *SVMP* con el vector de predictores óptimo obtenido con *Wrapper*



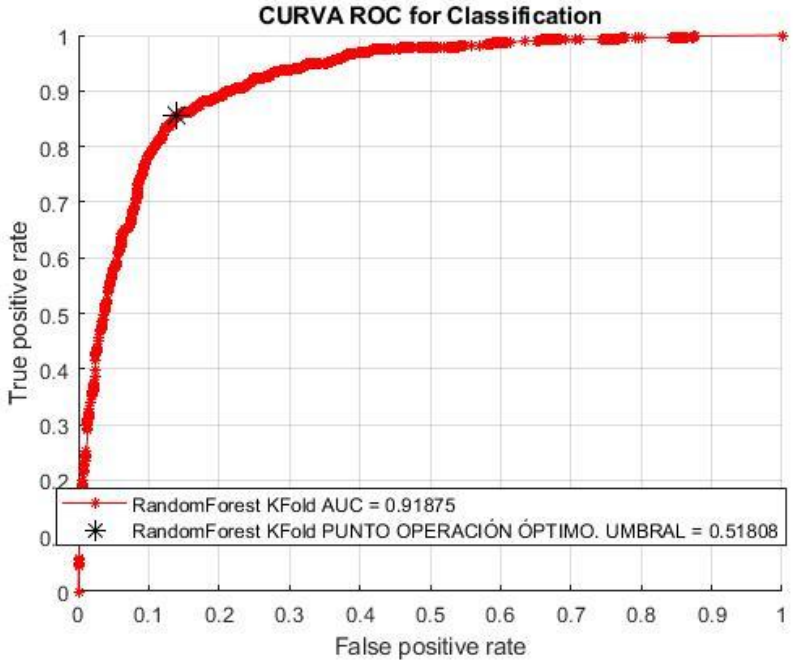
Curva ROC *Tree* con el vector de predictores óptimo clasificado



Curva ROC *Tree* con el vector de predictores óptimo obtenido con *Wrapper*



Curva ROC *Random Forest* con el vector de predictores óptimo clasificado



Curva ROC *Random Forest* con el vector de predictores óptimo obtenido con *Wrapper*