AUTOML

1. Reto

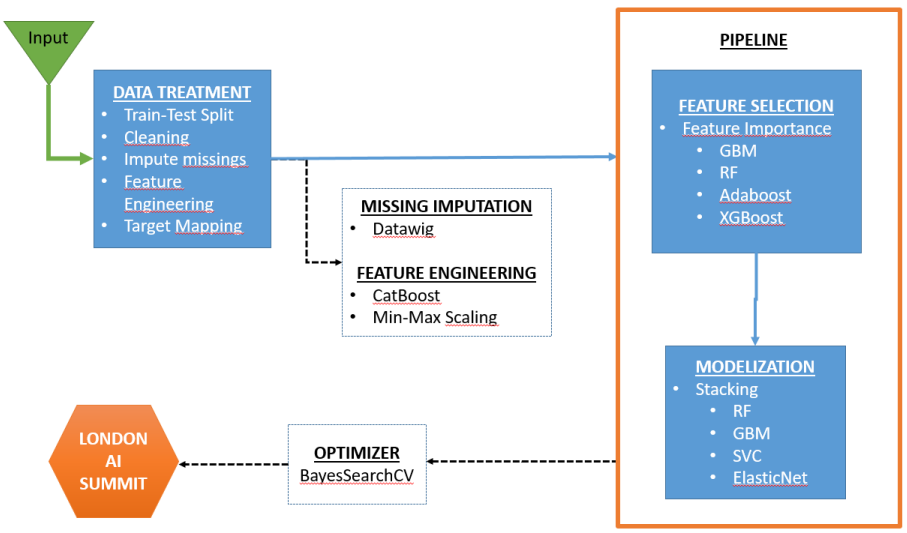
El reto es el diseño de una herramienta de AutoML que cubra el flujo end-to-end de un proceso de modelización automatizada. En este proceso se incluye la carga de datos, cualquier tipo de preprocesado, la selección de variables, el entrenamiento de modelos, la selección de modelos y la predicción. El reto está acotado para problemas de clasificación binaria en los que se valora el AUC como métrica de predicción a maximizar.

2. Propuesta

La idea principal de la propuesta es crear un flujo de arquitectura pipeline que incluya los distintos módulos/pasos del proceso de forma secuencial y se optimice mediante opimización Bayesiana. Cada apartado posee diferentes hiperparámetros como son el número de variables en feature selection o la profundidad máxima de los árboles de un Random Forest en el módulo modelización. La idea es optimizar todo el flujo de forma conjunta para llegar a una combinación ganadora mejor que la que se obtendría optimizando cada módulo por separado. La visión tradicional en Machine Learning es resolver cada módulo del flujo por separado y de forma secuencial. Un ejemplo es realizar las transformaciones de variables independientemente del apartado de modelización y proporcionar un set de datos 'inamovible' para entrenar los modelos.

En un problema de modelización habitual, las decisiones suelen tomarse mediante criterio experto dependiendo del tipo de problema y de datos. El desafío surge cuando cada apartado debe ser generalizado y automatizado, y el criterio experto deja de ser una herramienta. De este modo, se opta por ver el reto como un proceso conjunto y dejarlo todo en manos de la combinación que minimice la función de pérdida.

Otro problema que surge con esta perspectiva es el alto coste computacional al probar cada combinación; hay que limpiar datos, realizar transformaciones, seleccionar variables y ajustar un modelo por cada iteración. Si el número de combinaciones es elevado y la capacidad de computo es limitada, esta opción es inviable teniendo en cuenta que queremos ajustar un modelo en un intervalo de tiempo corto. Es por ello que se opta por introducir la selección de variables y la modelización en el pipeline a optimizar. Todo lo demás se realizará con parámetros por defecto y fuera del flujo.



3. Tratamiento de Datos

En este apartado se van a incluir los siguientes puntos:

- Carga de datos
- Partición Train-Test
- Limpieza
- Imputación de valores nulos
- Encoding de las variables categóricas
- Feature Enginnering
- Reducción de la memoria

Como ya se ha comentado en el apartado anterior, esta parte del flujo se va quedar fuera de la arquitectura pipeline debido al alto coste computacional y las limitaciones de tiempo que nos imponemos. Es por ello que, este módulo se va a optimizar de forma aislada. Todos los subapartados de esta parte del flujo se encuentran en la clase *DataTreatment* del archivo *datatreatment_module.py*.

3.1 Carga de datos

Este apartado consta de los métodos target_correction y types_correction. El primero convierte el target de una variable categórica a una variable binaria y el segundo convierte el separador decimal *coma* a *punto.*

3.2 Partición Train-Test

La partición de los datos en muestra de train y test (70-30) que va a permanecer a lo largo del proceso se realiza en el método *data_split*. Si una de ambas clases del target tiene una frecuencia relativa menor al 35%, la partición se realiza de forma estratificada para evitar que las muestras de train y test sean muy dispares en proporción.

3.3 Limpieza

Una vez realizada la partición, es necesario hacer una limpieza de variables que sabemos con certeza que no tienen poder predictivo. Es cierto sentido, es una especie de selección de variables previa que no requiere criterio experto. De este modo, también se reduce la dimensionalidad del dataset y el coste computacional del proceso. Se realiza una doble limpieza: primero se eliminan las variables con el mismo valor para todos los registros en la muestra de train (método *constant_features*) y posteriomente se eliminan las variables categóricas con demasiadas categorías en la muestra de train (método *categories_checking*). El objetivo de este segundo punto es deshacerse de variables que prácticamente actúan como identificadores de los registros (ejemplo: 1000 filas, 800 categorías). El umbral impuesto es un número de categorías superior o igual a la mitad del número de registros. Ambos métodos se ajustan en train y se mapean a test.

3.4 Imputación de valores nulos

Se han programado 3 métodos distintos de imputación de missing values. Datawig, Imputación Simple y eliminación de registros. Después de realizar pruebas con los 3, ninguno mejoraba los resultados en predicción respecto a los demás, por lo que se ha optado por imputación simple debido a su bajo coste computacional. Los 3 modelos están programados en la clase *Missings* del fichero *missings_module.py*. A continuación, una breve descripción de los tres métodos implementados:

- Datawig: uso de redes neuronales para imputar variables categóricas y numéricas.
- Eliminación de registros: se eliminan los missing values.
- Imputación simple: Imputación de variables numéricas por mediana y variables categóricas por moda.

En el proceso se imputan los datasets de test y se estima con los valores/modelos obtenidos en train. Un ejemplo sería imputar los missing values de la variable numérica ficticia 'RZ' en el set de test con la mediana de la variable 'RZ' de los datos de train.

3.5 Encoding de las variables categóricas

Los modelos de la librería *scikit-learn* no aceptan strings como clases de las variables categóricas, por lo que es necesario realizar algún tipo de encoding para que dichas variables puedan entrar en los modelos. Existen varias alternativas para esta tarea: One-Hot-Encoder, Weight of Evidence, Decision Trees… La idea inicial era crear un hiperparámetro del tipo de encoding con varias alternativas y que se eligiera la óptima, pero debido a las limitaciones de tiempo se va a elegir una sola variante y se va a aplicar por defecto. Dicho subapartado está desarrollado en el método *categoricalEncoding*. El encoding se va realizar usando el algoritmo de CatBoost ya implementado en la librería category_encoders. Dicho método es considerado uno de los más potentes para encoding de categóricas; mezcla ordered gradient boosting, leave-one-out encoding, etiquetas temporales y dropout. El encoding se realiza sobre la muestra de train y se mapea a la de test.

4. Feature Selection

El apartado de selección de variables se incluye dentro del pipeline a optimizar, por lo que el número de variables seleccionadas es el primer hiperparámetro del flujo. La metodología que se usa en este apartado es la siguiente:

- Se entrenan un Random Forest, un Adaboost, un Gradient Boosting y un Extreme Gradiente Boosting con toda la muestra de train
- Se crea un dataset en el que las filas son las variables, las columnas son los 4 modelos y los valores de cada registro son el feature importance asignado por cada modelo a cada variable.
- Se crea una quinta variable que es la media de importancia de cada variable asignada por los 4 modelos.
- Se ordena el dataset descendientemente por esa nueva columna (variables más importantes en los primeros registros)
- Se pone la nueva variable como una suma acumulativa de importancias (siendo 1 siempre el valor de último registro)

Una vez generado dicho dataframe, las variables se van a seleccionar mediante el hiperparámetro *threshold*. Por ejemplo, si su valor es 0.2, significa que se van a desechar el 20% de variables menos importantes.

5. Modelización

El apartado de modelización se va a componer de dos procesos paralelos: Stacking y Extreme Gradient Boosting.

Lo primero que se realiza es una partición de la muestra de train y una de validación y otra nueva muestra de train. A continuación, se genera el modelo Stacking Classifier según los base models y el meta model elegido. Para ello, se crea una clase *Stacking_Pipeline* en la que se genera el modelo y el diccionario de hiperparámetros correspondiente según los modelos que se usan. A su vez, se incluye el hiperparámetro threshold de feature selection en el diccionario que también va a ser optimizado en el pipeline. La clase *Stacking_Pipeline* prepara el modelo y los hiperparámetros para poder ser entrenados en formato pipeline. Los modelos seleccionados por defecto son un Random Forest, un Gradiente Boosting y un Support Vector Classifier como base models y un Gradient Boosting como meta-model. Paralelamente, se entrena un Extreme Gradient Boosting sobre la segunda partición de train sin haber realizado feature selection y con los valores de los hiperparámetros que presenta el modelo por defecto.

Una vez se entrenan ambos modelos, se realiza una predicción sobre el set de validación y se elige el que obtenga mayor AUC. El objetivo de entrenar el XGBoost paralelamente al Stacking es evitar el sobreentrenamiento. A veces, el Stacking es tan potente que llega a obtener un 1 de AUC sobre train pero generaliza peor que otros clasificadores.

6. Optimización del proceso

Para elegir la mejor combinación de hiperparámetros se debe realizar una optimización de todo el flujo; esto incluye la selección de variables y el Stacking. Para ello se va a usar Optimización Bayesiana junto con cross-validation. Dicha técnica de optimización encuentra combinaciones de hiperparámetros que tienden a conseguir mejor predicción en validación cruzada y, genera y evalúa puntos próximos en las siguientes iteraciones. Una vez llega número máximo de iteraciones, selecciona la mejor combinación. Este método de optimización es más práctico que Grid Search o Random Search en funciones de pérdida muy caras ya que itera eficientemente y resulta menos costoso que el primero, lo cual es importante debido a las limitaciones de tiempo. Su punto negativo es el más que probable sobre-entrenamiento de la combinación ganadora. El proceso se realiza en la clase *AutoML*; solo se introduce el pipeline generado en la clase *BayesSearchCV* de la librería *scikit-optimize* y se eligen los parámetros: número de iteraciones, número de particiones para cross validation, número de puntos en cada iteración, lista de hiperparámetros, etc…