Memoria: P1 - Análisis Inteligente de Datos y Selección de Modelos

Grupo: Luis Gómez Dios, Pedro Arias Redondo, Mateo Fraguas Abal.

1. Pre-procesado de los datos

En primer lugar y antes de nada debemos hacer un preprocesado del dataset para asegurarnos de que los datos estén limpios, completos y en un formato adecuado para el aprendizaje del modelo. Para esto debemos asegurarnos de que no hay variables repetidas o inconsistencias, ni huecos en los datos (MissinValues), además de comprobar si hay datos atípicos y pensar cómo trataremos con ellos. También prestaremos atención a el desbalance de datos ya que puede que tengamos menor cantidad de observaciones de una clase que de otra lo que sobreentrenaría o infraentrenaría el modelo en alguno de las clases. Por último normalizaremos los datos.

-1.1. Acceso a los datos

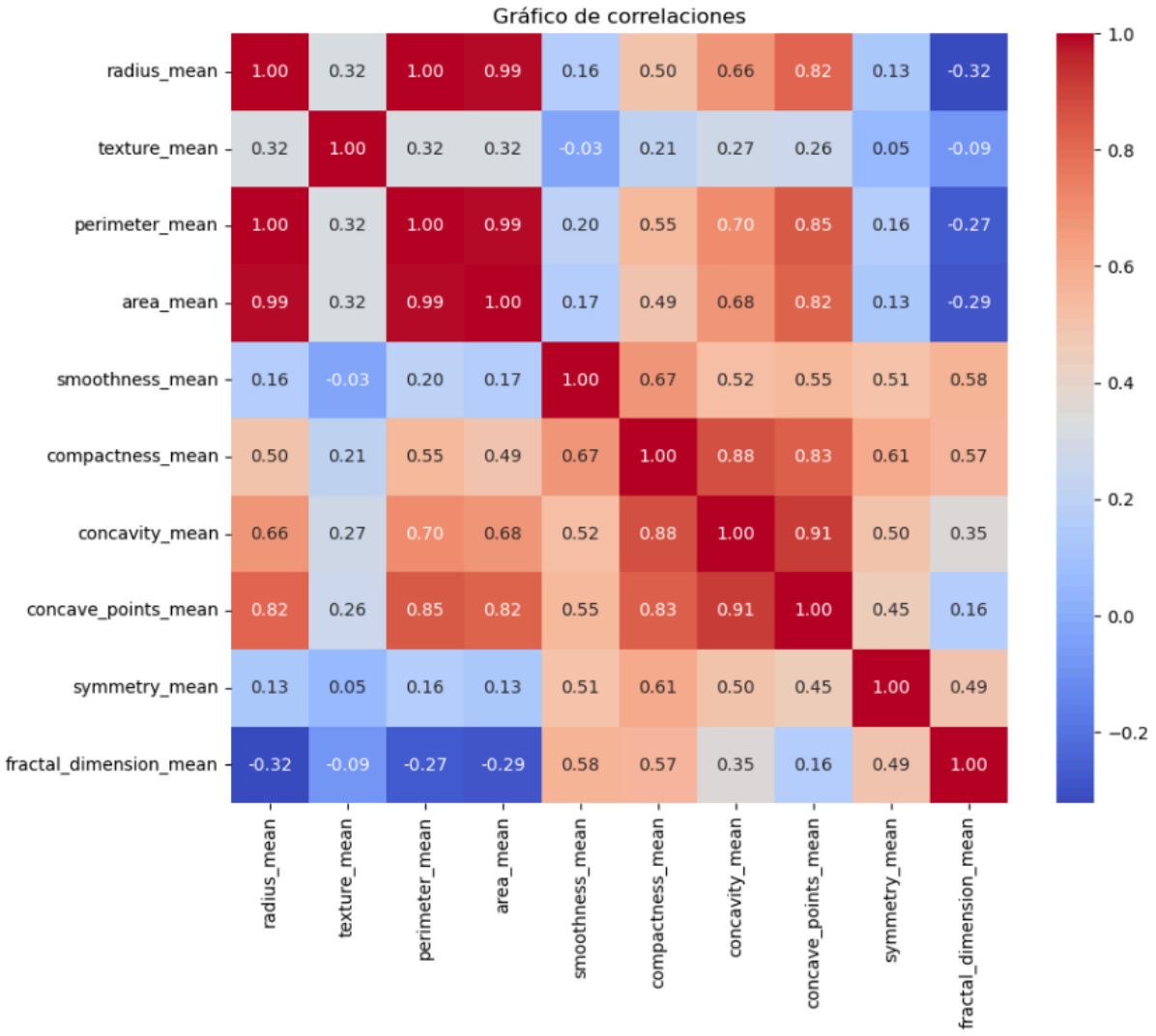
Disponemos de los datos en un fichero .csv y los leemos con la función *read_csv* de la librería *pandas*. Almacenamos el dataset en la variable 'data'.

-1.2. Limpieza inicial de los datos

Empezamos **eliminando todos los datos de la variable 'id'**, esto lo hacemos ya que estos datos tienen como única función identificar a cada paciente y no aportan información de la que se pueda extraer conocimiento sobre los datos. Seguidamente **eliminamos las observaciones duplicadas** ya que estas pueden sesgar el modelo al introducir redundancia.

También **cambiamos las variables categóricas por numéricas** para que puedan ser tratadas por los algoritmos.

Por último hacemos un gráfico para ver las **correlaciones**, en este gráfico observamos que las variables radio, perímetro y área están completamente correladas (como es de esperar) por lo tanto nos quedamos solamente con una de las 3 ya que las otras no aportan información adicional. En este caso nos quedamos con el radio y eliminamos las otras.



-1.3. Gestión de Missing Values

Utilizamos funciones para detectar la existencia de MVs, cuantos hay por cada variable y cuantos hay en total.

Hacemos un estudio sobre distintas opciones para gestionar los MVs:

- **Eliminar las filas con MVs**: Teniendo en cuenta el dataset con el cual estamos trabajando esta no es una opción a considerar debido a que supone la eliminación de muchas observaciones en un dataset que ya es por naturaleza pequeño y además aumenta el desbalance de los datos.
- **Eliminar las columnas con MVs**: Tampoco es una opción que nos parezca interesante porque como dijimos antes en este dataset ya de por sí pequeño no nos interesa eliminar variables enteras.
- **Hacer una imputación simple de los MVs**: Esta opción ya empieza a ser más interesante dado que nos permite mantener el tamaño original del dataset e imputar valores teniendo en cuenta los otros valores ya existentes dentro de la misma variable. Pero también tiene como desventaja que no tiene en cuenta las relaciones entre las variables a la hora de la imputación.

- **Hacer una imputación iterativa de los MVs**: Nos gusta mucho esta opción ya que para predecir los valores faltantes de una variable se basa en los valores faltantes del resto de variables lo que mejora la precisión respecto a otros métodos como la imputación simple. Esta relación entre las variables está especialmente latente en nuestro dataset ya que las variables predictoras son 10 medidas distintas pero de las cuales hay para cada una variables de su media, desviación típica y peor observación. También es un factor a favor para esta opción el hecho de que solo falten valores en las variables de desviación típica porque así se dispone de los valores completos de las medias y peores observaciones para la predicción.
- **Hacer una imputación usando el algoritmo KNN**: Esta opción tiene en cuenta las relaciones entre observaciones pero requiere una elección del parámetro de número de vecinos adecuada además de que por la naturaleza de cómo están distribuidos los MVs nos interesan más las relaciones entre variables que entre observaciones.

Finalmente nos decantamos por la imputación iterativa mediante la función *IterativeImputer*.

-1.4. Detección de datos atípicos

Utilizamos una función para detectar los valores atípicos. Detectamos algunos valores atípicos pero finalmente decidimos no eliminarlos por si pudieran ser relevantes y más adelante en el pre-procesado mitigaremos su impacto mediante la normalización de los datos.
Función ***LocalOutlierFactor*** de sklearn.

-1.5. Gestión de datos desbalanceados

Primero comprobamos cual es la división de datos que tenemos mediante el método 'value_counts'. Este nos indica que hay un total de 282 instancias de tipo benigno (0) y 137 de tipo maligno (1). Esto nos daría una proporción de aproximadamente **32,6% de tipo 0 y 67,3% de tipo 1**. A pesar de no ser un desbalance excesivamente grave nos decantamos por balancearlos debido a la poca cantidad de datos con los que trabajamos.

En el arreglo de datos desbalanceados tenemos tres opciones principales: undersampling, oversampling o una combinación de las anteriores. La primera de estas la descartamos debido al problema ya mencionado de que nos encontramos ante un dataset reducido.

En el apartado de **oversampling** teníamos varios candidatos, pero nos decantamos por la técnica de **SVMSMOTE**. A diferencia de otras técnicas de oversampling esta se centra en la inferencia de datos de la clase minoritaria en las zonas fronterizas, como en las 'esquinas' de esta clase. Gracias a esto, se forman una especie de

concentración de datos nuevos en las “esquinas” del cluster, a diferencia de otros métodos que se centran igual en añadir datos en el centro del cluster o sólo en las zonas limítrofes con otros cluster. Esto ayuda al modelo a predecir la zona en la que se clasificaría esta clase.

Después comprobamos dos técnicas de **combinación de over y under sampling**. **SMOTEENN** y **SMOTETomek**. Ambas utilizan primero SMOTE como técnica de oversampling y después cada una usa “ENN” (Edited Nearest Neighbours) y “Tomek’s link” respectivamente. Nos decantamos por SMOTEENN principalmente porque como está indicado en la documentación, **“ENN” suele eliminar más datos ruidosos que “Tomek’s link”**.

Finalmente creemos que es **mejor usar las técnicas de over y under sampling conjuntas** ya que genera **menos datos sintéticos** por lo que tendremos datos más reales y habremos eliminado el desbalance totalmente. Después del uso de SMOTEENN la división de datos es de un **49,8% (247 instancias) para el tipo 0 y un 50,2% (249 instancias) para el tipo 1**.

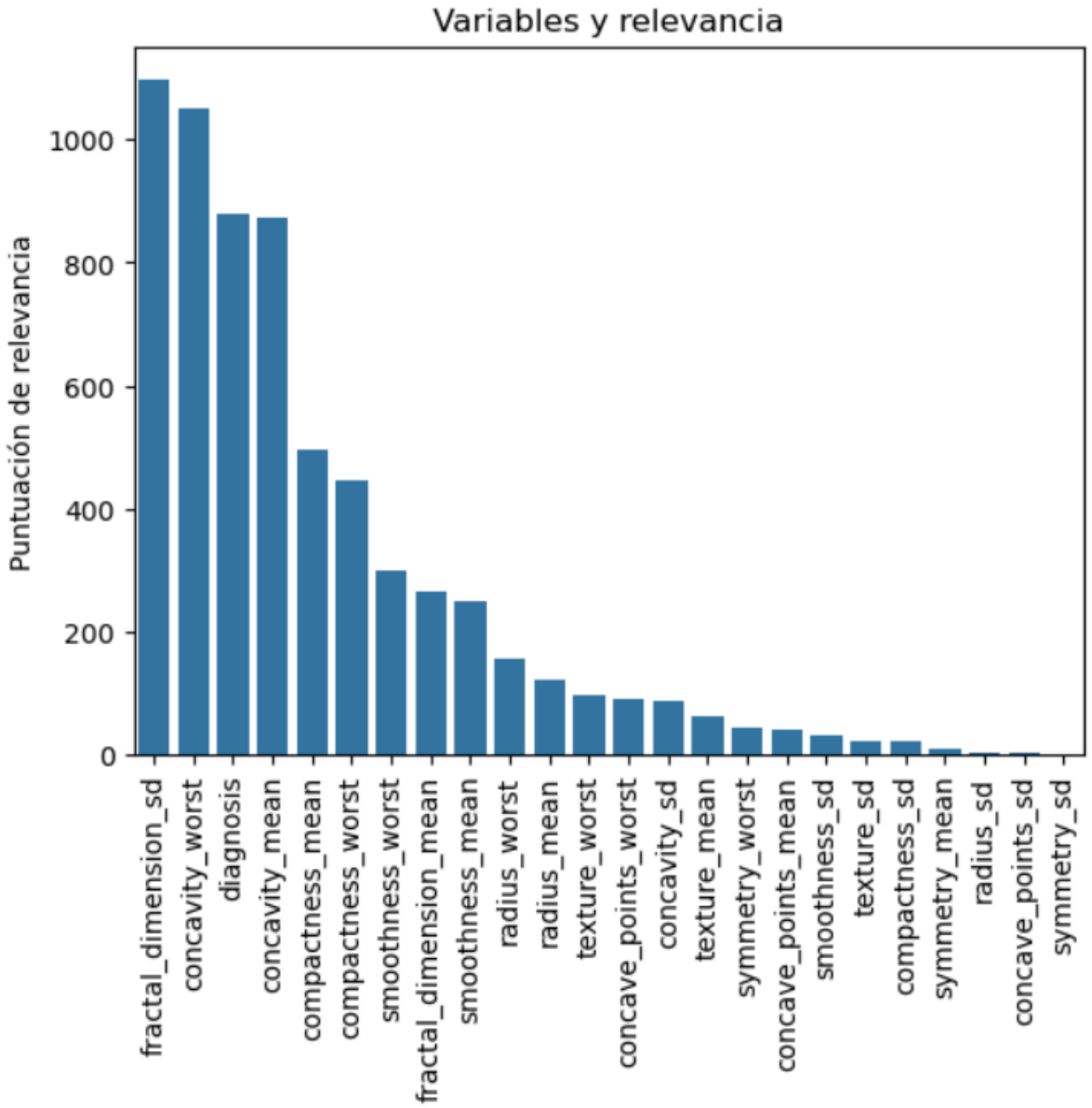
-1.6. Normalización

Utilizamos la función *StandardScaler* de sklearn para normalizar los datos del conjunto de entrenamiento. Esto hace que los datos tengan una media de 0 y una desviación estándar de 1. Esto hace que todas las variables tengan una escala similar equilibrando así las influencias de las variables, como consecuencia de esta mejora de los datos de entrada se mejora a su vez la eficiencia del modelo. Este equilibrio de las variables es especialmente relevante para mitigar la influencia de los datos atípicos.

2. Reducción de dimensionalidad

En este apartado utilizamos la función *SelectKBest* de sklearn por defecto, al estar por defecto utiliza la medida ANOVA F-value para valorar las características más relevantes/representativas.

En este gráfico vemos los atributos ordenados según la puntuación de relevancia que le da a cada uno la función *SelectKBest*. No se ve un método del codo claro en el gráfico pero sí varias caídas considerables en las puntuaciones de relevancia de unas variables a otras.



En el siguiente gráfico vemos la relevancia acumulada para k características, es decir la suma de los valores de las relevancias de las características en la función *SelectKBest* para cada k utilizado.

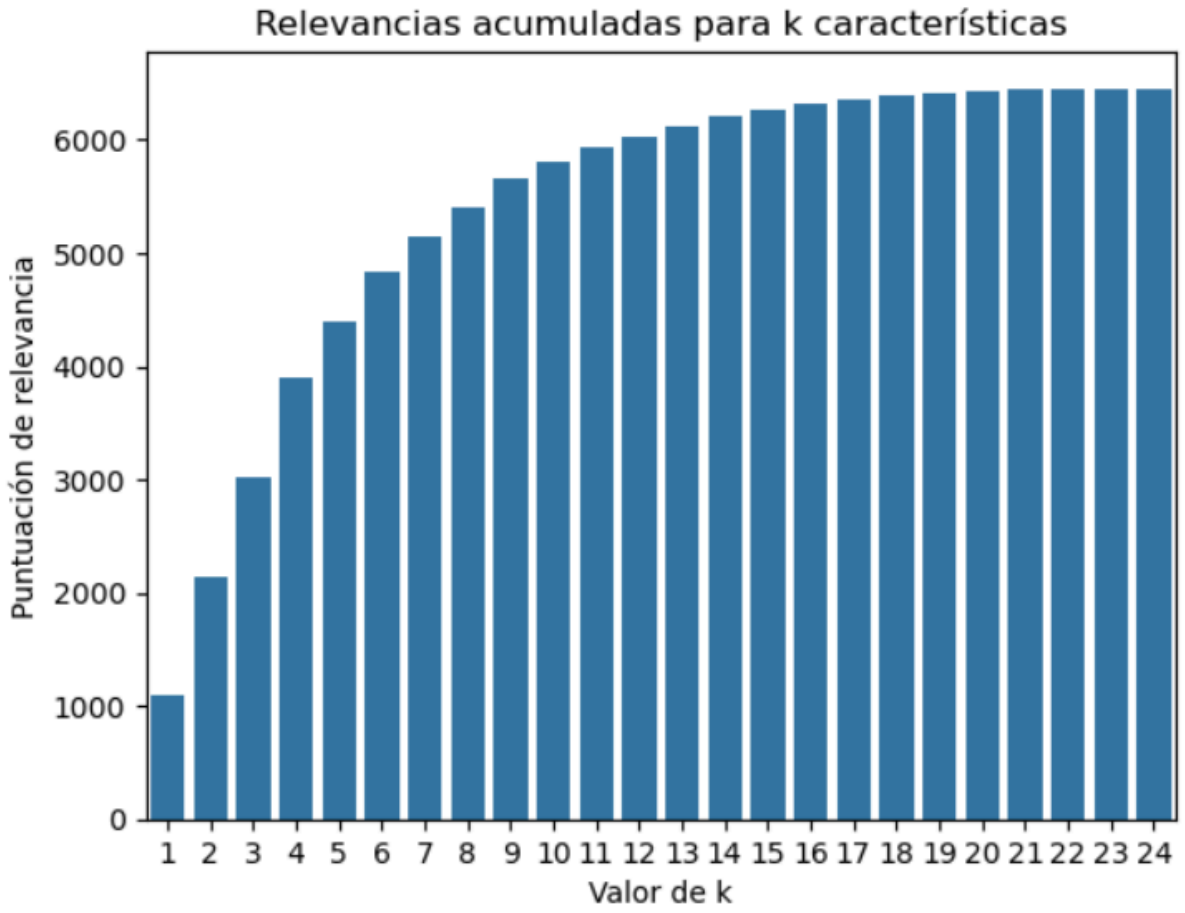
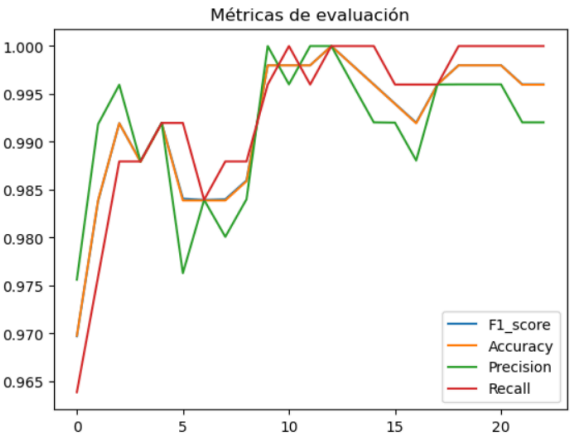
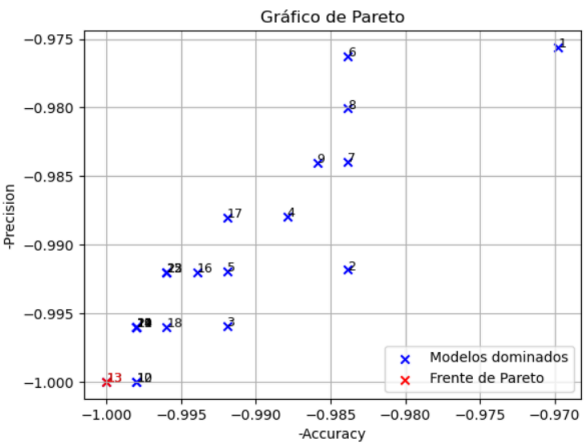


Gráfico pareto

Entrenamos modelos para cada valor de k en [1,24] (en nuestro caso tenemos 24 variables porque ya eliminamos algunas), calculamos distintas métricas de evaluación para cada uno y hacemos un gráfico de Pareto sobre las métricas accuracy y precision.





Como hay un solo modelo en el frente de Pareto y se corresponde al modelo con 13 variables vamos a transformar los datos finalmente con SelectKBest(k=13).

3. Creación y validación de modelos

En esta etapa utilizamos la función *GridSearchCV* de sklearn para hacer los ajustes de hiper-parámetros mediante validación cruzada (con 5 folds en nuestro caso) y para comparar y evaluar los mejores modelos.

Primero seleccionamos los parámetros con los que crearemos, ajustaremos y validaremos varios modelos de predicción. Los parámetros que cambiamos en los modelos son: El número de vecinos, el peso de los vecinos y el leaf_size.

En nuestro caso probamos con todos los modelos dentro de la combinatoria de parámetros:

- Nº de vecinos: Probamos de 1-10 vecinos.
- Pesos: Uniform y distance.
- Leaf_size: 30.

Esto nos da un total de 20 modelos distintos.

Seleccionamos los tres modelos con las configuraciones de parámetros que tuvieron un mejor rendimiento durante la validación cruzada respecto a la media y desviación típica del accuracy.

Posteriormente hacemos un bucle en el que se repite varias veces el siguiente procedimiento para estimar la generabilidad de los tres modelos seleccionados y escoger uno como modelo final:

- División de los datos con una semilla distinta en cada iteración(80%train,20%test).
- Entrenamiento de los modelos y evaluaciones de accuracy y recall.

Consideramos como mejor modelo el modelo con los parámetros: {'leaf_size': 30, 'n_neighbors': 8, 'weights': 'distance'}. Este es el modelo que más veces ha resultado como mejor modelo a lo largo de las iteraciones y el que obtuvo mejores valores de accuracy y recall.

Por último entrenamos este modelo seleccionado con el conjunto de datos completo y dejamos preparado un código que convierte cualquier database con las mismas variables que la inicial en una como la que puede aceptar el modelo final. Para esto primero borramos el ‘id’ y las columnas referentes al ‘área’ y al ‘perímetro’. Después dividimos los datos y los estandarizamos. Para finalizar borramos las columnas de las variables que habíamos eliminado mediante la reducción de dimensionalidad en los datos con los que entrenamos el modelo final.