Trabajo de investigación

AutoML: La herramienta TPOT

Grupo: EuroML

Carmona Lupiáñez, Adrián
Sánchez Herrera, Ignacio

April 14, 2024

Resumen

En los últimos años, el crecimiento del interés por el mundo del aprendizaje automático, la ciencia de datos y la inteligencia artificial a sufrido un crecimiento exponencial. Esto hace que cada vez sean más necesarias las herramientas que permitan el uso de este tipo de técnicas sin un conocimiento profundo del campo.

Dentro de este campo han surgido múltiples herramientas como Auto-WEKA o Auto-Sklearn. En este trabajo nos centramos en el análisis de la herramienta TPOT, la cual propone un enfoque diferente para la creación automática de flujos de Machine Learning. Al contrario que sus competidores, que se basan en la optimización Bayesiana, TPOT se basa en la programación genética: una clase de algoritmos e volutivos que simulan la evolución natural para resolver problemas de optimización. Aplica este enfoque al problema de encontrar el mejor pipeline de aprendizaje automático, considerando cada pipeline como un "individuo" dentro de una población.

Veremos que es una herramienta que obtiene muy buenos resultados y que facilita el acceso al uso de inteligencia artificial sin tener unos conocimientos avanzados de esta.

Índice de contenidos

Índice de figuras

Índice de tablas

1 Introducción

AutoML, o Aprendizaje Automático Automatizado, es un conjunto de técnicas y herramientas diseñadas para automatizar el proceso de construcción, entrenamiento y despliegue de modelos de aprendizaje automático. El objetivo principal del AutoML es hacer que el aprendizaje automático sea más accesible para personas sin experiencia en ciencia de datos o aprendizaje automático, al tiempo que acelera y mejora el proceso para expertos en el campo.

El proceso tradicional de construir modelos de aprendizaje automático implica varias etapas, como la selección y preparación de datos, la ingeniería de características, la selección de algoritmos, la optimización de hiperparámetros y la evaluación del rendimiento del modelo. Estas tareas pueden requerir un conocimiento profundo del dominio y experiencia en ciencia de datos.

El AutoML automatiza gran parte de este proceso, utilizando algoritmos y técnicas para realizar tareas como:

- Preprocesamiento de datos: Incluye la limpieza de datos, la imputación de valores perdidos, la normalización y la codificación de variables categóricas.
- Ingeniería de características: Identifica y crea nuevas características a partir de las existentes para mejorar el rendimiento del modelo.
- Selección de algoritmos: Prueba automáticamente una variedad de algoritmos de aprendizaje automático para encontrar el más adecuado para un conjunto de datos específico.
- Optimización de hiperparámetros: Ajusta automáticamente los hiperparámetros de los algoritmos para maximizar el rendimiento del modelo.
- Ensamblaje de modelos: Combina múltiples modelos para mejorar la precisión general.
- Despliegue del modelo: Facilita la implementación del modelo en producción.

El AutoML puede ofrecer diferentes niveles de automatización, desde soluciones totalmente automatizadas donde el usuario solo proporciona los datos y recibe un modelo listo para usar, hasta herramientas que permiten un control más granular sobre el proceso. Sin embargo, la mayoría de herramientas solo se centran en algunos aspectos de los comentados.

En este trabajo nos centraremos en el análisis de la herramienta de TPOT [6] para la automatización de *pipelines* de *Machine Learning*. TPOT es el acrónimo de *Tree-Based Pipeline Optimization Tool*. Se trata de una herramienta de optimización de pipelines de AutoML que se basa en la programación genética para generar, sin ninguna interacción humana, una serie de transformaciones de datos y ajustes de modelo con el propósito de maximizar la precisión (*accuracy*) de una tarea de clasificación o de regresión.

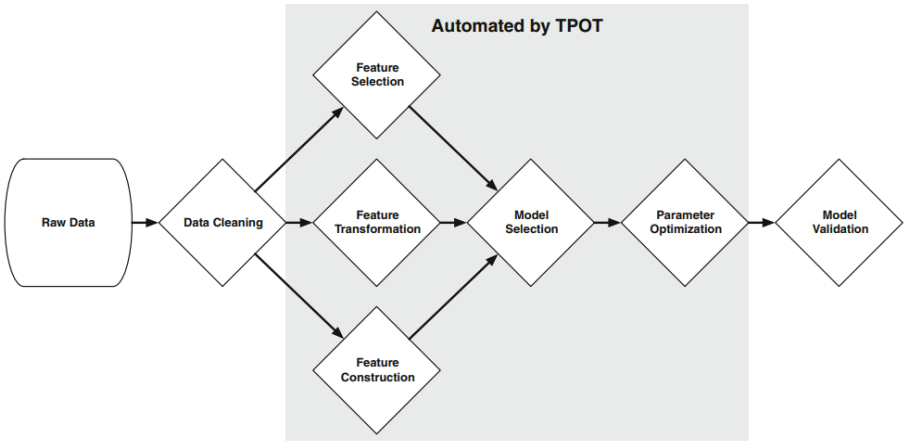

Figure 1: Alcance de TPOT

TPOT automatiza las tareas relacionadas con las distintas características del dataset, la selección del modelo y su ajuste. En la Figura 1 se pueden ver las etapas del proceso de Machine Learning y en cuales interviene TPOT.

En las siguientes secciones veremos la fundamentación teórica bajo esta herramienta (Sec. 2), explicaremos en detalle la herramienta TPOT (Sec. 3), repasaremos algunas de las aproximaciones a AutoML del estado del arte (Sec. 4), presentaremos un caso de estudo de TPOT (Sec. 5) y finalmente hablaremos sobre las lecciones aprendidas y conclusiones extraídas (Sec. 6 y 7)

2 Fundamentación teórica

En esta sección describiremos los fundamentos en los que se basa TPOT. Concretamente nos centraremos en la programación genética, ya que es la base de esta herramienta.

2.1 Programación Genética

La Programación Genética [1] (GP por sus siglas en ingles) es un tipo de algoritmo evolutivo donde los individuos de la población son programas. En este tipo de algoritmos, las poblaciones de individuos (programas) se transforman de forma iterativa en otras poblaciones de programas.

A rasgos generales, los pasos seguidos por los algoritmos de programación genética se pueden ver en el Algoritmo 1.

Algorithm 1 Resumen de algoritmo GP

1: Crear una población inicial aleatoria de programas a partir de las primitivas disponibles.
2: **repeat**
3: Ejecutar cada programa y calcular su idoneidad.
4: Seleccionar uno o dos programas de la población con una probabilidad basada en la idoneidad para participar en las operaciones genéticas.
5: Crear nuevos individuos aplicando operaciones genéticas.
6: **until** Se encuentra una solución aceptable

En las siguientes secciones veremos como se representan los problemas de GP, así como los elementos principales de este tipo de algoritmos.

2.1.1 Representación

En GP, los programas se representan mediantes *árboles de sintaxis*. La Figura 2 representa el árbol de sintaxis para el programa `max(x*x, x+3y)`. En este tipo de árboles, las variables y las constantes son las hojas del arbol, mientras que los operadores aritméticos son nodos internos.

2.1.2 Inicialización de la población

De manera análoga al resto de algoritmos evolutivos, en GP los individuos que forman la población inicial se generan de forma aleatoria. Existen multitud de métodos para generar estas poblaciones en GP. Un ejemplo de ellos es el método *Full*. Este método tiene pre-establecida una profundidad máxima del árbol. Los nodos son escogidos de manera aleatoria del conjunto de funciones hasta que se alcanza esta profundidad máxima, y a partir de aquí solo se pueden escoger nodos terminales. En la Figura 3 se puede ver un ejemplo de generación de un árbol usando este método.

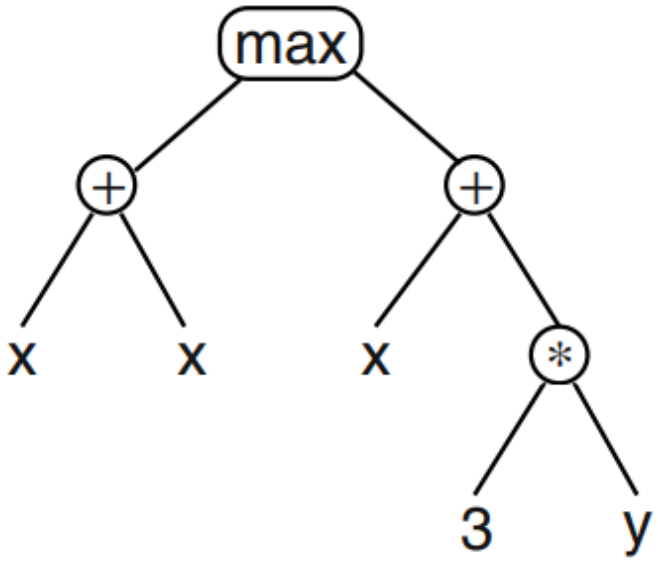

Figure 2: Árbol de sintaxis que representa `max(x*x, x+3y)`

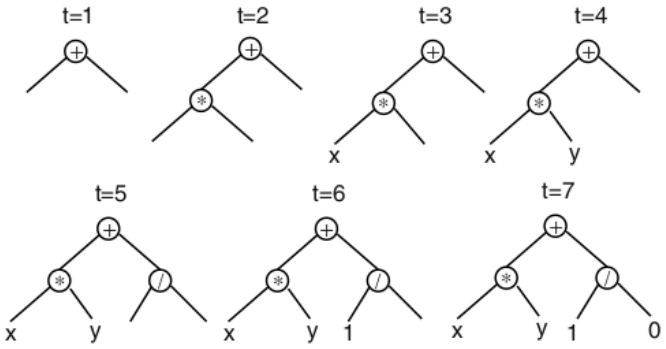

Figure 3: Creación de un árbol utilizando el método *Full* con una profundidad máxima de 2

2.1.3 Selección

En los algoritmos evolutivos es necesario escoger los individuos de la población sobre los que se aplicarán los operadores genéticos (operadores de cruce, mutación, ...). Existen numerosas alternativas para escoger a estos individuos, pero todas se basan en la idoneidad de la solución de este, es decir, en lo bueno que sea el resultado obtenido.

Normalmente se escogen los mejores individuos y se introduce algún grado de aleatoriedad para que la solución no se estanque en óptimos locales.

2.1.4 Operadores de cruce y mutación

Tras la selección de los individuos de la población, se aplican diferentes operadores sobre estos. Los principales son el operador de cruce y la mutación.

El operador de cruce implica que dos elementos de la población se cruzan creando uno nuevo. En GP el principal operador de cruce es el *subtree crossover*. Dados dos padres, *subtree crossover* selecciona un punto de cruce en el árbol de sintaxis de cada uno de los padres de manera aleatoria. Luego genera el nuevo individuo reemplazando el sub-árbol que queda por debajo del punto de cruce de uno de los padres, por el sub-árbol que queda por debajo del punto de cruce del otro, como se puede ver en la Figura 4

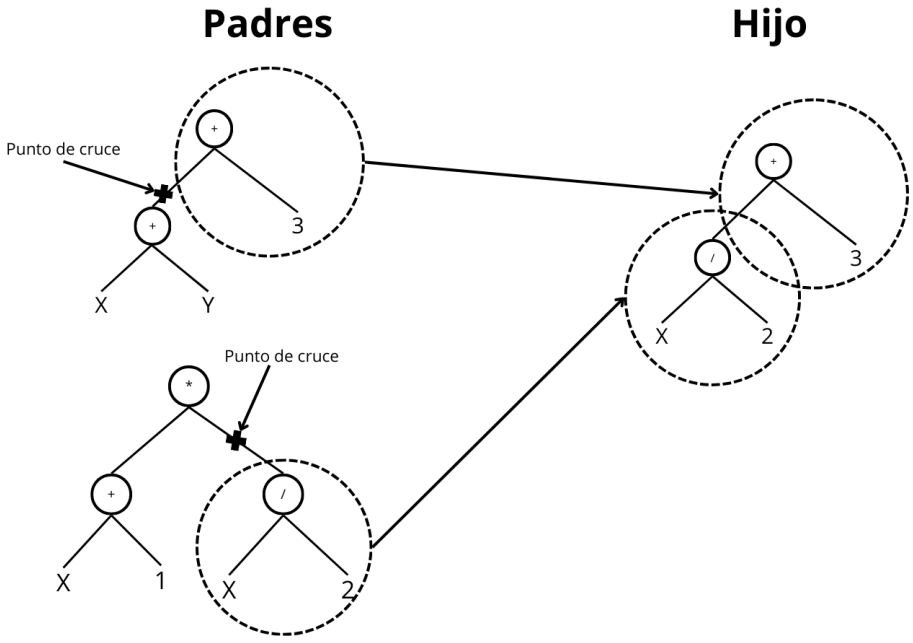

Figure 4: Ejemplo de *subtree crossover*

Por otra parte, la mutación consiste en una modificación aleatoria de un individuo de la población. El operador de mutación más usado en GP es *subtree mutation*. Este tipo de operador selecciona un punto aleatorio en el árbol y sustituye el sub-árbol que queda por debajo por un árbol aleatorio.

3 La herramienta TPOT

TPOT es una herramienta de AutoML, la cual aplica GP para la creación de *pipelines* de Machine Learning. Se trata de un paquete software en python construido encima de Scikit Learn que asegura su integracion completa con este ecosistema(Scikit-Learn, XGBoost, etc.).

Cada *pipeline* es representado por un árbol de sintáxis, donde cada nodo representa un operador de Scikit-learn que puede ser:

1. Un modelo de aprendizaje supervisado.
2. Una transformación sobre características.
3. Una combinación de dos datasets.

Es decir, las primitivas de los problemas de programación genética se igualan aquí a las distintas funciones de Scikit-Learn que corresponden con modelos, transformaciones de características y operadores de unión.

3.1 Construcción del árbol de operadores

El proceso de construcción del árbol comienza desde las hojas que albergan los datos, hacia el nodo raíz que alberga un modelo, como se puede ver en la Figura 5. Cada árbol puede comenzar con al menos una copia de los datos, teniendo que hacer uso de un operador de combinación mas adelante en el pipeline si fuera necesario. Durante el procesamiento, las copias de los datos se pueden encontrar con distintos tipos de operadores que pueden añadir mas información al conjunto de datos.

Los operadores del tipo transformación de características produce un nuevo conjunto de datos transformado. Sin embargo, si el operador es un modelo clasificador o regresor, albergará sus resultados en una nueva columna del conjunto de datos que se irá reescribiendo con los resultados de los sucesivos operadores de modelo. Esta columna también se incluirá en las entradas de los operadores posteriores y será la que albergue los resultados del operador final que se usan para calcular la métrica de evaluación.

3.2 Optimización

La tarea de optimización llevada a cabo con programación genética comienza generando una población de arboles representando *pipelines* arbitrarios.

El primer paso es evaluar con la métrica deseada los pipelines generados y escoger aquellos individuos que son mas prometedores para aplicar sobre ellos los operadores genéticos. En este caso, los individuos mas prometedores son los pipelines que mejor valor de métrica escogida muestran. Estos se escogen en función de algún criterio. Por ejemplo: *El percentil 90 o superior.*

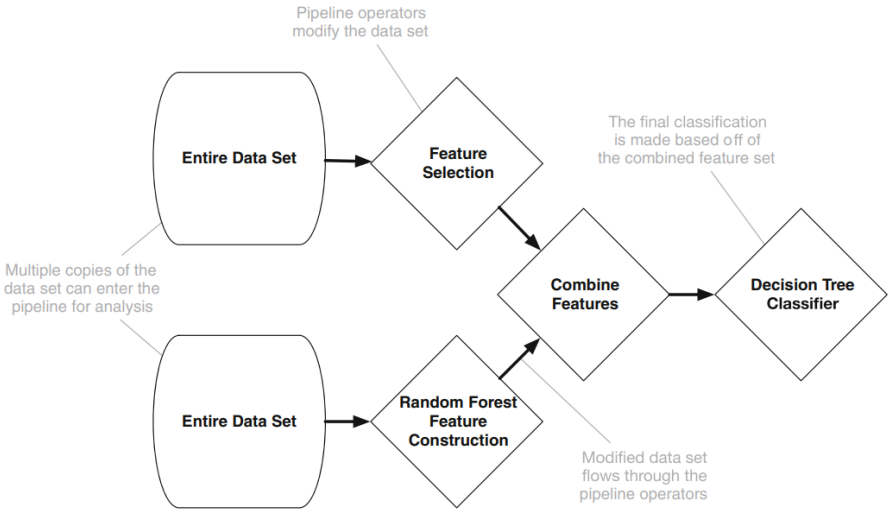

Figure 5: Construcción de árboles.

Tras esto, se les aplican las distintos operadores de cruce y mutación a los individuos escogidos. Estos operadores pueden ser de tipo *subtree mutation*, descrito en los fundamentos teóricos, o bien de tipo *shrink tree* dónde se eliminan secuencias de operadores desde un nodo aleatorio. Con la aplicación de estas Como resultado de esto se obtienen los individuos mutados que conformarán una determinada proporción de la población de la segunda iteración.

El proceso acaba tras superar un umbral de métrica o de número de ejecuciones.

.

4 Estado del arte

En los últimos años han surgido diferentes herramientas para la automatización del proceso de Machine Learning. Son herramientas con la misma finalidad pero que presentan diferencias entre ellas, ya sea en el método de optimización del *pipeline* subyacente o en las fases del *pipeline* que automatiza.

En esta sección haremos un repaso de las principales herramientas de AutoML del estado del arte.

4.1 Auto-WEKA

Auto-WEKA [7] es una herramienta basada en WEKA [8] la cual permite automatizar el proceso de selección del mejor modelo y de sus hiperparámetros. Para ello establece un conjunto de algoritmos y sobre estos define el espacio de búsqueda de los hiperparámetros. Para cada uno de estos algoritmos hace una optimización de sus hiperparámetros haciendo uso de Optimización Bayesiana [2]. Estos algoritmos son probados sobre el dataset, quedandose finalemente con el que mejor resultado obtenga.

4.2 Auto-Sklearn

Auto-Sklearn [4] es una herramienta basada en Auto-WEKA. Emplea las mismas técnicas de optimización Bayensiana que esta última, sin embargo, incrementa su eficiencia y robustez gracias al *meta-learning* y a la construcción automática de *ensembles*.

4.2.1 *Meta-learning* en Auto-sklearn

En AutoML, el *meta-learning* se refiere al uso de algoritmos de aprendizaje automático para aprender sobre el comportamiento y el rendimiento de diferentes modelos de aprendizaje automático en conjuntos de datos específicos. En lugar de buscar directamente el mejor modelo y configuración para un problema dado, el enfoque de meta-learning busca aprender heurísticas, patrones o reglas que guíen la selección y configuración de modelos para problemas futuros. Para ello se basa en soluciones pasadas a problemas, extrayendo meta-características de los datasets de estos problemas y los modelos empleados para resolverlos.

Auto-sklearn hace uso del meta-learning para hacer un inicio en caliente del problema lo que hace que el proceso de optimización sea mucho más rápido.

4.2.2 Construcción automática de *ensembles*

Durante el proceso de optimización Bayesiana, se entrenan muchos modelos que finalmente no son usados, por lo que, la propuesta de Auto-Sklearn consiste en almacenarlos y hacer un proceso de post-procesado para construir un *ensemble* a partir de ellos.

Esta construcción automática de *ensembles* evita comprometerse con un único ajuste de hiperparámetros y, por tanto, es más robusta (y menos propensa al sobreajuste) que el uso de la estimación puntual que arroja la optimización estándar de hiperparámetros.

4.3 Auto-Net

Las soluciones de AutoML explicadas en las Secciones 4.1 y 4.2 no incluyen las redes neuronales en el conjunto de algoritmos de aprendizaje automático a seleccionar. Es por esto que Auto-Net [5] surge para solventar este problema, siendo una herramienta de AutoML dedicada exclusivamente a redes neuronales.

En una primera versión (Auto-Net 1.0), emplea las mismas técnicas de optimización Bayesiana para la automatización del proceso de creación de redes neuronales limitándose a redes totalmente conectadas. Esto lo hace empleando como base la librearía de python de aprendizaje profundo Lasagne.

La segunda versión (Auto-Net 2.0) introduce tres cambios fundamentales:

1. Usa PyTorch en vez de Lasagne [3].
2. Usa un espacio de configuración más grande incluyendo las últimas técnicas de aprendizaje profundo, arquitecturas modernas (como ResNet) y una representación más compacta del espacio de búsqueda.
3. Modifica el proceso de optimización y ya no usa el mismo que Auto-Sklearn o Auto-WEKA.

Con estos cambios introducidos, produce una mejora significativa sobre su predecesor.

5 Caso de estudio

En esta sección se han hecho diferentes análisis con la herramienta TPOT. Principalmente se han comparado los resultados obtenidos por TPOT de manera automática, con los resultados obtenidos por nosotros realizando a mano los procesos de selección de características, reducción de dimensionalidad, tuneo de hiperparametros, etc.

Para estas pruebas se ha aplicado TPOT sobre problemas de clasificación y selección, como veremos en las Secciones 5.1 y 5.2 respectivamente. Las pruebas realizadas se pueden ver en el archivo `html` adjunto, si bien los resultados de no tienen por qué coincidir con los aquí expuestos ya que diferentes ejecuciones pueden dar diferentes resultados, debido a que no es un método determinista.

5.1 Clasificación

El primer problema consiste en un problema de clasificación ordinal, es decir, las etiquetas de salida tienen un orden. El problema consiste en asignar un rating a los distintos clientes de un banco, el cual va de 1 (el mejor) a 4 (el peor).

Las medidas de error empleadas para la comparación de los resultados han sido: *Mean Zero-One Error* (MZE), que mide el ratio de fallos, y *Mean Absolute Error* (MAE) que mide el error absoluto medio en distancia a la clase correcta. Este último es útil en problemas de clasificación ordinal ya que no es lo mismo clasificar un elemento en clase 1 cuando la real es 4, que clasificarlo en la 3, puesto que la 3 está más cerca de la 4 en clasificiación ordinal.

En este caso de estudio hemos empleado diferentes algoritmos, aplicando y sin aplicar el enfoque multiple ordinal (específico para problemas de clasificación ordinal), y los hemos comparado con el resultado obtenido por TPOT. Estos resultados se pueden ver en la Tabla 1

Model	**MZE**	**MAE**
RandomForest	0.330502	0.363707
SVC	0.558301	0.795367
MLP	0.566795	0.736680
MMOC - RandomForest	0.314286	0.335135
MMOC - SVC	0.579923	0.641699
MMOC - MLP	0.547490	0.681853
XGBClassifier	0.318919	0.349035
TPOT	**0.271604**	**0.283950**

Table 1: **Comparativa TPOT en clasificación**: Los modelos precedidos por MMOC usan el enfoque Múltiple Ordinal.

En este caso el mejor resultado de TPOT ha sido usando un XGBClassifier sobre el que ha hecho una optimización de hiperparámetros que ha dado un mejor resultado que nuestra versión.

Como se puede ver, se logra obtener un muy buen resultado aplicando TPOT sobre este problema, siendo, de hecho, el mejor resultado obtenido, con un MZE inferior al resto, lo cual nos indica que el porcentaje de aciertos ha sido mayor y con un MAE también inferior, lo que indica que los errores cometidos han estado más cercanos a la clase real.

5.2 Regresión

En este caso hemos comparado el resultado de TPOT con diferentes métodos de regresión para un problema de predicción de precios de la vivienda. La metrica de error utilizada para las comparativas ha sido el *Mean Squared Error* (MSE).

Los resultados obtenidos sobre el conjunto de test se pueden ver en la Tabla 2.

Model	**MSE**
CatBoost	1.532×10^{12}
LightGBM	1.602×10^{12}
Ridge Regresion	1.525×10^{12}
Support Vector Machine	4.462×10^{12}
XGBoost	1.655×10^{12}
TPOT	9.843×10^{11}

Table 2: Comparativa TPOT en regresión.

En este caso, el mejor *pipeline* obtenido por TPOT es el siguiente:

1. Aplica `RidgeCV` sobre el dataset de entrada.
2. Aplica de nuevo `RidgeCV` sobre el dataset obtenido en el paso anterior.
3. Aplica `RobustScaler` sobre el dataset obtenido en el paso anterior.
4. Aplica `ExtraTreesRegressor` sobre el dataset obtenido en el paso anterior.

En la salida obtenida por TPOT se puede ver la expresión del *pipeline* en python: `ExtraTreesRegressor(RobustScaler(RidgeCV(RidgeCV(input_matrix))), bootstrap=True, max_features=0.9, min_samples_leaf=4, min_samples_split=7, n_estimators=100)`

Se puede ver que no es un *pipeline* trivial, que además, obtiene mejores resultados que cualquier modelo aplicado por nosotros, como se muestra en la Tabla 2.

6 Lecciones aprendidas y trabajo futuro

Usar la programación genética para abordar el problema de AutoML es posible. El resultado de la aplicación son pipelines de transformación de datos y modelado que en ocasiones son imposibles de encontrar para un humano y que mejoran los resultados de los casos de uso como hemos podido comprobar en la Sección 5. Esto nos demuestra que es posible hacer que el uso de modelos de IA llegue a personas que no tienen un gran conocimiento de esta.

La aplicación de esta técnica se puede llevar a cabo con facilidad gracias a los paquetes de Python dispobibles. Sin embargo, la interpretabilidad del modelo obtenido no esta garantizada. Con la transformación de características desde el espacio inicial del problema hacia espacios con un menor número de características, se obtienen otros nuevos que pueden llegar a ser imposibles de interpretar.

En este aspecto de la interpretabilidad es posible que surjan nuevas lineas de trabajo en el futuro que permitan el desarrollo de AutoML interpretable. Esto sería un gran avance viendo el marco de regulación que se está implantando en una gran cantidad de países.

7 Conclusiones

TPOT es una técnica y una herramienta eficaz para la aplicación de programación genética en la optimización de pipelines de *machine learning*. Es capaz de mejora los resultados de los modelados humanos como hemos visto en la Sección 5. Sin embargo, las técnicas empleadas pueden hacer que se pierda la interpretabilidad del modelo, al realizar modificaciones o combinaciones de los datos.

En el contexto de la regulación de la inteligencia artificial, esta técnica y todas las que hacen uso de transformaciones de características combinando variables no van a poder aplicarse a según que problemas, volviendo AutoML un campo muy útil y prometedor en entornos académicos pero comprometiendo su potencial empresarial.

Al margen de esto, podemos decir que TPOT es una herramienta que compite en el estado del arte del AutoML, con una propuesta diferente al resto en cuanto al proceso de generación de *pipelines* se refiere, la cual hemos visto que da muy buenos resultados. A pesar de ello, deja sin explotar algunas fases del pipeline, principalmente el preprocesado de los datos; lo que podría ser muy interesante introducir de cara al futuro.

References

[1] Wolfgang Banzhaf et al. *Genetic programming: an introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Inc., 1998.

[2] Eric Brochu, Vlad M. Cora, and Nando de Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. 2010.

[3] Sander Dieleman et al. *Lasagne: First release.* Aug. 2015.

[4] Matthias Feurer et al. "Efficient and Robust Automated Machine Learning". In: *Advances in Neural Information Processing Systems*. 2015.

[5] Hector Mendoza et al. "Towards Automatically-Tuned Deep Neural Networks". In: *Automated Machine Learning: Methods, Systems, Challenges*. 2019, pp. 135–149.

[6] Randal S Olson and Jason H Moore. "TPOT: A tree-based pipeline optimization tool for automating machine learning". In: *Workshop on automatic machine learning*. PMLR. 2016, pp. 66–74.

[7] Chris Thornton et al. "Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms". In: *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2013, pp. 847–855.

[8] Ian H Witten et al. "Weka: Practical machine learning tools and techniques with Java implementations". In: (1999).