Selección de características en modelos predictivos SFS vs SFFS

Antonio Javier Moreno González
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
antmorgon4@alum.us.es

Félix Jiménez González
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
feljimgon1@alum.us.es

Resumen*— La *selección de características en modelos predictivos busca seleccionar aquellas variables que aporten la mayor cantidad posible de información, descartando aquellas que proporcionen menos. La aplicación funciona correctamente y aporta resultados positivos en cuanto a la devolución de los mismos (mejor rendimiento y menor número de variables), aunque bien es cierto que puede existir una mejora con respecto a la implementación para poder realizar los cálculos en un tiempo más razonable (aplicación de recursividad).

Palabras Clave—Inteligencia Artificial, selección de características, recursos tecnológicos, tiempo de entrenamiento, sobreajuste, modelos predictivos, método de envoltura, rendimiento, diagrama de violín, validación cruzada, árboles de decisión, SFFS, SFS.

I. INTRODUCCIÓN

La selección de características es un proceso que es recomendable aplicar, ya que descarta ciertas características (o variables) que pueden aportar poca información y mantiene aquellas que son más relevantes.

Este proceso se lleva a cabo por tres razones principales[i]:

- Recursos tecnológicos y tiempo: Se puede ver fácilmente que, si tenemos un número más elevado de variables en nuestro conjunto de datos, tendremos un mayor consumo de recursos (CPU, RAM, etc.) y el tiempo de entrenamiento será mayor.
- Simplificación: Con menos variables, los modelos predictivos obtenidos se simplifican, siendo más sencillos de interpretar.
- Sobreajuste: Con un conjunto más amplio de variables tendremos más datos, pero no quiere decir que aporten más información. Es decir, es posible que el modelo se ajuste demasiado al conjunto de entrenamiento proporcionado, aprendiendo patrones que en este se dan, pero en la realidad no, obteniendo patrones "falsos" (ruido).

  Sin embargo, con un subconjunto de variables que aporten más información, es muy posible obtener patrones que se ajusten más a casos que se encuentren fuera de nuestro conjunto de entrenamiento.

Existen distintas técnicas de selección de características (agrupadas en tres grupos [ii] principales; a saber, métodos de filtro, métodos de envoltura y métodos integrados), pero el trabajo se centrará en el Método de Envoltura (Wrapper Method).

Este método consiste en escoger ciertos subconjuntos de variables (características). Se entrena un modelo predictivo con cada uno de estos subconjuntos y finalmente se evalúa el rendimiento de cada uno para quedarnos con el mejor.

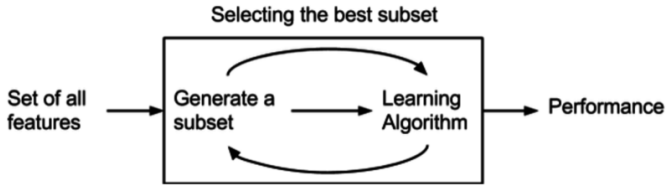

Figura 1. Método de envoltura. Fuente:https://ligdigonzalez.com/metodos-de-seleccion-de-caracteristicas-machine-learning/

Para el trabajo, el objetivo marcado no es devolver el subconjunto proporcionado por el método de envoltura que tenga mejor rendimiento, sino el que tenga una mejor correlación entre el rendimiento (se busca el mayor posible) y el número de variables/características (se busca el menor posible).

A continuación, se procede a definir la estructura del documento: una vez finalizada esta breve introducción, se dará lugar a unos preliminares, en los que se mencionan las técnicas empleadas y los trabajos relacionados. Más adelante se entra en detalle y se ve la metodología de trabajo, viendo los métodos implementados por los autores, explicando con pseudocódigo la funcionalidad de cada uno. A continuación, se realizará una recopilación analizando los resultados obtenidos (además de explicar el trabajo extra realizado) por los distintos algoritmos, y se finalizará con un apartado de conclusiones. Existe una parte adicional denominada ANEXO MARCO TEÓRICO, en el que se expondrán más en detalle las distintas familias de selección de características.

II. PRELIMINARES

En esta sección se explicará de manera introductoria las técnicas empleadas para implementar los algoritmos y solucionar el problema. También se mencionarán trabajos relacionados que utilicen estas mismas técnicas.

A. *Métodos empleados*

Para realizar la selección de características, se ha implementado (como se mencionó en la Introducción) el Método de Envoltura. Para ello:

- Tenemos en cuenta dos algoritmos de búsqueda secuenciales para selección de características:
    - SFS[iii]: o algoritmo hacia adelante, en el que comenzamos con un conjunto vacío y vamos introduciendo variables hasta llegar al número deseado.

1. Start with the empty set $Y_0 = \{\emptyset\}$
2. Select the next best feature $x^+ = \arg\max_{x \notin Y_k} J(Y_k + x)$
3. Update $Y_{k+1} = Y_k + x^+;\ k = k + 1$
4. Go to 2

Figura 2. Algoritmo Sequential Forward Selection. Fuente: https://www.researchgate.net/figure/Sequential-forward-selection-algorithm-wrapper-In-the-early-assumption-the-chances-to_fig1_319435092

    - SFFS[iv]: o algoritmo mixto, que a diferencia del algoritmo SFS, puede ir hacia adelante y hacia atrás. Se establece una dirección principal, pero en caso de que se crea conveniente puede ir hacia el otro lado para corregir una posible solución "no óptima". En nuestro caso, la dirección principal será la misma que utilizada en SFS, corrigiendo si es necesario volviendo hacia atrás. Nótese además, que si se usa SFFS, aunque tiene un coste computacional mayor, devuelve unos mejores resultados porque optimiza resultados.

1. $Y = \{\emptyset\}$
2. Select the best feature
   $x^+ = \arg\max_{x \notin Y_k} J(Y_k + x)$
   $Y_k = Y_k + x^+; k = k + 1$
3. Select the worst feature*
   $x^- = \arg\max_{x \in Y_k} J(Y_k - x)$
4. If $J(Y_k - x^-) > J(Y_k)$ then
   $Y_{k+1} = Y_k - x^-;\ k = k + 1$
   Go to step 3
   Else
   Go to step 2

Figura 3. Sequential Floating Forward Selection Algorithm. Fuente: https://www.cc.gatech.edu/~bboots3/CS4641-Fall2018/Lecture16/16_FeatureSelection.pdf

- Una vez seleccionamos el algoritmo de búsqueda (encargado de seleccionar los subconjuntos de variables), utilizamos la librería *sklearn* con el fin de entrenar los subconjuntos gracias a los árboles de decisión para finalmente, utilizando validación cruzada y el árbol devuelto, medir el rendimiento de este.

B. *Trabajo Relacionado*

Para entrar en contacto con el trabajo a realizar, se ha hecho uso de las siguientes fuentes para poder resolver los distintos problemas que han ido surgiendo:

- Práctica 1 vista en clase (Aprendizaje automático)[v]
- Documentación de las siguientes librerías:
    - *sklearn*[vi]
    - *pandas*[vii]
    - *statistics*[viii]
    - *matplotlib*[ix]
- El propio documento en el que se define y explica la implementación a llevar a cabo.

III. METODOLOGÍA

A continuación, pasamos a describir los métodos y los ficheros.py implementados para la realización del proyecto.

Para mostrarlo de una forma más organizada:

- Comenzaremos hablando de los ficheros, diciendo cuál es el contenido de cada uno de ellos.
- Una vez se ha explicado el contenido resumido de cada fichero, nos centraremos en su funcionalidad, para así mostrar los métodos implementados y entender el contexto de su uso.

Los ficheros que se han completado para la realización del proyecto son las siguientes (**en la entrega del proyecto se han volcado ambos ficheros en un único notebook que para más información, leer el fichero README.txt**):

1) *Script_Final.py (correspondiente a la segunda celda de código del notebook).*

El fichero Script_Final, no es más que el fichero de ejecución del proyecto.

En este se especifican tanto los parámetros de entrada que reciben los métodos (ficheros Excel, número de folds que se van a utilizar en la validación cruzada y otros que se mencionarán más adelante) como qué algoritmo de búsqueda se va a utilizar en el momento de la ejecución (SFS o SFFS).

2) *Proyecto.py (correspondiente a la primera celda del código de notebook).*

Es el grueso del proyecto. En este fichero, recibimos los parámetros de entrada proporcionados por el Script_Final.py y

son procesados según qué algoritmo hayamos seleccionado para que se ejecute.

A continuación, se procede a describir cómo sería la funcionalidad del proyecto tras la ejecución del Script_Final.py:

En este fichero podemos llamar a dos métodos incluidos en el fichero Proyecto.py. Estos son:

Seleccionar_características_sfs

Entrada:

- datos_csv (Datos en formato excel).
- names (Atributos a seleccionar, en principio vacía).
- umbral (El umbral a utilizar por SFS y SFFS. Es 10 por defecto).
- n_exp (El número de repeticiones del experimento por validación crucada, nuevamente 10 por defecto).
- cv (Número de folds a considerar en la validación cruzada, de nuevo 10 por defeto).
- métrica (Representa la métrica de evaluación a utilizar. Por defecto *"balanced_accuracy"*)

Salidas:

- Variables necesarias para poder aplicar la solución cruzada y el algoritmo SFS.

Algoritmo:

1. Instanciamos una variable *atributos* en la que almacenaremos las características.
2. Si la lista *atributos* está vacía, la rellenamos con las columnas del excel (que contendrán las características)
3. Si la métrica está a *none*, aplicamos la métrica por defecto (*balanced_accuracy*)
4. Devolvemos el método *aplicar_SFS*.

Este método es el encargado de inicializar las características que posteriormente serán utilizadas para aplicar el algoritmo SFS. De momento lo único que se ha hecho es instanciar las variables necesarias para poder aplicar SFS (que de hecho, es el método que se devuelve).

A continuación, presentamos el pseudocódigo de *aplicar_SFS*. Este, es el encargado de aplicar el algoritmo de búsqueda secuencial hacia adelante (SFS). El objetivo de este es seleccionar entre las variables escogidas (partiendo de un conjunto vacío de variables), la mejor característica a añadir al subconjunto, evaluando los posibles candidatos. Para esta evaluación, se crea un árbol de decisión con las variables escogidas por el algoritmo y obteniendo su rendimiento mediante validación cruzada.

aplicar _SFS

Entrada:

- datos (Datos del Excel pasados como parámetro).
- variables (Atributos proporcionados por el método Seleccionar_características_sfs).
- D (El umbral a utilizar. Es 10 por defecto).
- n_exp (El número de repeticiones del experimento por validación crucada, nuevamente 10 por defecto).
- cv (Número de folds a considerar en la validación cruzada, de nuevo 10 por defeto).
- métrica (Representa la métrica de evaluación a utilizar. Por defecto *"balanced_accuracy"*)

Salidas:

- dataframe compuesto por los posibles subconjuntos de características junto a su tamaño y rendimiento, ordenado por el rendimiento en orden descendente.

Algoritmo:

1. Instanciamos una variable atributos en la que incluimos las variables proporcionadas por el método anterior.
2. Instanciamos una variable solucion_actual en la que iremos almacenando la solución por cada iteración.
3. Instanciamos la variable column_names en la que, utilizando pandas, se visualizarán los datos de tamaño rendimiento y características seleccionadas (MejorSolucionTemporal).
4. Mientras que k (variable de iteración) sea menor que el umbral especificado como parámetro.
   a) Instanciamos un diccionario vacío llamada *rendimiento_s_t*.
   b) Iteramos sobre los atributos:
      - Instanciamos una lista vacía llamada *solución_temporal*.
      - Si la variable *solución_actual* no contiene la variable y el tamaño de la solución actual es mayor que cero, incluimos en la *solución_temporal* la *solución_actual*
      - Si no es así, incluiremos en la *solución_temporal* la variable sobre la que estamos iterando.
      - Aplicamos la validación cruzada (método *aplicar_validacion_cruzada*) para evaluar la *solución_temporal* y guardar su rendimiento en el diccionario *rendimiento_s_t* junto a

dicha *solución_temporal*.

- La siguiente línea de código es simplemente para representar los resultados con pandas.

c) Una vez hemos terminado de iterar sobre las variables, seleccionamos la mejor solución temporal y actualizamos la solución actual, además de la k para seguir iterando con el while.

5. Una vez ha finalizado de iterar, nos deshacemos de los duplicados y devolvemos los mejores resultados.

Este es uno de los métodos principales del trabajo. En este, iteramos sobre las variables y, llamando a *aplicar_validacion_cruzada*, obtenemos los resultados deseados del rendimiento para evaluar las soluciones temporales y así poder diferenciar mejores soluciones frente a otras.

Además, cuando se devuelven los resultados, obtenemos los mismos ordenados a razón del rendimiento de manera descendente, facilitando la visualización de aquellas soluciones que son mejores.

A continuación, mostramos en pseudocódigo el método aplicar_validacion_cruzada, con el que entrenamos ciertos subconjuntos a partir de la implementación con árboles de decisión, repitiendo el proceso un número de veces y devolviendo el resultado promedio.

aplicar_validacion_cruzada

Entrada:

- datos (Datos en formato excel).
- variables (La solución temporal de la iteración que corresponda).
- n_exp (El número de repeticiones del experimento por validación crucada, siendo 10 el valor por defecto).
- cv (Número de folds a considerar en la validación cruzada, de nuevo 10 por defeto).
- métrica (Representa la métrica de evaluación a utilizar. Por defecto *"balanced_accuracy"*)

Salidas:

- Devuelve la media de scores proporcionados por el subconjunto elegido en el algoritmo.

Algoritmo:

1. Seleccionamos el conjunto de datos de entrada del excel.
2. Seleccionamos el subconjunto de características que queremos evaluar.
3. Implementamos el algoritmo de árboles de decisión con la librería sklearn (más concretamente hacemos uso de tree y model_selection) como algoritmo de aprendizaje.
4. Como parte de la validación cruzada, instanciamos una variable array_scores, e iteramos hasta n_exp veces para, finalmente ir almacenando las medias de los resultados en esta variable
5. Finalmente se devuelve el resultado promedio de los almacenados en la variable array_scores devuelto por el algoritmo.

Como se ha mencionado antes, el método aplicar_validacion_cruzada, devuelve el score promedio resultante de aplicar el algoritmo de árboles de decisión, a través del método cross_val_score. Este método recibe el árbol, las características seleccionadas, el objetivo, el número de folds a considerar y el parámetro que representa la métrica usada (parámetro scoring instanciado a la variable métricas).

Una vez finalizada esta ejecución existe la posibilidad de visualizar más resultados (como la evolución de la capacidad predictiva en el conjunto de datos dependiendo del número de características escogidas con respecto al rendimiento medio, o la representación gráfica de los rendimientos máximos/mínimos y su dispersión).

Sin embargo, antes de pasar a mostrar estos métodos de visualización de resultados, se enseñará la implementación del otro algoritmo de búsqueda secuencial: Sequential Floating Forward Selection (de ahora en adelante SFFS).

SFS es la base sobre la que construiremos SFFS, ya que con lo que respecta a la adición de atributos, el core es prácticamente el mismo. En este método de búsqueda secuencial, tenemos dos direcciones: la dirección principal que en nuestro caso será hacia adelante (como el caso de SFS). Sin embargo, este método no es tan simple: tiene la opción de retroceder para eliminar una característica en el caso de que sin esta, el score vaya a aumentar.

Para ello, se debe descomentar en el notebook (en la segunda celda) el método Seleccionar_características_sffs y comentar el método Seleccionar_características_sfs. Este llamará primero al método Seleccionar_caracteristicas_sffs para proceder a aplicar el algoritmo. Su pseudocódigo se presenta a continuación:

Seleccionar_características_sffs

Entrada:

- datos_csv (Datos en formato excel).
- names (Atributos a seleccionar, en principio vacía).
- umbral (El umbral a utilizar por SFS y SFFS. Default = 10).
- n_exp (El número de repeticiones del experimento por validación crucada, nuevamente 10 por defecto).
- cv (Número de folds a considerar en la validación cruzada, de nuevo 10 por defeto).
- métrica (Representa la métrica de evaluación a utilizar. Por defecto *"balanced_accuracy"*)

Salidas:

- Variables necesarias para poder aplicar la solución cruzada y el algoritmo SFFS.

Algoritmo:

1. Instanciamos una variable *atributos* a los pasados por

parámetros.

2. Condición si la lista está vacía.

a) Si está vacía, la rellenamos con las columnas del excel (que contendrán las características)

b) Si no está vacía, continuamos con la ejecución

3. Si la métrica es *none*, aplicamos por defecto *balanced_accuracy*.

4. Devolvemos el método *aplicar_SFFS*.

Como se explicó previamente, este método, al ser una analogía de Seleccionar_características_sfs, es el encargado de inicializar las características que posteriormente serán utilizadas para aplicar el algoritmo SFFS. Se han realizado sendos métodos con la intención de hacer la depuración del código de manera más sencilla.

A continuación, mostramos el pseudocódigo del método aplicar_SFFS, responsable de la ejecución del algoritmo Sequential Floating Forward Selection.

aplicar _SFFS

Entrada:

- datos (Datos del Excel pasados como parámetro).
- variables (Atributos proporcionados por el método Seleccionar_características_sffs).
- umbral (El umbral a utilizar. Es 10 por defecto).
- n_exp (El número de repeticiones del experimento por validación crucada, nuevamente 10 por defecto).
- cv (Número de folds a considerar en la validación cruzada, de nuevo 10 por defeto).
- métrica (Representa la métrica de evaluación a utilizar. Por defecto *"balanced_accuracy"*)

Salidas:

- Devuelve los diez mejores resultados (mejor rendimiento menor tamaño) con el algoritmo SFFS.

Algoritmo:

1. Instanciamos una variable atributos en la que incluimos las variables proporcionadas por el método Seleccionar_caracteristicas_sffs.
2. Instanciamos una variable solucion_actual en la que iremos almacenando la solución por cada iteración.
3. Instanciamos dos listas en las que incluimos las variables que se van añadiendo/eliminando por iteración.
4. Instanciamos la variable column_names en la que, utilizando pandas, devolveremos los datos de tamaño rendimiento y características seleccionadas (MejorSolucionTemporal).
5. Mientras que k (variable de iteración) sea menor que el umbral especificado como parámetro y que la longitud de las variables pasadas por parámetros sea igual que las añadidas.
   - Instanciamos un diccionario vacío llamada rendimiento_s_t (análogo al del método aplicar_sfs)
   - Instanciamos otro diccionario, donde almacenamos los candidatos a ser eliminados.
   - Establecemos la condición de parada (len(añadidos)!=len(variables))
   - Iteramos sobre las variables:
     - Instanciamos una lista vacía llamada solución_temporal.
     - Si la variable sobre la que estamos iterando no se encuentra ni en solucion_actual ni en añadidos y len(solución_actual)>0, incluimos en la solución_temporal la solución_actual.
     - Si no es así, incluiremos en la solución_temporal la variable sobre la que estamos iterando.
     - Aplicamos la validación cruzada como se hacía en aplicar_SFS para evaluar la solución_temporal y guardar su rendimiento en el diccionario rendimiento_s_t junto a dicha solución_temporal.

   c) Seleccionamos la mejor solución temporal y actualizamos la solución actual.

   d) Actualizamos la lista de características añadidas.

   e) Creamos un diccionario auxiliar para acceder a la variable que potencialmente borraremos.

   f) Iteramos sobre las variables:
      - Si no hemos eliminado la variable: la eliminamos de la solución actual (una copia de esta variable).
      - Evaluamos el rendimiento una vez hemos eliminado dicha variable.

   g) Comprobamos si el rendimiento tras borrar la variable es mejor frente al que se obtiene manteniéndola. Nos quedaremos con la opción que de mejor score.
6. Actualizamos k para seguir iterando.
7. Una vez ha finalizado de iterar, nos deshacemos de los duplicados y devolvemos los mejores resultados ordenados por el rendimiento.

Nuevamente, este método llama a aplicar_validacion_cruzada para evaluar el rendimiento, que es el mismo al que se llama en SFS, por lo que no reiteraremos su explicación.

Una vez expuesta la funcionalidad principal de los métodos de búsqueda, pasamos a mostrar los métodos de visualización de resultados:

plot_linea

Entrada:

- dataframe (La solución devuelta por el método correspondiente SFS o SFFS).

Salidas:

- Devuelve una gráfica con la evolución de la capacidad predictiva en el conjunto de datos dependiendo del número de características escogidas con respecto al rendimiento medio.

Algoritmo:

1. Instanciamos una lista vacía llamada medias en la que almacenaremos el rendimiento medio.
2. Instanciamos una variable x en la que almacenamos los tamaños.
3. Iteramos sobre x (los tamaños).
   a. Actualizamos el tamaño.
   b. Añadimos los rendimientos de x y hacemos la media.
   c. Mostramos la gráfica.

Como se mencionó previamente, la gráfica representa la evolución de la capacidad predictiva en el conjunto de datos dependiendo del número de características escogidas con respecto al rendimiento medio. Se obtiene una vez el algoritmo SFS ha finalizado su ejecución (o el SFFS).

A continuación, vemos el pseudocódigo del "violin_plot"

violin_plot

Entrada:

- dataframe (La solución devuelta por el método correspondiente SFS o SFFS).

Salidas:

- Devuelve una gráfica que muestra la dispersión del rendimiento dependiendo del número de características seleccionadas.

Algoritmo:

1. Instanciamos una lista valores, que serán los rendimientos devueltos por el método aplicar_SFS/SFFS.
2. Instanciamos una lista tamaños con las características.
3. Iteramos sobre las características:
   a) Capturamos todas las filas con tamaños igual a 1.
   b) Obtenemos los rendimientos de dichas filas y los almacenamos en valores.
4. Mostramos la gráfica.

Esta gráfica tiene el fin de mostrar la dispersión del rendimiento dependiendo del número de características seleccionadas.

IV. RESULTADOS + EXTRA

En esta sección se explicarán tanto los resultados obtenidos por los métodos desarrollados, como la parte extra que se ha implementado en el proyecto. El trabajo adicional (extra) que se ha implementado es el siguiente: Selección de la métrica a utilizar y la representación gráfica de resultados.

- Se muestran las tablas con los resultados tras la ejecución del código.
- Se realizan dos representaciones gráficas (implementación explicada en la sección METODOLOGÍA):
  A) Plot_linea

  Pretende mostrar el rendimiento medio que se tiene con un número de características escogidas.

  B) Violin_plot^x

  Pretende mostrar la dispersión del rendimiento dependiendo del número de características seleccionadas por el algoritmo.

1) Se reproducirán ambos métodos (SFS y SFFS) con parámetros de igual valor. Estos son:

datos_csv = titanic.csv
names = []
umbral = 10
n_exp = 1
cv = 3
metrica = none
x = 10

2) Se aumentará el n_exp y el cv a 10 y 10 para ver la efectividad que se tiene cuando ampliamos el número de experimentos que se realizan con la validación cruzada.

- Tablas de Resultados

1) SFS

Aplicamos SFS con los parámetros definidos en esta sección.

MejorSolucionTemporal	Tamaño	Rendimiento
(Title, Deck, Initial, SibSp)	4	0.813824
(Initial, SibSp, Deck, Sex, Title)	5	0.813824
(Initial, SibSp, Deck, Sex, Is_Married, Title)	6	0.813824
(Initial, SibSp, Deck,	5	0.812913

Is_Married, Title)		
(Deck, Initial, SibSp)	3	0.811260
(Deck, Initial, Sex, SibSp)	4	0.811260
(Deck, Initial, Is_Married, SibSp)	4	0.811260
(Initial, SibSp, Deck, Sex, Title, Fare_cat)	6	0.809438
(Initial, SibSp, Deck, Alone, Sex, Title)	6	0.808551
(Initial, SibSp, Deck, Alone, Title)	5	0.808551

Ampliando los parámetros cv y n_exp a 10.

MejorSolucionTemporal	Tamaño	Rendimiento
(Fare_cat, Initial, Title, SibSp, Sex, Deck, Is_Married)	7	0.816047
(Fare_cat, Initial, Title, SibSp, Deck)	5	0.815476
(Fare_cat, Initial, Title, SibSp, Sex, Deck)	6	0.815476
(Fare_cat, Initial, Title, SibSp, Deck, Is_Married)	6	0.815476
(Fare_cat, Initial, SibSp, Sex, Deck)	5	0.811226
(Fare_cat, Initial, Title, Alone, SibSp, Sex, Deck, Is_Married)	8	0.809879
(Fare_cat, Initial, Title, Alone, SibSp, Deck)	6	0.809593
(Fare_cat, Initial, Title, SibSp, Deck, Is_Married)	7	0.809451
(Deck, Fare_cat, SibSp, Initial)	4	0.809184
(Fare_cat, Initial, SibSp, Deck, Is_Married)	5	0.808898

Como podemos apreciar, ampliando el número de experimentos que se realizan, obtenemos un mejor rendimiento frente al que se obtiene manteniendo el número de folds a 3 y las repeticiones a 1.

2) SFFS

Aplicamos SFFS con los parámetros definidos arriba

MejorSolucionTemporal	Tamaño	Rendimiento
(Title, SibSp, Deck, Is_Married, Sex)	5	0.816197
(SibSp, Title, Deck, Sex)	4	0.816197
(SibSp, Title, Initial, Deck)	4	0.814735
(Title, SibSp, Deck, Sex, Initial)	5	0.813824
(Title, SibSp, Deck, Is_Married, Initial)	5	0.812913
(Is_Married, SibSp, Initial, Deck)	4	0.811260
(SibSp, Initial, Deck)	3	0.811260
(SibSp, Sex, Initial, Deck)	4	0.811260
(Title, Alone, SibSp, Deck, Is_Married, Sex)	6	0.810924
(Title, Alone, SibSp, Deck, Sex)	5	0.810924

Aplicamos cv y n_exp = 10

MejorSolucionTemporal	Tamaño	Rendimiento
(Title, Deck, Initial, SibSp, Fare_cat)	5	0.815984
(Title, Sex, Deck, Initial, SibSp, Fare_cat)	6	0.815619
(Is_married, Title, Deck, Initial, SibSp, Fare_cat)	6	0.815476
(Title, Sex, Deck, SibSp, Fare_cat)	5	0.813590
(Sex, Deck, Initial, SibSp, Fare_cat)	5	0.810940
(Alone, Title, Deck, Initial, SibSp)	5	0.810635
(Alone, Title, Deck, Initial, SibSp, Fare_cat)	6	0.809736
(Is_Married, Deck, Initial, SibSp, Fare_cat)	5	0.809470
(SibSp, Fare_cat, Deck, Initial)	4	0.809327
(Title, Deck, Initial, SibSp, Age_band)	5	0.808791

Como podemos apreciar, tenemos peores rendimientos habiendo aumentado el número de repeticiones.

Sin embargo esto no significa que sea mejor, sino que el factor de aleatoriedad hace que se modifiquen los rendimientos en mayor medida y por tanto tengamos resultados que no se ajustan a la realidad.

- Gráficas Resultantes

A continuación, mostramos la gráficas resultantes (plot_linea) de ejecutar SFS y SFFS con los parámetros iniciales (sin modificar el cv ni el n_exp):

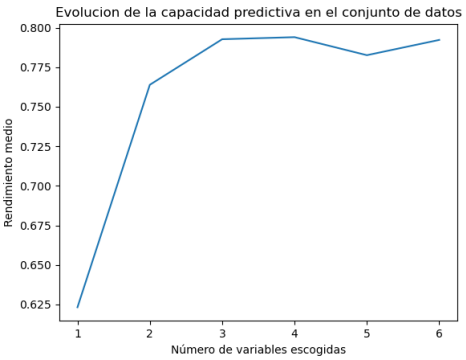

Figura 4. plot_linea con aplicación de SFFS

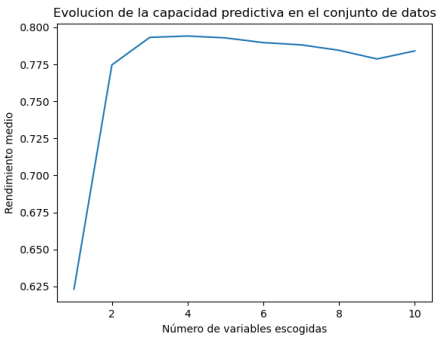

Figura 5. plot_linea con aplicación de SFS

Como se ha visto previamente, la aplicación de SFS tiene sus pros y sus contras con respecto al algoritmo SFFS. SFFS es bastante más costoso con respecto a recurso computacional y tiempo. Sin embargo, también ofrece resultados más precisos que SFS, ya que tiene la opción de eliminar variables considerando que su eliminación pueda significar una optimización del resultado final.

Pese a tener unos resultados parejos en la gráfica, se puede apreciar que SFS nos devuelve resultados de tamaño 10. Sin embargo, SFFS a partir del tamaño 6 no se dan resultados.

A continuación, mostramos la gráficas resultantes (violin_plot) de ejecutar SFS y SFFS con los primeros parámetros:

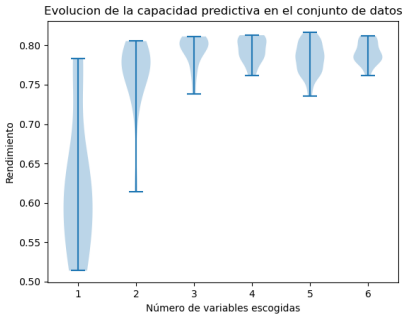

Figura 6. violin_plot con aplicación de SFFS

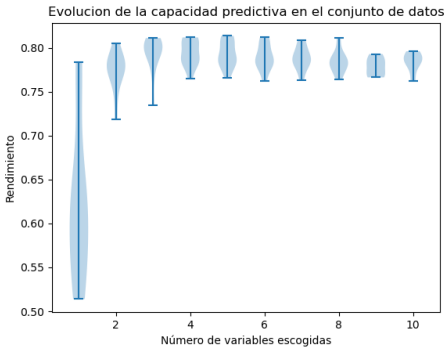

Figura 7. violin_plot con aplicación de SFS

Nuevamente, nos vamos a reiterar en el hecho que SFFS, pesa a tener un coste computacional mejor, devuelve mejores resultados. También se ve claramente en esta gráfica: podemos apreciar que la gráfica resultante de aplicar SFFS, tiene más resultados en secciones superiores (la forma se hace más ancha por la parte de arriba), lo que significa que tienen un número mayor de rendimientos mejores. Sin embargo, vemos que las gráficas resultantes de aplicar SFS tienen una forma que se ensancha en la parte baja, lo que quiere decir que tienen un mayor número de resultados con peor rendimiento. Como resultado vemos que es el esperado, ya que como se ha mencionado en varias ocasiones, SFFS devuelve mejores resultados que SFS al poder corregir pasos en la iteración.

A continuación, pasamos a ampliar el valor de los parámetros cv y n_exp a 10. Hay que tener en cuenta que existe un cierto índice de aleatoriedad, el cual disminuye cuando aumentamos el valor de dichas variables:

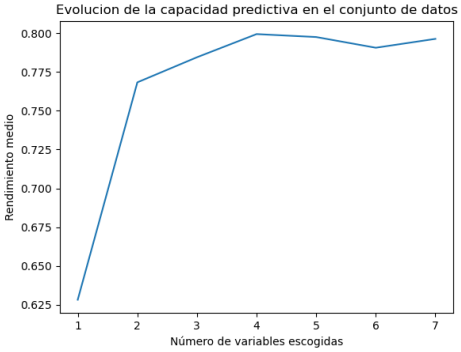

Figura 8. plot_linea con aplicación de SFFS.

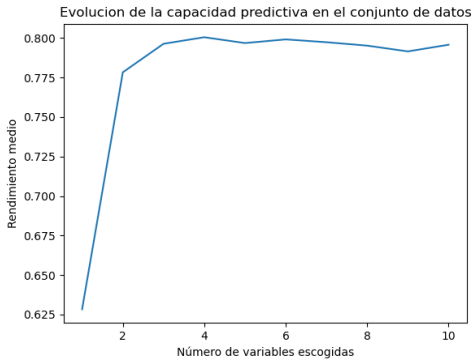

Figura 9. plot_linea con aplicación de SFS.

Habiendo aumentado el número de veces que repetimos el experimento en la validación cruzada y el número de folds, vemos que se es posible considerar una variable más como resultado (comparar figura 8 y figura 4). Además, comparando nuevamente los algoritmos SFFS y SFS, tenemos el mismo razonamiento que se hizo sin ampliar los parámetros: con SFFS vemos que a partir de 7, no tenemos resultados óptimos. Sin embargo, con SFS nos dan resultados hasta la variable 10.

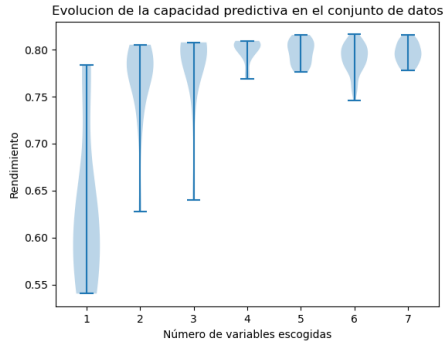

Figura 10. plot_violin con aplicación de SFFS

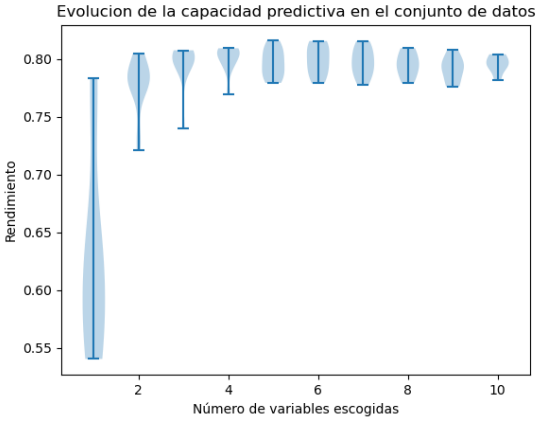

Figura 11. violin_plot con aplicación de SFS

Nuevamente, si comparamos las gráficas habiendo modificado parámetros (comparación figura 10 vs figura 6), de nuevo vemos que se es capaz de obtener una variable más (de tamaño 6 pasa a 7). Además, si volvemos a comparar SFS vs SFFS, tenemos el mismo razonamiento: con SFFS obtenemos mejores rendimientos que en SFS (se puede apreciar en la anchura de las formas. En SFFS tenemos formas más anchas en la parte de arriba, es decir, zonas con mayor rendimiento).

V. CONCLUSIONES

La conclusión principal de todo el proyecto es que un número mayor de características existentes en el conjunto de entrenamiento no implica un mejor rendimiento. Como se ha mencionado antes, nos puede conducir al sobreajuste y por tanto tener resultados que no son realistas.

Como hemos podido ver a lo largo del documento, se ha llevado a cabo la implementación de las estrategias de búsqueda secuenciales Sequential Forward Selection (SFS) y Sequential Floating Forward Selection (SFFS).

Se han desarrollado conocimientos sobre el método de envoltura, sendos métodos de búsqueda y sobre la validación cruzada. Todo esto, aplicado a Python (ya que sus librerías son muy eficientes para trabajar en el campo de machine learning, además de tener una documentación muy clara), por lo que ya no solo se ha tratado el tema teórico, sino que se ha profundizado también en la práctica (librerías que no se habían tratado previamente, etc.).

Para la implementación, se han intentado desarrollar los métodos necesarios de manera que se puedan destacar las diferencias entre SFS y SFFS fácilmente. Sin embargo, no se han llevado a cabo implementaciones que disminuyan el coste computacional de los métodos (métodos recursivos, paralelización de código, etc.).

Cabe mencionar la existencia de una componente de aleatoriedad, que disminuye con el número de repeticiones que se llevan a cabo en la validación cruzada. Este factor, puede llevarnos a resultados poco precisos si no se han realizado experimentos suficientes como para obtener valores realistas.

Con respecto al ámbito teórico, finalizamos con la misma afirmación que se ha arrastrado durante todo el documento: SFFS devuelve resultados más precisos que SFS, aunque también tiene un coste computacional mayor. Esto es debido a que el algoritmo busca optimizar siempre el resultado corrigiendo pasos que haya podido dar.

VI. ANEXO MARCO TEÓRICO

En esta sección se pretende realizar una extensión del trabajo con un trasfondo algo más teórico.

Durante todo el documento, se profundiza sobre todo en las técnicas y en los métodos de búsqueda secuenciales (además del algoritmo de árboles de decisión y la validación cruzada).

Todas estas técnicas y algoritmos son un aspecto muy concreto e ínfimo de la selección de características/variables en el ámbito del aprendizaje automático. Sin embargo, no se entra en detalle en una de las partes más genéricas del trabajo: los métodos de selección de características (métodos de filtrado, métodos de envoltura y métodos integrados).

Es por ello que se intentará entrar en detalle en cada uno de ellos, mencionando ventajas y desventajas que puedan representar y cuál sería mejor en una situación en concreto.

Antes de entrar en detalle y comenzar a enumerar los métodos, se contextualizará la materia, empezando por introducir nuevamente la selección de características.

Como se vio previamente, la selección de características permite reducir el conjunto total de variables de entrada a un número más reducido de datos. Esto se hace por varias razones, entre las que destacan:

- El coste computacional que conlleva tratar todos el conjunto de características (CPU, memoria RAM, etc.) es mayor al que tiene tratar un subconjunto de características.
- La sencillez que aportan menos variables frente a la complejidad que se tiene al tener que procesar más características.
- Tratar todas las características incluidas en el conjunto de datos, hace que aumente el riesgo de que se sufra de sobreajuste. Esto quiere decir (como ya se había mencionado en la introducción), que es posible que los datos resultantes se centren demasiado en el conjunto de entrenamiento proporcionado (no se ajustan a casos reales).

Para hacer frente a este problema, no existe una solución genérica. Existen diversas soluciones que pueden ser aplicadas a la selección de características en modelos predictivas. Cada una tiene sus ventajas y sus inconvenientes, por lo que cada método puede funcionar mejor que otro (o peor) dependiendo del tipo de problema que se debe afrontar.

Estos métodos o técnicas se pueden resumir en tres grupos fundamentales, métodos de filtro, métodos integrados y métodos de envoltura (el cual ha sido objeto de estudio en este trabajo). A continuación entramos en detalles con cada uno de ellos:

1) Filter Methods[xi]

Los "Filter Methods", o métodos de filtro, son un tipo de método que se utiliza generalmente como un paso de preprocesado (ver figura 12). La selección de las características no depende de ningún algoritmo de aprendizaje que se vaya a utilizar, ya que este proceso es previo a la aplicación de dicho algoritmo.

La selección del subconjunto de características depende de las propiedades de estas, ya que se realizan pruebas estadísticas para poder evaluar los distintos scores resultantes.

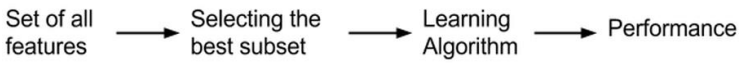

Figura 12. Método de filtro. Fuente: https://upload.wikimedia.org/wikipedia/commons/2/2c/Filter_Methode.png

Este método es en general muy rápido. Apenas tiene coste computacional y además los resultados pueden ser utilizados por cualquier algoritmo de aprendizaje.

Sin embargo, no se detectan las correlaciones que pueden existir entre variables (que es el motivo principal por el que este método sea utilizado en el preprocesado).

2) Wrapper Methods[xii]

Los "Wrapper Methods", o métodos de envoltura, son técnicas de selección de características que, a diferencia del método de filtro, sí depende del algoritmo de aprendizaje que se utiliza para evaluar cada subconjunto por entrenamiento.

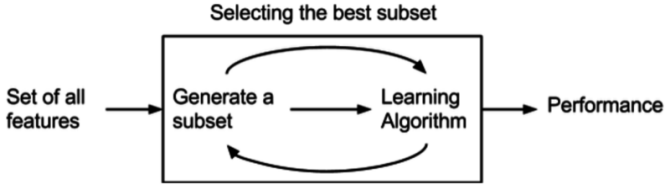

Figura 13. Método de envoltura. Fuente:https://ligdigonzalez.com/metodos-de-seleccion-de-caracteristicas-machine-learning/

Como hemos visto a lo largo de los experimentos, estos métodos utilizan estrategias de búsqueda para ir seleccionando posibles subconjuntos de características, que van siendo evaluados basándose en el rendimiento devuelto por un determinado algoritmo. En nuestro trabajo, utilizamos árboles de decisión como algoritmo de entrenamiento y validación cruzada como método de evaluación. Además, como estrategias de búsquedas tenemos en consideración aquellas que son secuenciales (búsqueda hacia atrás, búsqueda hacia adelante y búsqueda mixta), aunque es posible utilizar otros enfoques como algoritmos genéticos, algoritmos voraces[xiii], etc.

Figura 14. Validación cruzada. Fuente: https://i1.wp.com/www.statworx.com/wp-content/uploads/k-fold-cross-validation-1024x416.png?resize=456%2C185&ssl=1

Estos métodos, a diferencia de los métodos de filtro, sí tienen en cuenta la posible correlación entre dos características, por lo que pueden encontrar soluciones óptimas. Suelen ser más precisos que los métodos de filtro.

Por otra parte, es computacionalmente costoso, sobre todo en búsquedas que puedan ser más exhaustivas.

3) *Embedded Methods*[xiv]

Los "Embedded Methods", o métodos integrados, seleccionan las características en el mismo modelo de aprendizaje.

Durante este proceso de entrenamiento, se almacena la información necesaria para poder evaluar los subconjuntos de características para poder escoger uno mejor frente a otro que dé peores resultados.

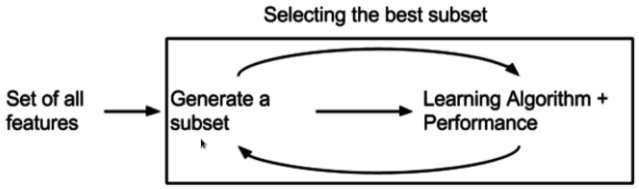

Figura 15. Método integrado. Fuente: https://upload.wikimedia.org/wikipedia/commons/b/bf/Feature_selection_Embedded_Method.png

Estos métodos buscan cubrir las desventajas que tienen tanto los métodos de filtro como los métodos de tipo envoltura. Por tanto, es de esperar que las ventajas que tiene este método combinan aquellas que tienen ambos métodos:

1) Al igual que los métodos de envoltura, tiene en cuenta la correlación existente entre características, desventaja que por otro lado tiene el método de filtro.
2) Al igual que los métodos de filtro, tienen un coste computacional muy bajo (nuevamente, a diferencia de los métodos de envoltura).
3) A diferencia de los métodos de filtro, esta técnica da resultados más precisos, como era el caso del método de envoltura.
4) Son propensos a que no se den casos de sobreajuste.

Como hemos podido apreciar, cada método tiene sus ventajas y sus desventajas, por lo que cada uno puede ser útil en un caso mientras que otro método puede ser computacionalmente más costoso o dar soluciones menos precisas. Por ejemplo, al dar resultados menos precisos pero al ser computacionalmente menos costoso se puede utilizar el método de filtro en el preprocesado, mientras que los métodos de envoltura y de integración se pueden utilizar en el proceso de aprendizaje.

Esto, no solamente depende del problema que se esté llevando a cabo, si no también del algoritmo que se esté utilizando que puede aumentar/disminuir la complejidad de la ejecución.

Cabe mencionar que la complejidad de los algoritmos de búsqueda es muy alta, por lo que se necesita de paralelización/optimización de código para disponer de un tiempo de ejecución razonable.

VII. REFERENCIAS

[i] Motivación de selección de características, https://es.wikipedia.org/wiki/Selecci%C3%B3n_de_variable#M%C3%A9todo_de_filtro

[ii] Métodos principales de búsquedas de variables, https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-an-introduction-1d8dc6d86c16

[iii] SFS http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/

[iv] SFFS http://rasbt.github.io/mlxtend/user_guide/feature_selection/SequentialFeatureSelector/

[v] Práctica 1 de la asignatura, https://www.cs.us.es/docencia/aulavirtual/mod/resource/view.php?id=1370

[vi] API sklearn, https://scikit-learn.org/stable/modules/classes.html

[vii] Api pandas, https://pandas.pydata.org/docs/reference/index.html

[viii] Api statistics https://docs.python.org/3/library/statistics.html

[ix] Api matplotlib https://matplotlib.org/3.2.1/api/index.html

[x] Diagrama de violín, https://datavizcatalogue.com/ES/metodos/diagrama_de_violin.html

[xi] Método de filtro, https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-filter-methods-f248e0436ce5

[xii] Método de envoltura, https://heartbeat.fritz.ai/hands-on-with-feature-selection-techniques-wrapper-methods-5bb6d99b1274

[xiii] Metaheurísticas comunes https://es.wikipedia.org/wiki/Metaheur%C3%ADstica#Metaheur%C3%ADsticas_comunes

[xiv] Métodos integrados https://heartbeat.fritz.ai/hands-on-with-feature- selection-techniques-embedded-methods-84747e814dab