Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Identificación y análisis de las principales tecnologías de AutoML

Identification and analysis of the main AutoML technologies

Chesen Castilla Gil

La Laguna, 7 de septiembre de 2021

D. **Dagoberto Castellanos Nieves**, con N.I.F. 79.234.766-L profesor Titular de Universidad adscrito al Departamento de Ingeniería Informática y de Sistemas Departamento de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"*Identificación y análisis de las principales tecnologías de AutoML*"

ha sido realizada bajo su dirección por D. **Chesen Castilla Gil**, con N.I.F. 51.148.471-J

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de septiembre de 2021

Agradecimientos

Quería agradecer a todas las personas que me han ayudado en este periodo de mi vida en el cual no he estado en mis mejores momentos. Primero, a mi familia por apoyarme y acompañarme en todo momento, y especialmente en los momentos malos que he tenido en este ultimo año. Segundo, a mi tutor Dagoberto Castellano Nieves, no solo por su tutorización y apoyo en la realización de este TFG, también por estar siempre que necesitaba ayuda a lo largo de la realización del grado. Y por último a mis amigos por estar ahí siempre, tanto en las buenas como en las malas.

Licencia

Resumen

El Auto Aprendizaje Automático (AutoML) es un campo en crecimiento del aprendizaje automático y del aprendizaje profundo el cual esta consiguiendo la democratización del sector de los científicos de datos. Con diferentes técnicas y herramientas se desarrollan con poca experiencia modelos competentes en un mercado que se diferencia entre herramientas de código abierto y herramientas comerciales desarrolladas por las empresas más grandes de la informática. Por lo tanto, en este proyecto se tratara de investigar este campo del AutoML con sus principales técnicas y analizar el mercado de herramientas que se le ofrecen al usuario para realizar una experimentación con ellas.

Palabras clave: Aprendizaje Automático, Automatizador del aprendizaje automático, Aprendizaje profundo, Redes neuronales, Hiperparámetros, Software de código abierto

Abstract

Automatic Machine Learning (AutoML) is a growing field of machine learning and deep learning which is democratizing the data scientist industry. With different techniques and tools, competent models are developed with little experience in a market that differs between open source tools and commercial tools developed by the largest IT companies. Therefore, this project will try to investigate this field of AutoML with its main techniques and analyze the market of tools that it offers the user to experiment with them.

Keywords: Machine Learning, Auto Machine Learning, Deep Learing, Neural networks, Hyperparameters, Open-source software

Índice general

1. **Introducción** **1**
   - 1.1. Antecedentes y estado actual del tema . . . 1
   - 1.2. Objetivos . . . 2
2. **Auto Machine Learning** **3**
   - 2.1. Ciclo de Vida . . . 3
   - 2.2. Tipos de AutoML . . . 4
   - 2.3. Ventajas y Desventajas del AutoML . . . 5
   - 2.4. Ecosistema del AutoML . . . 6
     - 2.4.1. Plataformas Open Source . . . 7
     - 2.4.2. Plataformas Comerciales . . . 8
3. **Principales técnicas y algoritmos del AutoML** **11**
   - 3.1. Ingeniería de Características . . . 12
   - 3.2. Selección del Modelo . . . 12
   - 3.3. Optimización del Modelo . . . 12
     - 3.3.1. Búsqueda Arquitectura Neuronal . . . 13
     - 3.3.2. Optimización Hiperparámetros . . . 14
4. **Experimento con tecnologías seleccionadas** **17**
   - 4.1. Selección de Tecnologías . . . 17
   - 4.2. Experimentación . . . 19
     - 4.2.1. Preparación . . . 19
     - 4.2.2. Experimentación preliminar . . . 22
   - 4.3. Experimentación final . . . 32
   - 4.4. Análisis de los resultados . . . 38
     - 4.4.1. Comparación de resultados obtenidos . . . 38
     - 4.4.2. Opinión de las herramientas . . . 39
5. **Conclusiones y líneas futuras** **41**
6. **Summary and Conclusions** **43**

7. **Presupuesto** 45
7.1. Presupuesto de recursos humanos 45
7.2. Presupuesto material 45
7.3. Presupuesto final 46
A. **Cuadernos Jupyter de la Experimentación** 47
A.1. Cuaderno Jupyter con el Código de AutoKeras 47
A.2. Cuaderno Jupyter con el Código de AutoSklearn 49
A.3. Cuaderno Jupyter con el Código de TPOT 50
B. **Grafos de arquitectura neuronal** 53
B.1. Grafo de arquitectura neuronal de la experimentación preliminar . . 53
B.2. Grafo de arquitectura neuronal de la experimentación final 54

Índice de Figuras

2.1. Esquema general del AutoML 3
2.2. Ciclo de vida del AutoML 4
2.3. Tipos de AutoML 5

3.1. Proceso del AutoML detallado 11
3.2. Taxonomía de las técnicas del NAS 14
3.3. Taxonomía de las técnicas del HPO 16

4.1. Software de Weka 20
4.2. Software de AutoWeka 20
4.3. Logo Google Colab 21
4.4. Formula de la métrica Accuracy 22
4.5. Matriz de Confusión 23
4.6. Selección de Dataset 24
4.7. Gráfica de clases 24
4.8. Opciones de hiperparámetros 25
4.9. Resultados AutoWeka 26
4.10 Gráfica de muestra del Dataset MNIST 27
4.11 Resultados AutoKeras 28
4.12 Resumen mejor modelo de Autokeras 29
4.13 Ejemplo de Load Digits 30
4.14 Resultados AutoSklearn 31
4.15 Resultados TPOT 32
4.16 Ejemplo de especies de IRIS 32
4.17 Resultados AutoWeka 33
4.18 Resultados AutoKeras 34
4.19 Matriz de confusión del modelo de AutoKeras 34
4.20 Resultados AutoSklearn 35
4.21 Matriz de Confusión del Modelo de AutoSklearn 36
4.22 Resultados TPOT 37
4.23 Matriz de Confusión del Modelo de TPOT 38
4.24 Gráfica comparativa del Tiempo de Ejecución 39

B.1. Grafo Arquitectura Neuronal del mejor modelo de AutoKeras 53
B.2. Grafo Arquitectura Neuronal del mejor modelo de AutoKeras 54

Índice de Tablas

2.1. Tabla de Herramientas Open Source 7
2.2. Tabla de Herramientas Comerciales 9

4.1. Tabla de Comparación de Resultados del experimento Preliminar . . 38
4.2. Tabla de Comparación de Resultados del experimento sobre IRIS . . 39
4.3. Tabla de comparación de funcionamiento de las Herramientas 40

7.1. Presupuesto Personal 45
7.2. Presupuesto de los materiales necesarios 46
7.3. Presupuesto Final 46

Capítulo 1

Introducción

El tema general del proyecto es la inteligencia artificial, en concreto trataremos el tópico del aprendizaje automático (machine learning) y el aprendizaje profundo (deep learning), para ser más concretos nos centraremos en el tópico del Auto machine learning o AutoML que es uno de los campos más innovadores dentro del sector.

El Auto Machine Learning o AutoML es una forma de automatizar el ciclo de un extremo a otro del Machine Learning. Los científicos de datos suelen ser responsables de crear modelos de aprendizaje automático y de todas las tareas complejas que conllevan ese desarrollo: preprocesamiento de datos, ingeniería de características, selección de modelos, optimización de hiperparámetros y postprocesamiento de modelos. Los procesos de AutoML completan estos pasos (o al menos algunos de ellos) automáticamente, de modo que las personas sin experiencia en ciencia de datos puedan crear modelos de ML exitosos. [8, 7].

En este proyecto se realiza un estudio de las diferentes tecnologías y herramientas que nos ofrece este sector como usuarios, contemplaremos varias posibilidades y analizaremos los resultados obtenidos y se dará una visión general del sector, el cual está en auge y que tiene un gran potencial, con la elaboración de un informe técnico detallado podremos ver rápidamente cuáles herramientas son más útiles en la elaboración de un proyecto de aprendizaje automático.

1.1. Antecedentes y estado actual del tema

El aprendizaje automático (ML) tiene como objetivo desarrollar un sistema o un algoritmo capaz de aprender de los datos sin intervención humana. El objetivo de un científico de datos no es instruir al algoritmo sobre cómo aprender, sino más bien proporcionar un conjunto de datos preparado y de tamaño adecuado para el algoritmo y especificar brevemente la relaciones entre las variables del conjunto de datos. El ML se ha adoptado en casi todas las industrias en la última década, la razón principal son los avances en hardware, además, el aprendizaje automático se ha vuelto mucho más accesible estos últimos años para los menos expertos. El ML es un tema complejo, generalmente reservado para científicos de datos, después de todo, el éxito de los proyectos con la ayuda del ML no se mide normalmente por el nivel de sofisticación de los algoritmos utilizados, sino más bien por el éxito

obtenido.

Dado que el proceso de desarrollo de un modelo de ML es algo extenso y repetitivo, se desarrollo el Auto Machine Learning o AutoML, una herramienta que nació simplemente para ahorrar este ciclo repetitivo de selección y entrenamiento de los modelos.

En lo que respecta a las desventajas, como todo lo demás, el AutoML no es la solución definitiva. Sin embargo, los métodos automatizados para la selección de modelos y la optimización de hiperparámetros tienen la promesa de permitir que los no expertos y los científicos de datos novatos sean capaces de probar e implementar modelos de ML de alta calidad. Las herramientas en torno al AutoML están cogiendo forma y, con suerte, la brecha entre modelos de ML y modelos generados en herramientas AutoML se reducirá, lo que permitirá una mayor participación en el sector.

Gracias al aparecimiento de estas herramientas a surgido la noción de la democratización de la inteligencia artificial, que está permitiendo a las personas que no están capacitadas formalmente en matemáticas, estadística, ciencias de la computación y campos relacionados diseñar, desarrollar y utilizar modelos predictivos. Por lo tanto el campo del AutoML tiene el potencial para la democratización de la inteligencia artificial a las masas. [6].

Para la elaboración de estos modelos surgieron varias herramientas en el mercado, cada una con sus distintivas características, sus ventajas y desventajas, y demás. Con estas herramientas, el ingeniero de datos no necesita definir su propia arquitectura, hallar el modelo ideal ni la combinación de parámetros adecuada, puesto que AutoML hace todo eso por él.

En resumen, cuando hablamos de Auto Machine Learning nos referimos a métodos para automatizar la selección de modelos y optimización de hiperparámetros. Tales mecanismos existen para algoritmos como random forests, redes neuronales, gradient boosting machines y muchos otros que se contemplaran en los siguientes capítulos.

1.2. Objetivos

El objetivo principal en este proyecto es realizar el análisis y experimentación de las principales o más relevantes tecnologías del AutoML. Identificar sus principales características, como también hacer validaciones de estas herramientas valorando sus ventajas y desventajas.

Capítulo 2

Auto Machine Learning

AutoML es un proceso que automatiza, utilizando algoritmos de la Inteligencia Artificial (IA), cada paso del proceso de ML, desde el preprocesamiento de datos hasta la implementación del modelo ML, Permite que los científicos que no son científicos de datos (como los desarrolladores de software) utilicen técnicas de aprendizaje automático sin necesidad de experiencia en el campo. En la siguiente figura, podemos ver un simple representación de las entradas y salidas de un sistema AutoML (ver Figura 2.1).

Figura 2.1: Esquema general del AutoML

2.1. Ciclo de Vida

Las técnicas de ML funcionan muy bien cuando se trata de encontrar patrones en grandes conjuntos de datos. Actualmente, se utilizan estas técnicas para la detección de anomalías, segmentación de clientes, análisis de abandono de clientes, pronósticos de demanda, mantenimiento predictivo y optimización de precios, entre cientos de otros casos de uso [1].

Un ciclo de vida típico de ML se compone de recopilación de datos, tratamiento de datos, gestión del ciclo, reentrenamiento de modelos e implementación de modelos, durante los cuales el tratamiento de datos es normalmente, la tarea que consume más tiempo.

El objetivo del AutoML es simplificar y democratizar los pasos de este ciclo. Originalmente, el enfoque clave del AutoML es la selección de modelos y el ajuste de hiperparámetros, es decir, encontrar el mejor modelo de ejecución para el trabajo y los parámetros correspondientes que funcionan mejor para el problema. En la siguiente figura vemos el esquema del ciclo de vida (Ver Figura 2.1).

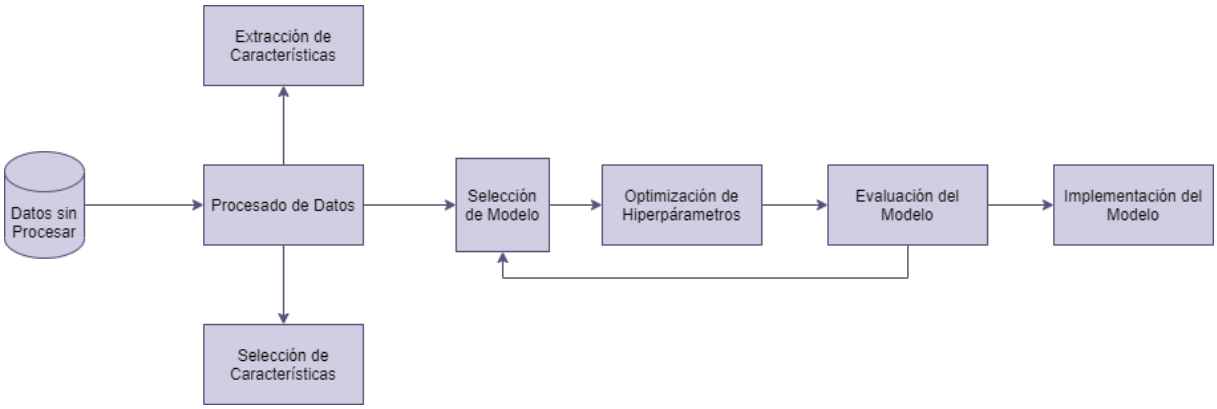


Figura 2.2: Ciclo de vida del AutoML

2.2. Tipos de AutoML

Hay tres áreas clave en el proceso del AutoML: la ingeniería de características, la búsqueda de arquitectura neuronal y la optimización de hiperparámetros. Son las más prometedoras para la democratización de la inteligencia artificial y el AutoML.

Algunas Técnicas de ingeniería de características automatizadas más comunes son expandir / reducir el conjunto de datos, organizar jerárquicamente transformaciones, meta aprendizaje y aprendizaje reforzado. Para la búsqueda arquitectónica (también conocida como búsqueda de arquitectura neuronal), tenemos algoritmos evolutivos, búsqueda local, meta aprendizaje, refuerzo del aprendizaje, el aprendizaje por transferencia, el morfismo de la red y la optimización continua que suelen ser los mas utilizados.

Por último, tenemos la optimización de hiperparámetros, que es el arte y la ciencia de encontrar el tipo correcto de parámetros fuera del modelo. Se utilizan una variedad de técnicas aquí, incluida la optimización bayesiana, algoritmos evolutivos, funciones de Lipchitz, búsqueda local, meta aprendizaje, optimización de enjambres de partículas, búsqueda aleatoria y aprendizaje por transferencia, por nombrar unos pocos. En la siguiente figura tenemos un esquema de tipos de AutoML dependiendo de las técnicas usadas 2.3.

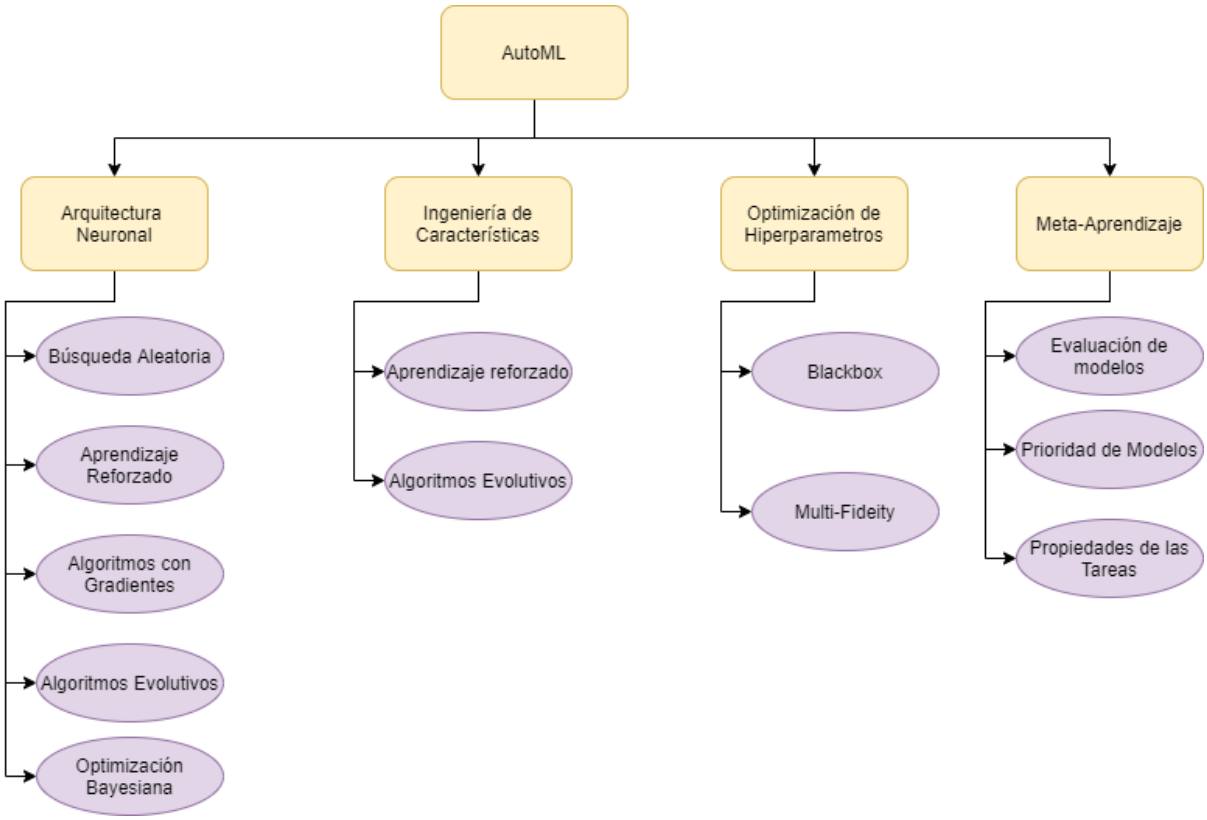


Figura 2.3: Tipos de AutoML

En el capitulo 3, proporcionaremos una descripción detallada de estas tres áreas clave del AutoML.

2.3. Ventajas y Desventajas del AutoML

En esta sección veremos las ventajas y desventajas del uso de herramientas AutoML para la generación de modelos de ML [18, 12]:

Ventajas:

1. Obtienes resultados más rápidos. Con AutoML, puedes omitir gran parte del trabajo del aprendizaje automático y, como resultado, ahorrar algo de tiempo. Es perfecto si se está construyendo un prototipo para probar un producto que se ajusta al mercado u obtener una experiencia temprana de un modelo de machine learning.

2. Es menos probable que esté desactualizado. Un problema muy común en el desarrollo de la IA es que los modelos envejecen casi a medida que se desarrollan. La tecnología de la inteligencia artificial se mueve tan rápido que lo que hoy podría ser de vanguardia y con mucho trabajo detrás, mañana podría quedarse atrás de sus competidores. Con AutoML, esa labor recae en los grandes proveedores de tecnología AutoML que tienen la economía de escala para que puedan invertir en mantenerse a la vanguardia.

3. Con una solución alojada en AutoML, ahorrará mucho tiempo y código al no tener que construir la infraestructura circundante. En promedio, el 95 % del código en las soluciones de aprendizaje automático es un código adhesivo que construye la infraestructura en torno a los modelos. El código de aprendizaje automático real es solo del 5 %.

4. Necesitas menos experiencia. Un problema común en la IA es que requiere mucha experiencia y, por lo general, involucra a varios tipos diferentes de expertos. Eso es extremadamente caro y los expertos son difíciles de encontrar. Se pueden ahorrar muchos de esos problemas con el AutoML.

Desventajas:

1. Se obtiene menos información sobre los datos. Una de las grandes ventajas de realizar un modelo de aprendizaje automático a mano es que obtienes muchos conocimientos sobre por qué tus modelos no funcionan como se esperaba. Esto le brinda información valiosa, para saber qué datos se podrían necesitar recopilar para obtener los resultados necesarios.

2. AutoML es inflexible. Es posible que obtenga resultados rápidamente, pero como ocurre con la mayoría de los equipos de IA, no pasará mucho tiempo antes de que los requisitos cambien y, con una solución de AutoML, corre el riesgo de que estos requisitos queden repentinamente fuera del alcance de los que AutoML puede manejar. Si eso sucede, se debe comenzar de nuevo con un modelo personalizado.

3. El coste de funcionamiento puede ser demasiado alto cuando escala. Una herramienta de pago es ideal para soluciones de pequeña a mediana escala. Pero si se esperan volúmenes muy altos de uso, es posible que se termine gastando demasiado dinero en los distintos proveedores. En este caso, una mejor solución sería crear y alojar los modelos en local.

4. La calidad probablemente no será la más alta. Si la característica más importante de su solución es la más alta calidad en comparación con los competidores, AutoML probablemente se quede corto. AutoML son modelos generalizados y, en la mayoría de los casos, no pueden competir con modelos verdaderamente especializados que han sido cuidadosamente construidos y adaptados al problema específico desde cero.

2.4. Ecosistema del AutoML

Casi parece redundante señalar que el AutoML es un campo en rápido crecimiento pero está lejos de ser un producto comercial, los marcos existentes están en constante evolución y las nuevas ofertas y plataformas se están generalizando. En este sección discutiremos algunos de estos frameworks y librerías en detalle y en las siguientes subsecciones comentaremos las herramientas más populares en el mercado, tanto como de código abierto (open source), como servicios comerciales.

2.4.1. Plataformas Open Source

Al revisar la historia del AutoML, es evidente que, en los primeros días, el enfoque siempre había estado en la optimización de hiperparámetros. Las herramientas principales, como AutoWeka y HyperoptSkLearn, y más tarde TPOT, tenían un enfoque original en el uso de la optimización bayesiana para encontrar los hiperparámetros más adecuados para el modelo [5, 10].

Sin embargo, esta tendencia se cambio para incluir la selección de modelos, finalmente todo el proceso gira en torno en incluir la selección de características, el preprocesamiento, la construcción y la limpieza de datos. La siguiente tabla muestra algunas de las herramientas de AutoML más destacadas, incluyendo TPOT, AutoKeras, auto-sklearn y Featuretools, junto con sus técnicas de optimización, tareas de aprendizaje automático y marcos de entrenamiento (Ver tabla 2.1).

	Lenguaje	***Extracción Características***	***Técnica AutoML***	***Meta Aprendizaje***
AutoWeka	JAVA	Sí	Optimización Bayesiana	No
AutoSklearn	Python	Sí	Optimización Bayesiana	Sí
TPOT	Python	Sí	Algoritmo Genético	No
Hyeropt-Sklearn	Python	Sí	Optimización Bayesiana y Búsqueda Aleatoria	No
AutoStacker	Python	Sí	Algoritmo Genético	Sí
AlphaD3M	Python	Sí	Aprendizaje Reforzado	Sí
OBOE	Python	No	Filtrado Colaborativo	Sí
PMG	Python	Sí	Filtrado Colaborativo y Optimización Bayesiana	Sí

Tabla 2.1: Tabla de Herramientas Open Source

Las herramientas mas destacadas son:

1. **AutoWeka:** Weka, abreviatura de Waikato Environment for Knowledge Analysis, es un biblioteca de código abierto que proporciona una colección de herramientas de visualización y algoritmos para el análisis de datos y modelado predictivo. Auto-Weka es similar a auto-sklearn pero está construido sobre

Weka e implementa los enfoques para la selección del modelo, optimización de hiperparámetros y más. El kit de herramientas está disponible en GitHub para descargar: github.com/automl/autoweka.

2. **AutoSklearn:** es una biblioteca de AutoML popular para el desarrollo en Python. Auto-sklearn es un kit de herramientas AutoML que realiza la selección de algoritmos y ajuste de hiperparámetros mediante optimización bayesiana, met aaprendizaje y conjunto construcción. El kit de herramientas está disponible en GitHub para descargar: github.com/automl/auto-sklearn.

3. **TPOT:** La herramienta de optimización de canalización basada en árboles, o TPOT para abreviar,es un producto del Laboratorio de Genética Computacional de la Universidad de Pensilvania. TPOT es un herramienta de AutoML escrita en Python. Ayuda a construir y optimizar canalizaciones de ML con programación genética. Construido sobre scikit-learn, TPOT ayuda a automatizar la selección de funciones, preprocesamiento, construcción, selección de modelos y optimización de hiperparámetros. El kit de herramientas está disponible en GitHub para descargar: github.com/ EpistasisLab/tpot.

4. **Hyeropt-Sklearn:** es una biblioteca de código abierto para AutoML a gran escala y es un conjunto para HyperOpt que admite AutoML con HyperOpt para la popular biblioteca de AutoML Scikit-Learn, incluye el conjunto de transformaciones de preparación de datos y algoritmos de clasificación y regresión. El kit de herramientas está disponible en GitHub para descargar: github.com/hyperopt/hyperopt-sklearn.

5. **AlphaD3M:** es un sistema AutoML que busca modelos automáticamente y deriva canalizaciones de un extremo a otro que leen, preprocesan los datos y entrenan el modelo. AlphaD3M aprovecha los avances recientes en el aprendizaje por refuerzo profundo y es capaz de adaptarse a diferentes dominios de aplicación y problemas a través del aprendizaje incremental. El kit de herramientas está disponible en GitHub para descargar: gitlab.com/ViDA-NYU/d3m/alphad3m.

6. **OBOE:** y TensorOboe son sistemas de AutoML que utilizan el filtrado colaborativo para encontrar buenos modelos para tareas de aprendizaje supervisadas dentro de un límite de tiempo especificado por el usuario. Posteriormente, se pueden realizar más ajustes de hiperparámetros. El kit de herramientas está disponible en GitHub para descargar: github.com/udellgroup/oboe

2.4.2. Plataformas Comerciales

Ahora, veamos algunas herramientas y plataformas comerciales que han desarrollado las grandes compañías. (Ver Tabla 2.2).

Herramientas	*Características*	*Lanzamiento*	*Sitio Web*
DataRobot	Optimización de Hiperparámetros	2012	datarobot.com
Google Cloud AutoML	Extracción De Características, Optimización de Hiperparámetros y Búsqueda de arquitectura neuronal	2018	cloud.google.com/automl
Amazon SageMaker	Extracción De Características, Optimización de Hiperparámetros y Búsqueda de arquitectura neuronal	2017	aws.amazon.com/es/sagemaker/
Microsoft Azure AutoML	Extracción De Características, Optimización de Hiperparámetros y Búsqueda de arquitectura neuronal	2018	hazure.microsoft.com/en-us/services/machine-learning/ automatedml/
H2O Driverless AI	Extracción De Características	2012	h2o.ai/products/h2o-driverless-ai/
IBM AutoAI Watson	Extracción De Características y Optimización de Hiper parámetros	2017	ibm.com/es-es/cloud/watson-studio/autoai

Tabla 2.2: Tabla de Herramientas Comerciales

Entre las herramientas destacadas tendríamos las siguientes.

1. **DataRobot:** plataforma patentada de AutoML. Como uno de los líderes en el espacio del AutoML, Data Robot afirma "automatizar el proceso de un extremo a otro para la construcción, implementación y mantenimiento de la IA a escala". El repositorio de modelos de Data Robot contiene código abierto, así como algoritmos y enfoques patentados para científicos de datos, con un enfoque para resultados comerciales. Las ofertas de Data Robot están disponibles tanto para la nube como para implementaciones locales.

2. **Google Cloud AutoML:** Integrada en la plataforma Google Cloud Compute, la oferta de Google Cloud AutoML tiene como objetivo ayudar a entrenar modelos de ML personalizados de alta calidad con un mínimo de esfuerzo y experiencia en ML. Esta oferta proporciona AutoML Vision, AutoML Video Intelligence, AutoML Natural Language, AutoML Translation y AutoML Tables para análisis de datos estructurados.

3. **Amazon SageMaker:** AWS ofrece una amplia variedad de capacidades en torno a la inteligencia artificial y el ML. El AutoPilot de SageMaker es una de estas ofertas que ayuda a crear, entrenar y ajustar modelos automáticamente como parte del ecosistema de AWS. SageMaker Autopilot proporciona un sistema automatizado de extremo a extremo del ciclo de vida de ML que incluye ingeniería automática de características, selección de modelos y algoritmos, ajuste, implementación y clasificación del modelo en función del rendimiento.

4. **Microsoft Azure AutoML:** Microsoft Azure proporciona capacidades de AutoML para ayudar a los científicos de datos a desarrollar modelos con velocidad y a escala. La plataforma ofrece ingeniería de características automatizada tales como imputación de valor perdido, transformaciones y codificaciones, drop ping de alta cardinalidad y demás. El aprendizaje automático automatizado de Azure también admite previsión de series de tiempo, selección de algoritmos, ajuste de hiperparámetros, barreras para mantener el sesgo del modelo en jaque, y una tablas de puntuación para modelos de clasificación y regresión.

5. **H2O Driverless AI:** La oferta comercial de H2O Driverless AI es una plataforma AutoML que aborda las necesidades de ingeniería de características, búsqueda de arquitectura neuronal y generación de procesos. La función "trae tu propia receta" es única (aunque ahora está siendo adaptada por otros proveedores) y se utiliza para integrar algoritmos personalizados. El producto comercial tiene amplias posibilidades y una interfaz de usuario con muchas funcionalidades.

Capítulo 3

Principales técnicas y algoritmos del AutoML

El proceso del AutoML consta de varios procesos: preparación de datos, ingeniería de características, generación de modelos, y evaluación de modelos. La generación de modelos se puede dividir aún más en la búsqueda espacial y métodos de optimización. La búsqueda espacial define los principios de diseño de los modelos ML, que se pueden dividir en dos categorías: los modelos tradicionales de ML (p. ej., SVM y KNN) y en arquitecturas neuronales profundas (DNN). Los métodos de optimización son clasificados en optimización de hiperparámetros (HPO) y Optimización de Arquitectura (AO), donde el primero indica el relacionado al entrenamiento de parámetros (por ejemplo, la tasa de aprendizaje y el tamaño del lote), y este último indica los parámetros relacionados con el modelo (por ejemplo, el número de capa para arquitecturas neuronales y el número de vecinos para KNN) [17]. La búsqueda de la arquitectura Neuronal (NAS) consta de tres componentes importantes: la búsqueda espacial de arquitecturas neuronales, métodos de AO y métodos de evaluación de modelos. En la siguiente figura podemos ver el esquema del proceso mas detallado (Ver Figura 3.1).

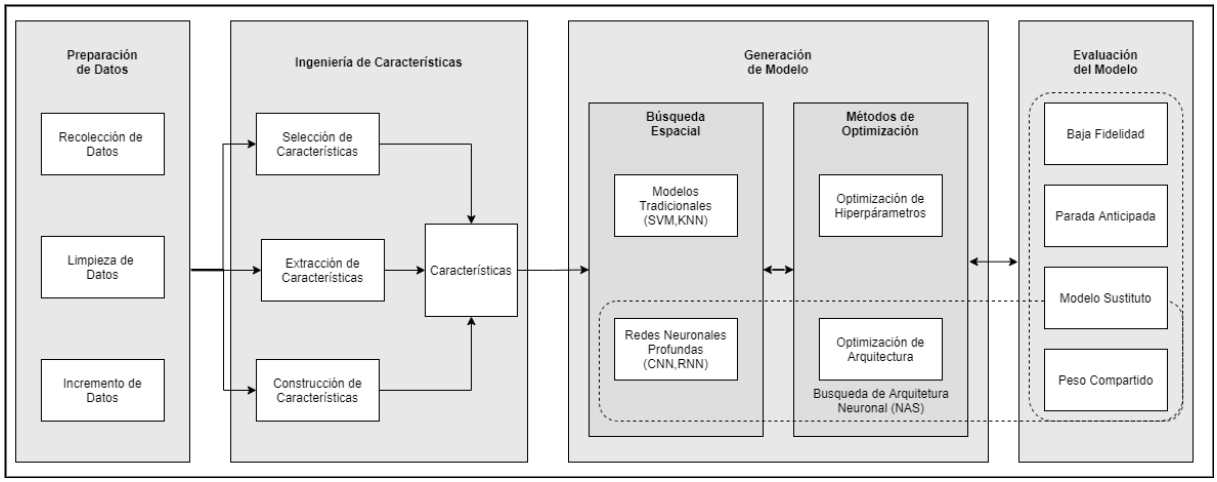


Figura 3.1: Proceso del AutoML detallado

En las siguientes secciones veremos las partes mas importantes dentro del proceso de una herramienta AutoML.

3.1. Ingeniería de Características

La ingeniería de características es el arte y la ciencia de extraer y seleccionar los atributos correctos de un conjunto de datos. Es un arte, porque no solo requiere experiencia en la materia, sino también conocimiento del dominio y comprensión de las preocupaciones éticas y sociales. Desde la perspectiva de un científico, la importancia de una característica está altamente correlacionada con su impacto en el resultado. La importancia de la característica en el modelado predictivo mide la cantidad que influye una característica en el objetivo, lo que facilita en retrospectiva la asignación de clasificación a los atributos con el mayor impacto.

Extraer una característica del conjunto de datos requiere la generación de características binarias categóricas basada en columnas con múltiples valores posibles, escalando las características, eliminando características correlacionadas, agregando interacciones de características, sustituyendo características cíclicas y manejo de escenarios de datos / tiempo. Los campos de fecha, por ejemplo, dan como resultado varias características, como año, mes, día, temporada, fin de semana / día laborable, festivo y período de vacaciones. Una vez extraído, seleccionar una característica de un conjunto de datos requiere la eliminación de características escasas y de baja varianza, así como aplicar técnicas de reducción de dimensionalidad como Análisis de Componentes Principales (PCA) para hacer manejable el número de funciones.

3.2. Selección del Modelo

La búsqueda espacial define los principios de diseño de las arquitecturas neuronales.Los posibles distintos escenarios requieren diferentes búsquedas espaciales. El espacio de búsqueda define las estructuras del modelo que se pueden diseñar y optimizar en un principio. Los tipos de modelos se pueden dividir en dos categorías: modelos tradicionales de ML, como soporte máquina de vectores (SVM) y algoritmo de k vecinos más cercanos (KNN), y en redes neuronales profundas (DNN) [17].

3.3. Optimización del Modelo

La optimización de hiperparámetros (HPO) tiene como objetivo encontrar una configuración de hiperparámetros de buen rendimiento para un modelo de aprendizaje automático determinado en un conjunto de datos, incluido el modelo de aprendizaje automático, sus hiperparámetros y otros pasos de procesamiento de datos.

El método de optimización de la arquitectura (AO) define cómo orientar la búsqueda para encontrar de manera eficiente la arquitectura del modelo con mejor rendimiento después de que se defina la búsqueda espacial. Hay dos tipos de parámetros para los métodos de optimización: hiperparámetros utilizados para la formación, como la tasa de aprendizaje, y los utilizados para diseño del modelo, como el tamaño del filtro y el número de capas para DNN. Ahora veremos estos

dos campos mas detallados.

3.3.1. Búsqueda Arquitectura Neuronal

La selección de modelo puede ser un desafío. En el caso de la regresión, es decir, predecir un valor numérico, puedes elegir entre regresión lineal, árboles de decisión, random forest, regresión de lazo versus cresta, red elástica de k-medias, métodos de aumento de gradiente, que incluyen XGBoost y SVM, entre muchos otros. Para la clasificación, que en otras palabras, separando las cosas por clases, tienes logística de regresión, random forest, AdaBoost, aumento de gradiente y clasificadores basados en máquinas de vectores de soporte (SVM) a tu disposición [2].

La arquitectura neuronal tiene la noción de espacio de búsqueda, que define qué arquitecturas se pueden utilizar en principio. Luego, se debe definir una estrategia de búsqueda que describa el uso de la compensación exploración-explotación. Finalmente, tiene que haber una actuación estrategia de estimación, que estima el desempeño de la arquitectura candidata. Esto incluye entrenamiento y validación de la arquitectura.

Existen varias técnicas para realizar la exploración de la búsqueda espacial.Las más comunes incluyen redes neuronales estructuradas en cadena, redes de múltiples ramas, búsqueda basada en células y enfoques de optimización utilizando la arquitectura existente. Las estrategias de búsqueda incluyen búsqueda aleatoria, enfoques evolutivos, optimización bayesiana, refuerzo de aprendizaje y enfoques de optimización sin gradientes y basados en gradientes, como Búsqueda de Arquitectura Diferenciable (DARTS). La estrategia de búsqueda jerárquica explora los distintos espacios de búsqueda de arquitectura, utilizando la búsqueda de árboles de Montecarlo o la escalada de colina (Hill climbing) que es popular ya que ayuda a descubrir arquitecturas de alta calidad acercándose rápidamente a mejores arquitecturas. En métodos basados en gradientes, la suposición subyacente de un espacio de búsqueda continua facilita DARTS, que, a diferencia del aprendizaje por refuerzo tradicional o los enfoques de búsqueda evolutiva, explora el espacio de búsqueda usando el descenso de gradiente. En la siguiente figura vemos un resumen de las técnicas del NAS (Ver Figura 3.2).

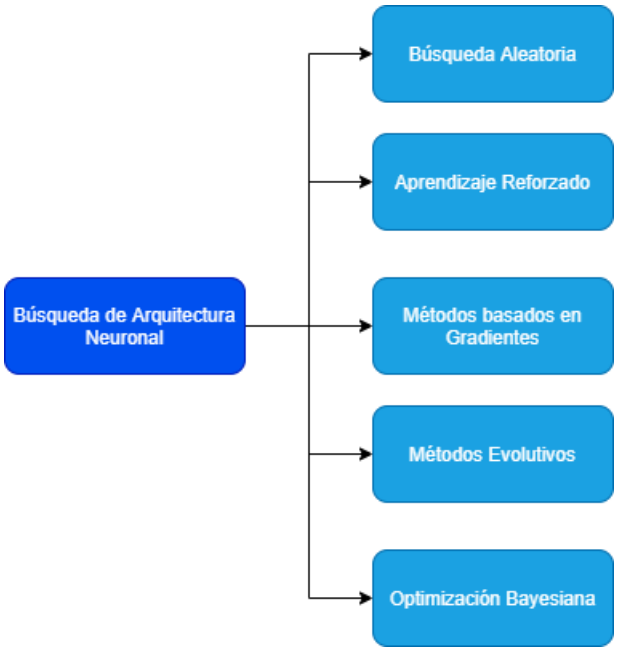


Figura 3.2: Taxonomía de las técnicas del NAS

3.3.2. Optimización Hiperparámetros

Hagamos un resumen rápido de lo que son los hiperparámetros. Cada modelo tiene sus parámetros internos y externos. Los parámetros internos o los parámetros del modelo son intrínsecos al modelo, como el peso o la matriz de predicción, mientras que los parámetros externos también se conocen como hiperparámetros, están "fuera"del modelo como la tasa de aprendizaje y el número de iteraciones. Por ejemplo, en k-medias, k representa el número de grupos requeridos y las etapas se utilizan para especificar el número de pasadas realizadas sobre los datos de entrenamiento. Ambos son ejemplos de hiperparámetros, es decir, parámetros que no son intrínsecos al modelo en sí mismo. De manera similar, la tasa de aprendizaje para entrenar una red neuronal, C y sigma para soporte de máquinas vectoriales (SVM), k número de hojas o profundidad de un árbol, factores latentes en una matriz de factorización, el número de capas ocultas en una red neuronal profunda, y demás son ejemplos de hiperparámetros [8].

Para encontrar los hiperparámetros correctos, hay varios enfoques, pero primero veamos qué diferentes tipos de hiperparámetros existen. Los hiperparámetros pueden ser continuos, por ejemplo:

- La tasa de aprendizaje de un modelo.
- El número de capas ocultas.
- El número de iteraciones
- Tamaño del lote

Los hiperparámetros también pueden ser categóricos, por ejemplo, el tipo de operador, función de activación, o la elección del algoritmo. También pueden ser

condicionales, por ejemplo, seleccionando el tamaño del núcleo convolucional si se utiliza una capa convolucional, o el ancho del núcleo si se selecciona un kernel de función de base radial (RBF) en una SVM. Dado que hay varios tipos de hiperparámetros, también hay una variedad de técnicas de optimización de hiperparámetros. Cuadrícula, búsqueda aleatoria, optimización bayesiana, técnicas evolutivas, enfoques basdos en bandit multibrazos y técnicas basadas en el descenso de gradientes.

Las técnicas más simples para el ajuste de hiperparámetros son la búsqueda manual, de cuadrícula (grid) y aleatoria (random). El giro manual, como su nombre indica, se basa en la intuición y la conjetura basada en las pasadas experiencias. La búsqueda en cuadrícula y la búsqueda aleatoria son ligeramente diferentes, ya que elige un conjunto de hiperparámetros para cada combinación (cuadrícula), o aleatoriamente e iterar hasta conservar los de mejor rendimiento. Sin embargo esto se puede ir de manos computacionalmente a medida que aumenta el espacio de búsqueda.

La otra técnica destacada es la optimización bayesiana, en la que se comienza con una combinación aleatoria de hiperparámetros y se utiliza para construir un modelo sustituto. Luego, usar este modelo sustituto para predecir cómo funcionaria otras combinaciones de hiperparámetros. Como principio general, la optimización bayesiana construye un modelo de probabilidad para minimizar la función objetivo, utilizando los valores pasados para seleccionar los valores futuros, y eso es exactamente lo que es bayesiano. Como se conoce en el universo bayesiano, sus observaciones son menos importantes que sus predicciones anteriores.

La naturaleza codiciosa de la optimización bayesiana se controla mediante el intercambio de exploración y explotación (mejora esperada), la asignación de evaluaciones de tiempo fijo, el establecimiento de umbrales,etc . Hay variaciones de estos modelos sustitutos que existen, como random forest sustituto y aumento de gradiente sustituto, que utilizan las mencionadas técnicas para minimizar la función sustituta.

La clase de métodos basados en la población (también llamados técnicas metaheurísticas u optimización de métodos de muestras) también se usan ampliamente para realizar modificaciones de hiperparámetros, siendo la programación genética (algoritmos evolutivos) la más popular, donde se agregan, mutan, seleccionan, cruzan y ajustan los hiperparámetros. Una partícula swarm se mueve hacia las mejores configuraciones individuales cuando el espacio de configuración es actualizado en cada iteración. Por otro lado, los algoritmos evolutivos funcionan manteniendo un espacio de configuración y mejoran haciendo cambios más pequeños y combinando soluciones para construir una nueva generación de configuración de hiperparámetros.

En la siguiente figura veremos un resumen de las técnicas de optimización de hiperparámetros. Podemos contemplar que hay dos tipos de técnicas unas basadas en la llamada caja negra (Black Box) y otras englobadas en multi fidelidad (Multy-Fidelity). Dentro de las técnicas basadas en black box tenemos la búsqueda grid, la búsqueda aleatoria, la optimización bayesiana, Recocido simulado (Simu-

lated Annealing) y algoritmos genéticos. Y en técnicas de multi fidelidad tenemos un método basado en la curva de aprendizaje y otro método basado en bantid, el cual tiene dos técnicas, Reducción a la mitad sucesiva y Hyperband (Ver figura 3.3).

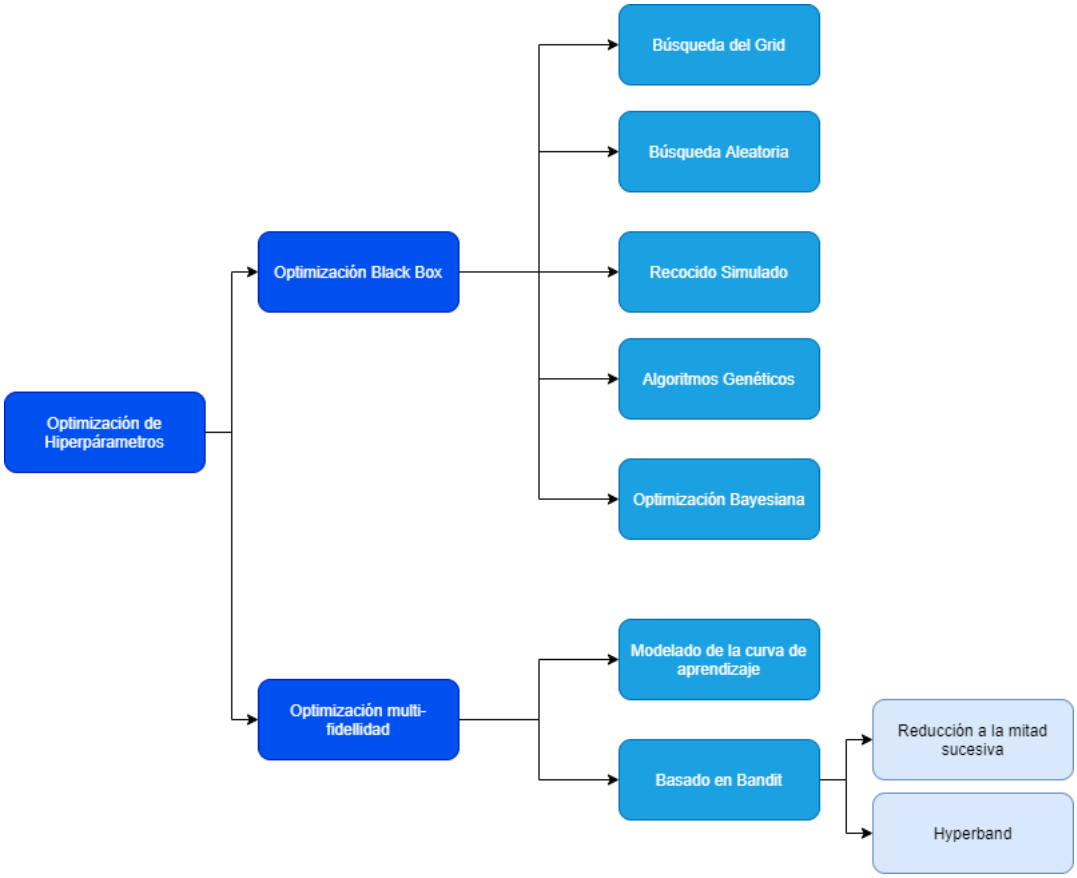


Figura 3.3: Taxonomía de las técnicas del HPO

Capítulo 4

Experimento con tecnologías seleccionadas

Con las distintas herramientas seleccionadas se realizara una experimentación, la cual servirá para hacer un análisis detalla y comparación entre ellas. Primero haremos una experimentación preliminar de ejemplo y después procederemos a hacer una experimentación completa. Todo el código desarrollado en la experimentación se encuentra en : `https://github.com/alu0101046853/TFG_Chesen`

4.1. Selección de Tecnologías

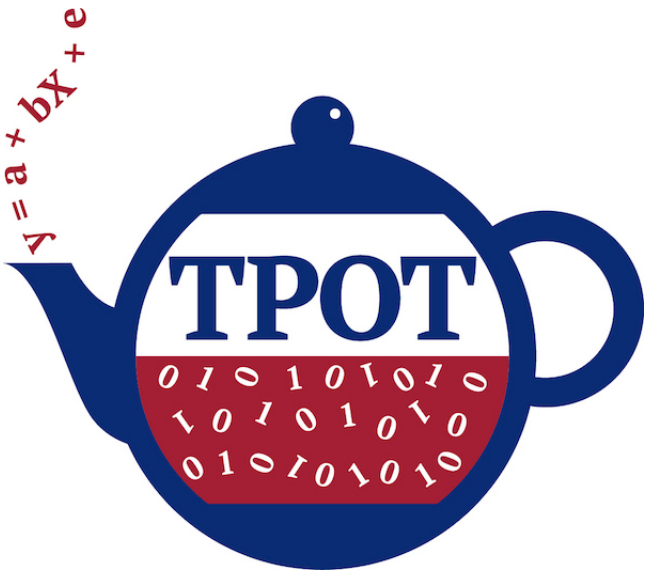
Hemos seleccionado las herramientas open source: AutoWeka, AutoKeras, AutoSklearn y TPOT, para comparar las distintas funcionalidades que ofrecen unas y otras.

AutoWeka [15] [16], El software de aprendizaje automático WEKA pone modelos de última generación con técnicas de aprendizaje a disposición incluso de usuarios novatos. Sin embargo, estos usuarios no suelen saber cómo elegir entre las docenas de procedimientos de aprendizaje automático implementados en WEKA y la configuración de hiperparámetros de cada procedimiento para lograr un buen rendimiento. Auto-WEKA aborda este problema al tratar a todo WEKA como un marco de aprendizaje automático único y altamente paramétrico, y al utilizar la optimización bayesiana para encontrar una instanciación sólida para un conjunto de datos determinado.

AutoKeras [9], Keras es uno de los marcos de aprendizaje profundo más utilizados y es una parte integral del ecosistema TensorFlow 2.0. Auto-Keras es un método novedoso para la búsqueda eficiente de arquitectura neuronal con morfismo de red, que permite la optimización bayesiana. Esto ayuda a la búsqueda de la arquitectura neuronal al diseñar un kernel de red neuronal y un algoritmo para optimizar las funciones de adquisición basados en árbol. Hay herramientas similares con el mismo objetivo, pero AutoKeras está especializada en buscar modelos de Aprendizaje Profundo (Deep Learning).

AutoSklearn [4, 3], scikit-learn (también conocido como sklearn) es una biblioteca ML muy popular para el desarrollo de Python. Como parte de este ecosistema auto-sklearn es un kit de herramientas autoML que realiza algoritmos de selección y ajuste de hiperparámetros mediante optimización bayesiana, metaaprendizaje, y construcción de conjuntos.

Basado en la selección de algoritmos combinados y la optimización de hiperparámetros (CASH), auto-sklearn aborda el problema de encontrar el mejor modelo y sus hiperparámetros al mismo tiempo. El motor automatizado de ML subyacente utiliza la recuperación de información (IR) y estadísticas de enfoque para seleccionar variedad de configuraciones, todas las cuales se utilizan como parte de la entrada de optimización bayesiana.



TPOT [11, 14, 13], o herramienta de optimización de canalización basada en árboles, es una biblioteca de código abierto para realizar el aprendizaje automático de forma automatizada con la programación en Python. Debajo de la superficie, utiliza la conocida biblioteca de aprendizaje automático scikit-learn para realizar la preparación, transformación y aprendizaje automático de datos. También usa GP, procedimientos para descubrir la canalización de mejor rendimiento para un conjunto de datos determinado.

4.2. Experimentación

4.2.1. Preparación

Para empezar la experimentación comenzaremos por elegir el entorno de desarrollo para el uso de estas tecnologías:

Weka

Dado que AutoWeka es un plugin del software de Weka tendremos que usarlo directamente en esta herramienta. Para instalar este software y usarlo en nuestra maquina lo único que tendremos que hacer es descargar el instalador desde la página oficial: `https://www.cs.waikato.ac.nz/ml/weka/`. Una vez instalado ya tendremos el software de Weka listo para usar (Ver figura 4.2.1).

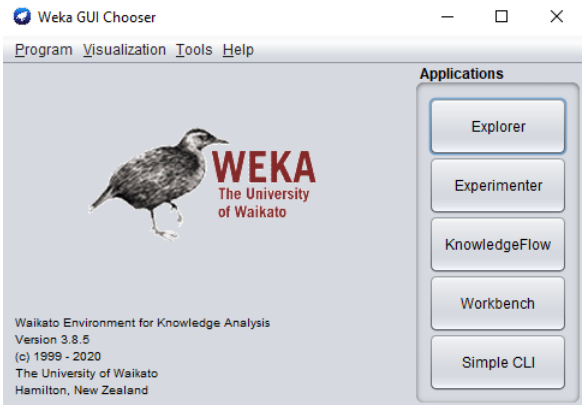


Figura 4.1: Software de Weka

Ahora solo tendremos que instalar el plugin de AutoWeka dentro del software de Weka, para ello iremos a Tools ->Packet Manager (O usar CTRL + U), dentro buscamos el plugin de AutoWeka y lo instalamos. Al acabar la instalación veremos como nos aparece el apartado de AutoWeka en la experimentación de Weka y ya tendremos AutoWeka listo para usar (Ver figura 4.2.1).

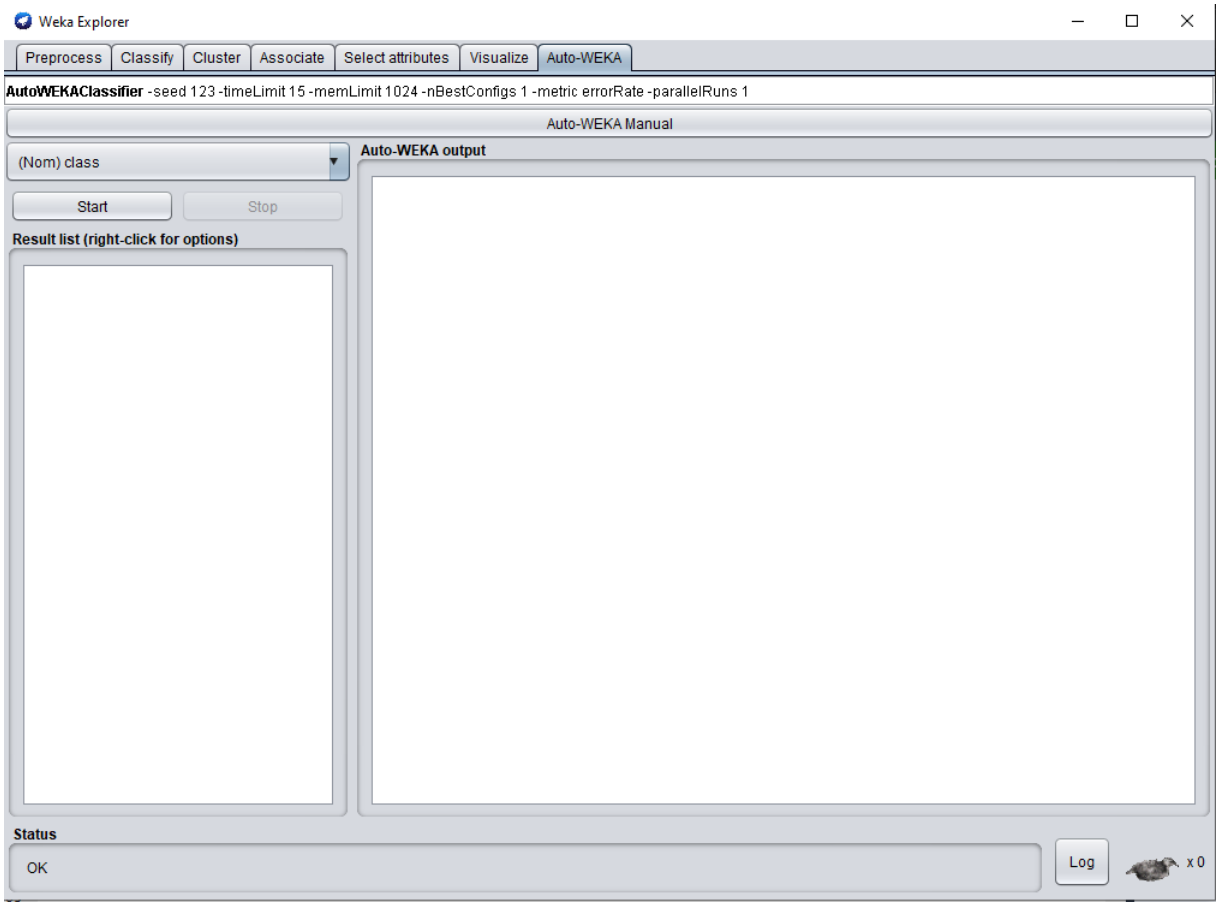


Figura 4.2: Software de AutoWeka

Google Colaboratory

Para el resto de herramientas (AutoKeras, AutoSklearn y TPOT), dado que son librerías de python utilizaremos el entorno de ejecución de cuadernos python en la nube de google. Google Colaboratory es un entorno de cuadernos Jupyter gratuito que se ejecuta completamente en la nube. Lo más importante es que Colab no requiere configuración, además, los miembros de un equipo pueden editar simultáneamente los cuadernos que se crean, de manera similar a la edición documentos en Google Docs. La mayor ventaja es que Colab admite las bibliotecas de aprendizaje automático más populares, que se pueden cargar fácilmente en tus cuaderno en la nube.

Figura 4.3: Logo Google Colab

Creamos un cuaderno de jupyter para cada una de las herramientas, para proceder a la instalación de las mismas.

AutoKeras

Para la instalación de AutoKeras solamente tendremos que seguir la guía de instalación en python desde su página oficial: `https://autokeras.com/install/`, y ejecutar esas lineas de código para instalarlas en nuestro entorno de desarrollo.

```
pip install git+https://github.com/keras-team/keras-tuner.git
pip install autokeras
```

AutoKeras ya estaría preparado para su uso.

AutoSklearn

Para la instalación de AutoSklearn solamente tendremos que seguir la guía de instalación en python desde su página oficial: `https://automl.github.io/auto-sklearn/master/installation.html`, y ejecutar esas lineas de código para instalarlas en nuestro entorno de desarrollo.

```
!curl https://raw.githubusercontent.com/automl/auto-sklearn/master
        /requirements.txt| xargs -n 1 -L 1 pip3 install
```

```
3 !pip3 install auto-sklearn
```

AutoSklearn ya estaría preparado para su uso.

TPOT

Para la instalación de TPOT solamente tendremos que seguir la guía de instalación en python desde su página oficial: `http://epistasislab.github.io/tpot/installing/`, TPOT te da la posibilidad de instalar plugins, como el de Dask para el entrenamiento del modelo en paralelo, elegimos lo que queremos instalar y ejecutamos esas lineas de código para instalarlas en nuestro entorno de desarrollo.

```
1 !pip install deap update_checker tqdm stopit xgboost
2 !pip install dask[delayed] dask[dataframe] dask-ml
3     fsspec>=0.3.3 distributed>=2.10.0
4 !pip install scikit-mdr skrebate
5 !pip install tpot
```

TPOT ya estaría preparado para su uso.

4.2.2. Experimentación preliminar

Instaladas todas las herramientas a utilizar procedemos a la experimentación preliminar, en la cual realizaremos el experimento de la búsqueda de un modelo de clasificación con el dataset de ejemplo que nos ofrece cada herramienta. La métrica que utilizaremos en todas las herramientas sera la métrica de la Classification Accuracy (Precisión), es lo que usualmente queremos decir cuando usamos el término exactitud. Es la relación entre el número de predicciones correctas y el número total de muestras de entrada. Funciona bien solo si hay el mismo número de muestras pertenecientes a cada clase, por ejemplo si tenemos 3 clases, y tenemos 100 muestras de estas, para que Accuracy funcione perfectamente tendrán que haber 100 muestras para las otras 2 clases. Podemos ver la sencilla formula de esta métrica (Ver figura 4.4).

$$Accuracy = \frac{Number\ of\ Correct\ predictions}{Total\ number\ of\ predictions\ made}$$

Figura 4.4: Formula de la métrica Accuracy

También utilizaremos la métrica de la matriz de confusión en el experimento final. Una matriz de confusión es una matriz N x N que se utiliza para evaluar el rendimiento de un modelo de clasificación, donde N es el número de clases objetivo. La matriz compara los valores objetivo reales con los predichos por el modelo de ML. Esto nos da una visión holística de qué tan bien se está desempeñando

nuestro modelo de clasificación y qué tipo de errores está cometiendo (Ver Figura 4.5).

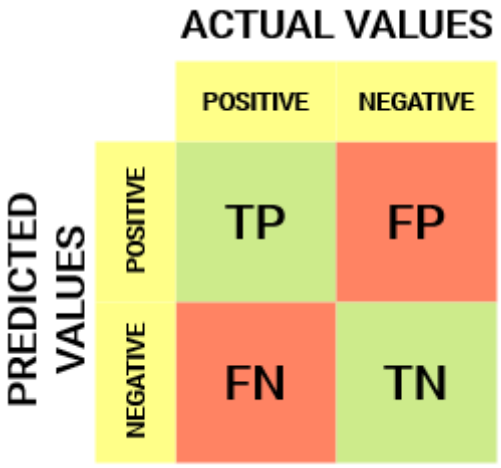


Figura 4.5: Matriz de Confusión

AutoWeka

Empezamos por la herramienta AutoWeka y utilizaremos el dataset de ejemplo Iris 2D una versión simplificada del famoso dataset IRIS el cual utilizaremos en la experimentación final, se amplia más en el apartado 4.3. Este dataset clasifica los tipos de flores iris dependiendo de 2 atributos que son el ancho y el largo del pétalo, en 3 tipos de flor iris: la setosa, la versicolor y la virginica.

Para empezar tendremos que cargar el dataset, simplemente le damos al botón de open file y elegimos el dataset con el queremos experimentar (Ver figura 4.6).

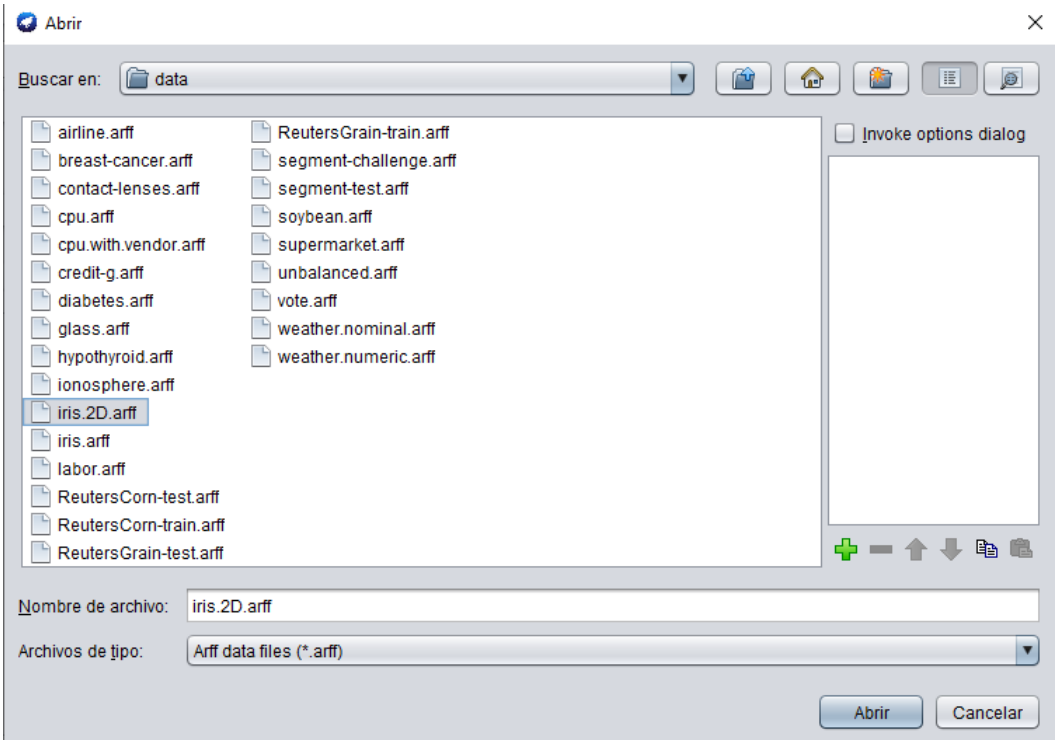


Figura 4.6: Selección de Dataset

Gracias al software de Weka podremos ver fácilmente las características de este dataset gracias a su interfaz interactiva, con esto podemos comprobar que tenemos los mismos números de muestras para las 3 distintas clases (Ver figura 4.7).

Figura 4.7: Gráfica de clases

Ahora accedemos a la pestaña de AutoWeka, es una pestaña sencilla e intuitiva. Elegimos sobre que atributo queremos clasificar, en este dataset es sencillo, la clase de flor, y abrimos la pestaña de hiperparámetros y vemos todos los que podemos modificar (Ver figura 4.8).

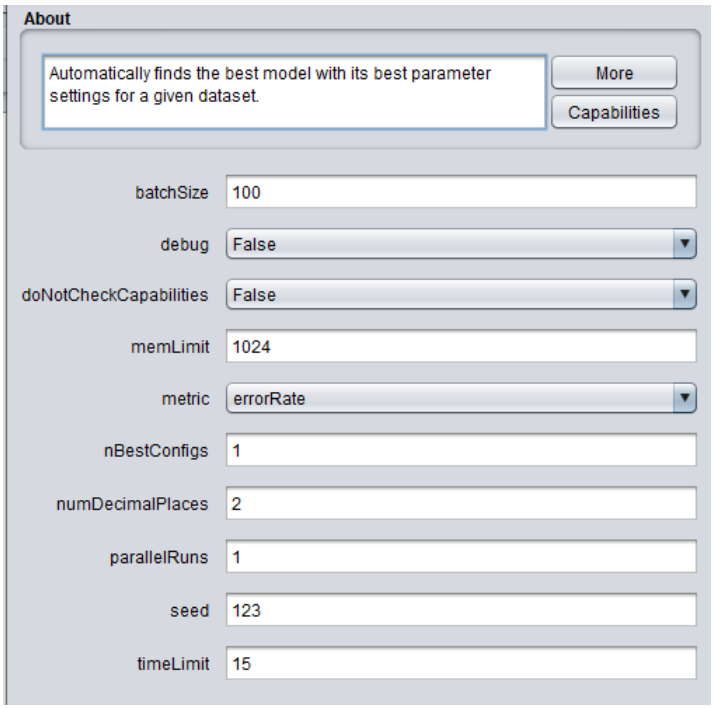


Figura 4.8: Opciones de hiperparámetros

Dado que es una experimentación preliminar los dejamos por defecto, y estaría todo listo para ejecutar. Le damos al botón de Start y esperamos unos 15 minutos.

Después de 15 minutos conseguimos el mejor modelo que ha encontrado AutoWeka con esos hiperparámetros tras haber probado unas 610 configuraciones distintas, Weka habla de la métrica de errorRate que es la metrica inversa a la Accuracy (1 - Accuracy). Por lo tanto vemos que nos a dado un errorRate de un 2 %, por lo tanto un Accuracy del 98 %. También podemos ver que el mejor algoritmo de clasificación que ha encontrado para este problema es el algoritmo JRip. También se observan muchas estadísticas del proceso, como la matriz de confusión, en la cual vemos que los únicos fallos son al clasificar 3 iris versicolor como iris virginica. Por último podemos ver una matriz desglosada de la métrica Accuracy (Ver figura 4.9).

```
Auto-WEKA result:
best classifier: weka.classifiers.rules.JRip
arguments: [-N, 3.370621681189805, -O, 5]
attribute search: weka.attributeSelection.GreedyStepwise
attribute search arguments: [-C, -R]
attribute evaluation: weka.attributeSelection.CfsSubsetEval
attribute evaluation arguments: []
metric: errorRate
estimated errorRate: 0.02
training time on evaluation dataset: 0.027 seconds

You can use the chosen classifier in your own code as follows:

AttributeSelection as = new AttributeSelection();
ASSearch asSearch = ASSearch.forName("weka.attributeSelection.GreedyStepwise", new String[]{"-C", "-R"});
as.setSearch(asSearch);
ASEvaluation asEval = ASEvaluation.forName("weka.attributeSelection.CfsSubsetEval", new String[]{});
as.setEvaluator(asEval);
as.SelectAttributes(instances);
instances = as.reduceDimensionality(instances);
Classifier classifier = AbstractClassifier.forName("weka.classifiers.rules.JRip", new String[]{"-N", "3.370621681189805", "-O", "5"});
classifier.buildClassifier(instances);


Correctly Classified Instances         147               98      %
Incorrectly Classified Instances         3                2      %
Kappa statistic                          0.97
Mean absolute error                      0.0252
Root mean squared error                  0.1122
Relative absolute error                  5.6604 %
Root relative squared error             23.7915 %
Total Number of Instances              150

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  0 50 |  c = Iris-virginica

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Iris-setosa
                 0,940    0,000    1,000      0,940    0,969      0,955    0,985     0,970     Iris-versicolor
                 1,000    0,030    0,943      1,000    0,971      0,957    0,985     0,943     Iris-virginica
Weighted Avg.    0,980    0,010    0,981      0,980    0,980      0,971    0,990     0,971
```

Figura 4.9: Resultados AutoWeka

AutoKeras

Con la librería de AutoKeras utilizaremos el dataset de ejemplo MNIST, que es una gran base de datos de dígitos escritos a mano que contemplan los números del 0 al 9, por lo tanto es una clasificación por imágenes. (El código se encuentra en el apéndice A.1).

Primero importamos los módulos que vayamos a utilizar en esta experimentación con AutoKeras.

```
1 import numpy as np
2 import tensorflow as tf
3 from tensorflow.keras.datasets import mnist
4 import matplotlib.pyplot as plt
5 import autokeras as ak
```

Primero NumPy, una biblioteca de python que se utiliza para trabajar con matrices. Después TensorFlow, la librería de ML de código abierto de Google que contiene muchas funciones útiles, en este caso cogemos el dataset de esta librería y lo importamos. También importamos la librería de pyplot de matplotlib que es una

biblioteca completa para crear visualizaciones estáticas, animadas e interactivas en python. Por último importamos el propio AutoKeras.

Después cargamos el dataset y los dividimos en datos de entrenamiento y datos de prueba. Para ver una pequeña muestra de las imágenes que tenemos en el dataset MNIST usamos pyplot y obtenemos esta imagen (Ver figura 4.10).

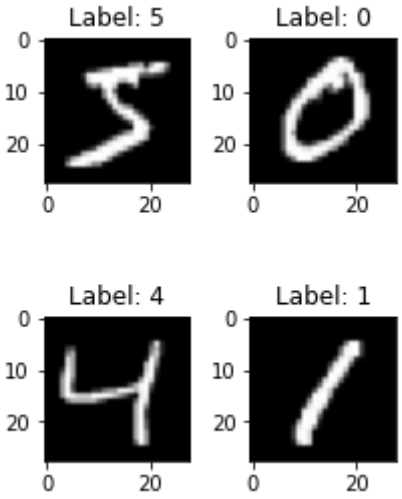


Figura 4.10: Gráfica de muestra del Dataset MNIST

Seguimos eligiendo el tipo de clasificación que utilizaremos para el AutoML, en este caso es una clasificación por imágenes.

```
clf = ak.ImageClassifier(overwrite=True, max_trials=1)
```

El parámetros de overwrite, es si queremos que se reescriba el proceso por si se ha hecho anteriormente, dado que el proceso de búsqueda de modelos para una clasificación por imágenes es extenso le indicamos que solo realice un intento. Y pasamos al proceso de entrenamiento.

```
clf.fit(x_train, y_train, epochs=10)
```

El algoritmo de entrenamiento recibe los datos de entrenamiento, en x_train tenemos las imágenes y en y_train su respectivo valor decimal, y le decimos que intente unos 10 epochs o época. Después hacemos una predicción con el mejor modelo que haya conseguido el algoritmo de AutoML y evaluamos ese mejor modelo conseguido.

```
predicted_y = clf.predict(x_test)
print(predicted_y)
print(clf.evaluate(x_test, y_test))
```

Por último cogemos el mejor modelo que hayamos conseguido y vemos un resumen de sus características y mostramos su arquitectura neuronal.

```
modelo_final = clf.export_model()
modelo_final.summary()
tf.keras.utils.plot_model(modelo_final,show_shapes=True)
```

Ejecutamos todas estas lineas de código y después de el proceso de AutoML de unos 23 minutos conseguimos nuestro modelo con una Accuracy del 96 %. También vemos las pruebas de las predicciones que son del 98 % (Ver Figura 4.11).

```
Trial 1 Complete [00h 24m 37s]
val_loss: 0.03986078500747681

Best val_loss So Far: 0.03986078500747681
Total elapsed time: 00h 24m 37s
INFO:tensorflow:Oracle triggered exit
Epoch 1/10
1875/1875 [==============================] - 178s 95ms/step - loss: 0.1606 - accuracy: 0.9506
Epoch 2/10
1875/1875 [==============================] - 182s 97ms/step - loss: 0.0710 - accuracy: 0.9781
Epoch 3/10
1875/1875 [==============================] - 183s 98ms/step - loss: 0.0574 - accuracy: 0.9826
Epoch 4/10
1875/1875 [==============================] - 183s 98ms/step - loss: 0.0496 - accuracy: 0.9847
Epoch 5/10
1875/1875 [==============================] - 183s 97ms/step - loss: 0.0434 - accuracy: 0.9861
Epoch 6/10
1875/1875 [==============================] - 183s 97ms/step - loss: 0.0398 - accuracy: 0.9876
Epoch 7/10
1875/1875 [==============================] - 185s 99ms/step - loss: 0.0367 - accuracy: 0.9888
Epoch 8/10
1875/1875 [==============================] - 184s 98ms/step - loss: 0.0335 - accuracy: 0.9892
Epoch 9/10
1875/1875 [==============================] - 182s 97ms/step - loss: 0.0332 - accuracy: 0.9889
Epoch 10/10
1875/1875 [==============================] - 183s 98ms/step - loss: 0.0285 - accuracy: 0.9909
INFO:tensorflow:Assets written to: ./image_classifier/best_model/assets
313/313 [==============================] - 10s 31ms/step
[['7']
 ['2']
 ['1']
 ...
 ['4']
 ['5']
 ['6']]
313/313 [==============================] - 9s 29ms/step - loss: 0.0366 - accuracy: 0.9886
[0.03662138804793358, 0.9886000156402588]
```

Figura 4.11: Resultados AutoKeras

También podemos ver el resumen de el modelo conseguido y un grafo de su arquitectura neuronal (Ver figura 4.12).

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 28, 28)]          0
_________________________________________________________________
cast_to_float32 (CastToFloat (None, 28, 28)            0
_________________________________________________________________
expand_last_dim (ExpandLastD (None, 28, 28, 1)         0
_________________________________________________________________
normalization (Normalization (None, 28, 28, 1)         3
_________________________________________________________________
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 10)                92170
_________________________________________________________________
classification_head_1 (Softm (None, 10)                0
=================================================================
Total params: 110,989
Trainable params: 110,986
Non-trainable params: 3
```

Figura 4.12: Resumen mejor modelo de Autokeras

Aquí podemos ver todas las capas de nuestra red neuronal y ver como entran y salen los parámetros de capa en capa. También tenemos la posibilidad de verlo en forma de grafo (Ver Apéndice B.2)

AutoSklearn

Seguimos con la herramienta AutoSklearn de python, utilizaremos el dataset de ejemplo Load Digits, que son imágenes de 8x8 que se categorizan como dígitos del 0 al 9 como MNIST, por lo tanto es una clasificación por imágenes(El código se encuentra en el apéndice A.2).

Primero importamos los módulos que vayamos a utilizar en esta experimentación con AutoSklearn.

```
1 import autosklearn.classification
2 import sklearn.model_selection
3 import sklearn.datasets
4 import sklearn.metrics
5 import matplotlib.pyplot as plt
```

Primero el paquete de modelos de AutoML para la clasificación de AutoSklearn, después importamos el paquete model_selection el cual utilizaremos para dividir el dataset para el entrenamiento y las predicciones. Importamos la librería de Sklearn datasets, la cual contiene muchos datasets de ejemplo, también importamos las métricas para medir el éxito de los modelos y por último el paquete pyplot

de matplotlib para los grafismos.

Cargamos el Dataset, Primero mostramos un ejemplo de imagen de nuestro dataset (Ver figura 4.13).

```
digits = sklearn.datasets.load_digits()
plt.gray()
plt.matshow(digits.images[0])
plt.show()
```

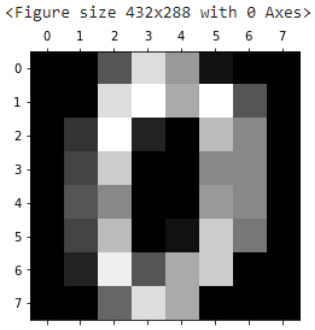


Figura 4.13: Ejemplo de Load Digits

Y después lo dividimos en datos de entrenamiento y datos de pruebas.

```
X, y = sklearn.datasets.load_digits(return_X_y=True)
X_train, X_test, y_train, y_test =
    sklearn.model_selection.train_test_split(X, y, random_state=1)
```

Seguimos eligiendo el tipo de clasificación que utilizaremos para el AutoML, en este caso es una clasificación por imágenes.

```
automl = autosklearn.classification.AutoSklearnClassifier()
```

Y pasamos al proceso de entrenamiento, y mostramos por pantalla las estadísticas del proceso de AutoML.

```
automl.fit(X_train, y_train)
print(automl.sprint_statistics())
```

El algoritmo de entrenamiento recibe los datos de entrenamiento, en x_train tenemos las imágenes y en y_train su valor decimal. Después hacemos una predicción con el mejor modelo que haya conseguido el algoritmo de AutoML y evaluamos ese mejor modelo conseguido.

```
y_hat = automl.predict(X_test)
print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))
```

Ejecutamos todas estas lineas de código y después de el proceso de AutoML de unos 40 minutos conseguimos nuestro modelo con una Accuracy del 99 %. También vemos las pruebas de la predicción que son del 98 %, Con un total de 133 algoritmos intentados (Ver Figura 4.14).

```
auto-sklearn results:
  Dataset name: f28031e6-0384-11ec-80fa-0242ac1c0002
  Metric: accuracy
  Best validation score: 0.991011
  Number of target algorithm runs: 133
  Number of successful target algorithm runs: 129
  Number of crashed target algorithm runs: 1
  Number of target algorithms that exceeded the time limit: 2
  Number of target algorithms that exceeded the memory limit: 1

Accuracy score 0.9888888888888889
```

Figura 4.14: Resultados AutoSklearn

TPOT

Y acabamos con la herramienta TPOT y utilizaremos el dataset de ejemplo Load Digits como con Sklearn, por lo tanto es una clasificación por imágenes de dígitos del 0 al 9 (El código se encuentra en el apéndice A.3).

Primero importamos los módulos que vayamos a utilizar en esta experimentación con TPOT.

```
from tpot import TPOTClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
```

Primero el paquete de modelos de AutoML para la clasificación de TPOT, después importamos el paquete model_selection el cual utilizaremos para dividir el dataset para el entrenamiento y la predicción, y por último importamos la librería de Sklearn datasets, la cual contiene muchos datasets de ejemplos.

Cargamos el dataset y lo dividimos en datos de entrenamiento en un 75 % y datos de pruebas el 25 %.

```
digits = load_digits()
X_train, X_test, y_train, y_test =
    train_test_split(digits.data, digits.target,
        train_size=0.75, test_size=0.25)
```

Seguimos eligiendo el tipo de clasificación que utilizaremos para el AutoML, en este caso es una clasificación. Los hiperparámetros los modificamos para que se generen 5 generaciones, que son las iteraciones en TPOT, el population size en 20, que es el numero de invididuales en cada generación en la programación genética que utiliza TPOT, y por último la semilla para la pseudo aleatoriedad del proceso.

```
pipeline_optimizer =
    TPOTClassifier
        (generations=5, population_size=20,
            random_state=42, verbosity=2)
```

Y pasamos al proceso de entrenamiento.

```
pipeline_optimizer.fit(X_train, y_train)
print(pipeline_optimizer.fitted_pipeline_)
```

Después hacemos una predicción con el mejor modelo que haya conseguido el algoritmo de AutoML y evaluamos ese mejor modelo conseguido.

```
print("Mejor Modelo:")
print(pipeline_optimizer.score(X_test, y_test))
```

Después de 10 minutos, conseguimos una tasa de Accuracy del 98 % y en la predicción de casi un 99 %. También vemos el mejor modelo conseguido en este caso con el algoritmo de clasificación k-vecinos con el número de vecinos en 2 y el peso de recorrido utilizado es la distancia (Ver figura 4.15).

```
Best pipeline: KNeighborsClassifier(input_matrix, n_neighbors=2, p=2, weights=distance)
Pipeline(steps=[('kneighborsclassifier',
                 KNeighborsClassifier(n_neighbors=2, weights='distance'))])
TEST:
0.9888888888888889
```

Figura 4.15: Resultados TPOT

4.3. Experimentación final

Para esta experimentación más profunda utilizaremos el mismo dataset en las diferentes herramientas, por lo tanto utilizaremos el dataset muy conocido IRIS.

IRIS

El conjunto de datos de Iris contiene cuatro características (largo y ancho de sépalos y pétalos) de 50 muestras de tres especies de Iris (Iris setosa, Iris virginica e Iris versicolor). Estas medidas se utilizaron para crear un modelo discriminante lineal para clasificar las especies. El conjunto de datos se utiliza a menudo en ejemplos de minería de datos, clasificación y agrupación y para probar algoritmos de Machine Learning. Podemos ver un ejemplo de los 3 distintas clases (Ver figura 4.16).

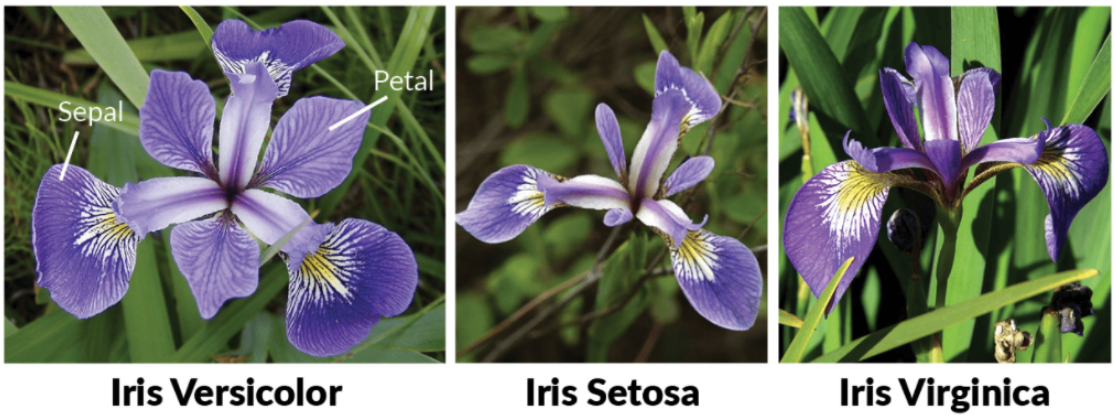


Figura 4.16: Ejemplo de especies de IRIS

AutoWeka

Primero probaremos con la herramienta AutoWeka, utilizaremos los hiperparámetros por defecto del software y solo cambiaremos la semilla que se utiliza para

separar los datos de entrenamiento y de prueba. Después de probar 20 minutos y probar 516 configuraciones nos muestra el mejor modelo, con un 96 % de Accuracy y solo 6 errores de 150 instancias.

```
Auto-WEKA result:
best classifier: weka.classifiers.functions.Logistic
arguments: [-R, 8.454506561093721E-7]
attribute search: null
attribute search arguments: []
attribute evaluation: null
attribute evaluation arguments: []
metric: errorRate
estimated errorRate: 0.013333333333333334
training time on evaluation dataset: 0.026 seconds

You can use the chosen classifier in your own code as follows:

Classifier classifier = AbstractClassifier.forName("weka.classifiers.functions.Logistic", new String[]{"-R", "8.454506561093721E-7"});
classifier.buildClassifier(instances);


Correctly Classified Instances         144               96      %
Incorrectly Classified Instances         6                4      %
Kappa statistic                          0.94
Mean absolute error                      0.0289
Root mean squared error                  0.1244
Relative absolute error                  6.4967 %
Root relative squared error             26.3808 %
Total Number of Instances              150

=== Confusion Matrix ===

  a  b  c   <-- classified as
 50  0  0 |  a = Iris-setosa
  0 47  3 |  b = Iris-versicolor
  0  3 47 |  c = Iris-virginica

=== Detailed Accuracy By Class ===

                 TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
                 1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     Iris-setosa
                 0,940    0,030    0,940      0,940    0,940      0,910    0,997     0,994     Iris-versicolor
                 0,940    0,030    0,940      0,940    0,940      0,910    0,997     0,994     Iris-virginica
Weighted Avg.    0,960    0,020    0,960      0,960    0,960      0,940    0,998     0,996
```

Figura 4.17: Resultados AutoWeka

AutoKeras

Ahora probamos la herramienta AutoKeras, Primero importamos el dataset iris y la librería train_test_split para para la división de los datos de este dataset.Y por último importamos Pandas que es una extensión de NumPy para el tratamiento de datos.

```
1 from sklearn.datasets import load_iris
2 from sklearn.model_selection import train_test_split
3 import pandas as pd
```

Dividimos el dataset en 20 % para test y el resto para el entrenamiento.

```
1 data = load_iris()
2 X, y = data['data'], pd.get_dummies(data['target']).values
3 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

Pasamos al proceso de búsqueda de modelo de clasificación, dado que no es una clasificación por imágenes podemos permitirnos mas intentos sin gastar tanto tiempo de computo, lo establecemos en 50.

```
model = ak.StructuredDataClassifier(overwrite=True, max_trials=50)
model.fit(X_train, y_train)
```

Ahora la etapa de predicciones y comprobamos como se comporta el mejor modelo encontrado.

```
print("TEST")
loss, acc = model.evaluate(X_test, y_test)
print("Accuracy: ", acc)
y_predictions = model.predict(X_test)
```

Y por último vemos un resumen del mejor modelo encontrado y su arquitectura neuronal.

```
modelo_final = model.export_model()
modelo_final.summary()
tf.keras.utils.plot_model(modelo_final,show_shapes=True)
```

Ejecutamos este código y en unos 50 minutos conseguimos nuestro modelo con una Accuracy del 96 % y con unas predicciones del 93 % (Ver figura 4.18).

```
Trial 100 Complete [00h 00m 15s]
val_accuracy: 0.9166666865348816

Best val_accuracy So Far: 0.9583333134651184
Total elapsed time: 00h 50m 50s
```

Figura 4.18: Resultados AutoKeras

Y podemos ver el grafo de la arquitectura neuronal del mejor modelo encontrado en 100 intentos en el apéndice 2 (Ver apéndice B.2).

También podemos contemplar la matriz de confusión(Ver figura 4.19).

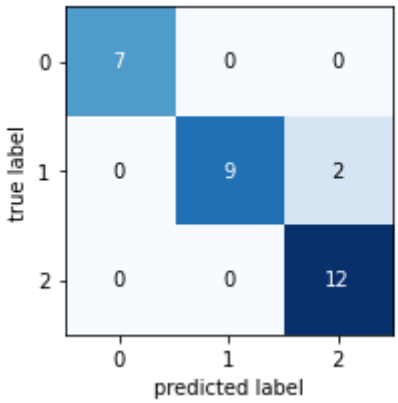


Figura 4.19: Matriz de confusión del modelo de AutoKeras

AutoSklearn

La siguiente herramienta a probar es AutoSklearn, Primero importamos el dataset iris y la librería train_test_split para para la división de los datos de este dataset.

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
```

Dividimos el dataset en 20 % para predicciones y el resto para el entrenamiento.

```
X,y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size = 0.2)
```

Pasamos al proceso de búsqueda de modelo de clasificación, Ponemos que dure un máximo de 5 minutos en la búsqueda y que cada método que utilice dure unos 10 segundos como máximo.

```
modelo = autosklearn.classification.AutoSklearnClassifier
        (time_left_for_this_task=300, per_run_time_limit=10)
modelo.fit(X_train, y_train)
print(modelo.sprint_statistics())
```

Ahora la etapa de predicciones y comprobamos como se comporta el mejor modelo encontrado.

```
y_hat = modelo.predict(X_test)
print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))
```

Después de 5 minutos conseguimos un modelo con una Accuracy del 100 % y un total de 68 algoritmos probados. La predicción en cambio a tenido un 96 % de Accuracy (Ver figura 4.20).

```
auto-sklearn results:
  Dataset name: 383260d2-03f7-11ec-80f1-0242ac1c0002
  Metric: accuracy
  Best validation score: 1.000000
  Number of target algorithm runs: 68
  Number of successful target algorithm runs: 67
  Number of crashed target algorithm runs: 0
  Number of target algorithms that exceeded the time limit: 1
  Number of target algorithms that exceeded the memory limit: 0

Accuracy score 0.9
```

Figura 4.20: Resultados AutoSklearn

Por último generamos la matriz de confusión para tener un grafismo de las predicciones.

```
from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
ypred = modelo.predict(X_test)
```

```
matriz = confusion_matrix(y_test,ypred)
plot_confusion_matrix(conf_mat=matriz, figsize=(3,3), show_normed=False)
plt.tight_layout()
```

Se genera la matriz de confusión y vemos el único error en 30 pruebas (Ver figura 4.21).

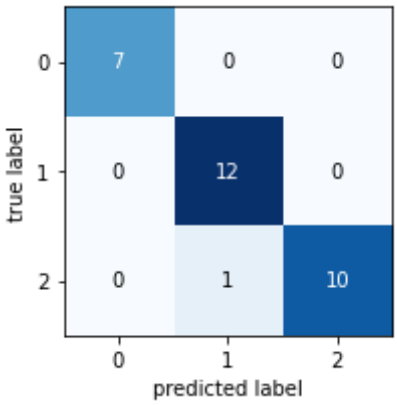


Figura 4.21: Matriz de Confusión del Modelo de AutoSklearn

TPOT

Y por último probamos con la herramienta TPOT, primero importamos el dataset iris y la librería NumPy para el tratamiento de los datos.

```
from sklearn.datasets import load_iris
import numpy as np
```

Dividimos el dataset en 25 % para predicciones y el resto para el entrenamiento, con una semilla para la pseudo aleatoriedad de la separación de datos.

```
iris = load_iris()
X_train, X_test, y_train, y_test = train_test_split
    (iris.data.astype(np.float64),iris.target.astype(np.float64),
        train_size=0.75, test_size=0.25, random_state=42)
```

Pasamos al proceso de búsqueda de modelo de clasificación, los hiperparámetros los modificamos para que se generen 10 generaciones, que son las iteraciones en TPOT, el population size en 50, que es el numero de invididuales en cada generación en la programación genética que utiliza TPOT, y por último la semilla para la pseudo aleatoriedad del proceso. El verbosity para que muestre por pantalla el proceso.

```
tpot = TPOTClassifier(generations=20, population_size=50,
                      verbosity=2, random_state=42)
tpot.fit(X_train, y_train)
print(tpot.fitted_pipeline_)
```

Ahora la etapa de predicciones y comprobamos como se comporta el mejor modelo encontrado.

```
print("TEST:")
print(tpot.score(X_test, y_test))
tpot.export('tpot_iris_pipeline.py')
```

Después de unos 10 minutos conseguimos una Accuracy de un 98 % y una Accuracy en las predicciones del 100 %. El mejor modelo conseguido usa el algoritmo XGB para la clasificación (Ver figura 4.22).

```
Pipeline(steps=[('stackingestimator',
                 StackingEstimator(estimator=XGBClassifier(base_score=0.5,
                                                           booster='gbtree',
                                                           colsample_bylevel=1,
                                                           colsample_bynode=1,
                                                           colsample_bytree=1,
                                                           gamma=0, gpu_id=-1,
                                                           importance_type='gain',
                                                           interaction_constraints='',
                                                           learning_rate=1.0,
                                                           max_delta_step=0,
                                                           max_depth=6,
                                                           min_child_weight=20,
                                                           missing=nan,
                                                           monotone_constraints='()',
                                                           n_estimators=100,
                                                           n_jobs=1,
                                                           num_parallel_tree=1,
                                                           objective='multi:softprob',
                                                           random_state=42,
                                                           reg_alpha=0,
                                                           reg_lambda=1,
                                                           scale_pos_weight=None,
                                                           subsample=0.1,
                                                           tree_method='exact',
                                                           validate_parameters=1,
                                                           verbosity=0))),
                 ('multinomialnb', MultinomialNB(alpha=0.01))])
TEST:
1.0
```

Figura 4.22: Resultados TPOT

Por último generamos la matriz de confusión para tener un grafismo de las predicciones. Y comprobamos los resultados (Ver figura 4.23).

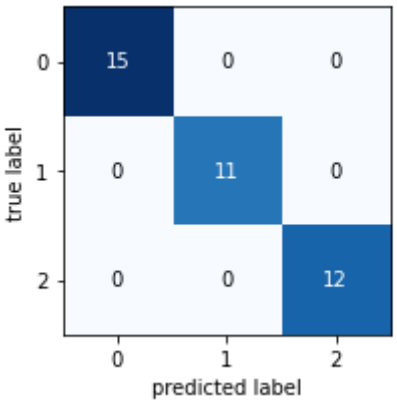


Figura 4.23: Matriz de Confusión del Modelo de TPOT

4.4. Análisis de los resultados

Después de el uso y experimentación con estas herramientas haremos una comparativa no solo de resultados sino también de ventajas y desventajas de cada herramienta.

4.4.1. Comparación de resultados obtenidos

Primero compararemos los resultados obtenidos en el experimento preliminar (Ver tabla 4.1).

Herramientas	***Accuracy Entrenamiento***	***Accuracy Test***	***Tiempo empleado***	***Dataset***
AutoWeka	98 %	98 %	15 min	IRIS-2D
AutoKeras	96 %	98 %	23 min	MNIST
AutoSklearn	99 %	98 %	40 min	Load Digits
TPOT	98 %	98 %	10 min	Load Digits

Tabla 4.1: Tabla de Comparación de Resultados del experimento Preliminar

Visto esta tabla podemos contemplar que todas las herramientas han conseguido un modelo con alto porcentaje de Accuracy tanto en el entrenamiento como en la prueba. Podemos contemplar que el que menos porcentaje ha conseguido en el entrenamiento es la herramienta AutoKeras pero esto se debe a que su dataset no tenia repartida equitativamente los datos de los distintos dígitos por lo tanto la Accuracy no es tan efectiva. Podemos también ver que AutoSklearn con más tiempo de computo ha encontrado un modelo con el 99 % de Accuracy, pero TPOT con un cuarto del tiempo ha conseguido un modelo muy cercano.

Ahora compararemos los resultado obtenidos en el experimento sobre el dataset IRIS (Ver tabla 4.2).

Herramientas	*Accuracy Entrenamiento*	*Accuracy Test*	*Tiempo empleado*
AutoWeka	96 %	96 %	20 min
AutoKeras	96 %	93 %	50 min
AutoSklearn	100 %	96 %	5 min
TPOT	98 %	100 %	10 min

Tabla 4.2: Tabla de Comparación de Resultados del experimento sobre IRIS

También podemos contemplar una gráfica comparativa de los tiempos empleados en la experimentación (Ver figura 4.24).

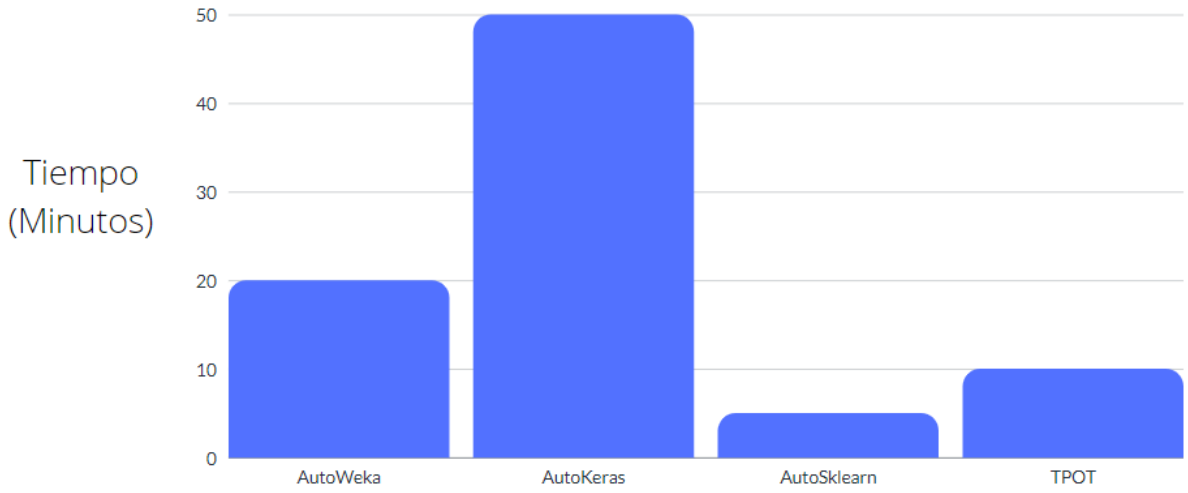


Figura 4.24: Gráfica comparativa del Tiempo de Ejecución

Podemos contemplar como Autoweka tiene un rendimiento bastante estable en el mismo tiempo de ejecución, Autokeras iguala el rendimiento pero con casi el doble de tiempo de ejecución. Con este dataset los modelos que se utilizan en AutoSklearn han conseguido una Accuracy del 100 % en el entrenamiento pero solo un 90 % en las pruebas, por lo tanto no es tan fiable como se esperaba al principio. Por último con TPOT se han conseguido unos números muy buenos en tan solo 10 minutos de búsqueda, con un 98 % en el entrenamiento y un 100 % en las pruebas.

Con las matrices de confusión podemos contemplar con un primer vistazo donde suele fallar nuestros modelos. Suelen fallar con las predicciones de los tipos versicolor y virginica, suele fallar entre predecir entre estos dos tipos, en cambio ningún fallo en predecir las de tipo setosa.

4.4.2. Opinión de las herramientas

Ahora haremos una comparativa de las utilidades después de el uso de estas de herramientas viendo sus pros y contras que tienen para su uso (Ver tabla 4.3).

Herramientas	*Pros*	*Cons*
AutoWeka	- Buena Documentación - Fácil de Usar e Instalar - Interfaz de Usuario - Visualización de los Datos	- Poca Personalización - Poca Versatilidad - Sin Actualizaciones desde 2017 - Requiere un software y Java
AutoKeras	- Muy Versátil - Grafismos del Modelo - Buena Documentación - Fácil de Instalar y Usar	- Poca modificación de Hiperparámetros - Sin Control de Tiempo - Muy Lento
AutoSklearn	- Muy Rápido - Buenos Modelos - Buena Documentación - Fácil de Instalar y Usar	- Problemas con las versiones - Documentación Pobre - Mala Pseudo aleatoriedad
TPOT	- Muy Consistente - Buena Documentación - Buen Funcionamiento	- Lento - Instalación Compleja

Tabla 4.3: Tabla de comparación de funcionamiento de las Herramientas

Visto todo estas ventajas y desventajas que tienen las herramientas seleccionadas vemos que todas tienen su parte buena y mala, AutoWeka es muy fácil de usar dado la interfaz gráfica de usuario que posee pero tiene pocas opciones de customización, Autokeras en cambio es muy fácil y rápido de usar y con muchas opciones de AutoML, pero con mucho tiempo de ejecución en sus búsquedas. AutoSklearn es rápido y fácil de usar, pero tiene problemas con sus versiones de las dependencias y no es muy transparente con su búsqueda del modelo ni es consistente en si misma. Y por último, TPOT es una muy buena herramienta pero con la instalación más compleja de estas herramientas y muy lento en la búsqueda de los modelos dependiendo del dataset.

Capítulo 5

Conclusiones y líneas futuras

En este capitulo se presentaran las conclusiones obtenidas tras el desarrollo del proyecto de introducción y análisis de las principales tecnologías de AutoML y sus respectivas lineas futuras.

Conclusiones

Tras finalizar el estudio y la experimentación se ha llegado a las siguientes conclusiones:

1. El AutoML es una rama de la inteligencia artificial que ha llegado para quedarse, ofrece una experiencia más simplificada en el desarrollo de modelos de Machine Learning, por lo tanto no hace tan dura la iniciación en la inteligencia artificial.

2. Se consiguen buenos resultados con el uso de herramientas AutoML, con el uso de distintas herramientas y de distintas técnicas se consiguen modelos competitivos en poco tiempo.

3. Las herramientas de AutoML tienen sus ventajas y desventajas, por ejemplo, las herramientas funcionan muy distintas entre sí, algunas no son muy intuitivas con los hiperparámetros, necesitan más tiempo de computo y más espacio de memoria para su ejecución a gran escala. Pero simplifican mucho el trabajo de entrenamiento, facilitan el preprocesamiento de los datos, son mas eficaces en casos de bajo y medio volumen, y no se necesita tanta experiencia en el campo.

4. Los conocimientos tanto de AutoML y Machine Learning adquiridos, usando y estudiando herramientas de AutoML se consiguen muchos conocimientos de Machine Learning que se pueden extrapolar para el uso de herramientas y desarrollo de modelos de Machine Learning. Tanto como las técnicas de evaluación de modelos como los algoritmos utilizados, como en el tratamiento de datos. En resumen, se adquieren conocimientos de todo el proceso de desarrollo de un modelo.

5. La necesidad de potenciar el uso de herramientas open source, y lograr que estas herramientas sean competentes y ofrezcan los mismos resultados y las mismas posibilidades que las herramientas comerciales.

Líneas de trabajo futuro

Entre las principales líneas futuras derivadas de este trabajo se tendrían:

1. Realizar un mayor número de experimentaciones de las distintas herramientas. Seleccionar diferentes datasets de distinto tipo y también incluir un ejemplo de regresión para comprobar el comportamiento de las herramientas.

2. Realizar gráficas de resultados comparativas. Con herramientas de terceros se pueden realizar unas gráficas comparativas de los modelos obtenidos.

3. Integrar más herramientas a la experimentación. Las herramientas seleccionadas son de las más conocidas pero hay más, como por ejemplo H20, sin hablar de las plataformas comerciales de Google o Amazon.

4. Lograr la integración de las distintas herramientas. Conseguir que se haga más popular su uso para desarrollar modelos de Machine Learning.

5. Divulgación de las herramientas del AutoML en el grado y posgrado. Proponer que se dedique un tiempo, tanto en el grado de Ingeniería Informática como en sus respectivos posgrados, a dar visibilidad a este campo de la inteligencia artificial.

Capítulo 6

Summary and Conclusions

In this chapter the conclusions obtained after the development of the project for the introduction and analysis of the main AutoML technologies and their respective future lines will be presented.

Conclusions

After completing the study and experimentation, the following conclusions have been reached:

1. AutoML is a branch of artificial intelligence that has come to stay, it offers a more simplified experience in the development of Machine Learning models, therefore it does not make the initiation in artificial intelligence so hard.

2. Good results are achieved with the use of AutoML tools, with the use of different tools and different techniques, competitive models are achieved in a short time.

3. AutoML tools have their advantages and disadvantages, for example, the tools work very differently from each other, some are not very intuitive with hyperparameters, they require more compute time and more memory space to run on a large scale. But they greatly simplify training work, facilitate data preprocessing, are most effective in low and medium volume cases, and not as much experience in the field is needed.

4. The knowledge of both AutoML and Machine Learning acquired, using and studying AutoML tools, a lot of Machine Learning knowledge is obtained that can be extrapolated for the use of tools and development of Machine Learning models. As well as the techniques for evaluating models and the algorithms used, as well as in data processing. In short, you acquire knowledge of the entire process of developing a model.

5. The need to promote the use of open source tools, and ensure that these tools are competent and offer the same results and the same possibilities as commercial tools.

Future Improvements

Among the main future lines derived from this work would be:

1. Carry out a greater number of experimentations with the different tools. Select different datasets of different types and also include a regression example to check the behavior of the tools.

2. Make graphs of comparative results. With third-party tools, comparative graphs of the obtained models can be made.

3. Integrate more tools to experimentation. The selected tools are among the best known but there are more, such as H20, not to mention the commercial platforms of Google or Amazon.

4. Achieve the integration of the different tools. To make its use more popular to develop Machine Learning models.

5. Dissemination of AutoML tools in graduate and postgraduate degrees. Propose that time be dedicated, both in the Computer Engineering degree and in their respective postgraduate degrees, to give visibility to this field of artificial intelligence.

Capítulo 7

Presupuesto

En este capítulo se presenta una estimación sobre el coste de la realización de este proyecto. También se presenta un coste hipotético de la implementación de este modelo en un entorno real.

7.1. Presupuesto de recursos humanos

En este apartado se muestra de forma desglosada el coste de los recursos humanos que ha sido invertido en el desarrollo del proyecto (Ver tabla 7.1).

Tarea	***Cantidad de horas empleadas***	***Coste bruto por hora***	***Coste total***
Estudios relacionados al Machine Learning/Deep Learning	15	14€	210€
Estudio relacionados al AutoML	20	14€	240€
Búsqueda de Herramientas AutoML	5	14€	70€
Aprendizaje de las diferentes librerías y herramientas a usar en la experimentación	25	14€	350€
Experimentación con las diferentes herramientas	40	14€	560€
Total:	**166 horas**		**1430€**

Tabla 7.1: Presupuesto Personal

7.2. Presupuesto material

En este apartado se calculan los costes de los distintos componentes necesarios para el desarrollo de la aplicación (Ver tabla 7.2).

Elemento	*Coste total*
Portátil	1000€
Total:	**1000€**

Tabla 7.2: Presupuesto de los materiales necesarios

7.3. Presupuesto final

El presupuesto final teniendo en cuenta el presupuesto del personal humano y utilizando el presupuesto de los materiales necesarios para la realización del modelo, sería el siguiente (Ver tabla 7.3).

Elemento	*Coste total*
Coste humano	1430€
Coste de los materiales	1000€
Total:	**2430€**

Tabla 7.3: Presupuesto Final

Apéndice A

Cuadernos Jupyter de la Experimentación

A.1. Cuaderno Jupyter con el Código de AutoKeras

```
/**************************************************************************
*
* Fichero .ipynb
*
**************************************************************************
*
* Chesen Castilla Gil
*
* Agosto 2021
*
* Cuaderno Experimentación de AutoKeras
*
**************************************************************************/
!pip install git+https://github.com/keras-team/keras-tuner.git
!pip install autokeras
import numpy as np
import tensorflow as tf
from tensorflow.keras.datasets import mnist
import matplotlib.pyplot as plt
import autokeras as ak

# Muestra del Dataset de MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()
num = 4
images = x_train[:num]
labels = y_train[:num]
num_row = 2
num_col = 2
fig, axes = plt.subplots
    (num_row, num_col, figsize=(1.5*num_col,2*num_row))
for i in range(num):
    ax = axes[i//num_col, i%num_col]
```

```
33     ax.imshow(images[i], cmap='gray')
34     ax.set_title('Label: {}'.format(labels[i]))
35 plt.tight_layout()
36 plt.show()
37
38
39 # Experimentación preliminar
40 clf = ak.ImageClassifier(overwrite=True, max_trials=1)
41 clf.fit(x_train, y_train, epochs=10)
42
43 predicted_y = clf.predict(x_test)
44 print(predicted_y)
45 print(clf.evaluate(x_test, y_test))
46
47 modelo_final = clf.export_model()
48 modelo_final.summary()
49 tf.keras.utils.plot_model(modelo_final,show_shapes=True)
50
51 # Experimentación con IRIS
52 from sklearn.datasets import load_iris
53 from sklearn.model_selection import train_test_split
54 import pandas as pd
55
56 data = load_iris()
57 X, y = data['data'], pd.get_dummies(data['target']).values
58 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
59
60 model = ak.StructuredDataClassifier(overwrite=True)
61 model.fit(X_train, y_train)
62
63 print("TEST")
64 loss, acc = model.evaluate(X_test, y_test)
65 print("Accuracy: ", acc)
66 y_predictions = model.predict(X_test)
67 modelo_final = model.export_model()
68 modelo_final.summary()
69 tf.keras.utils.plot_model(modelo_final,show_shapes=True)
70
71 # Matriz de confusión
72 from sklearn.metrics import  confusion_matrix
73 from mlxtend.plotting import plot_confusion_matrix
74 matrix = confusion_matrix(y_test.argmax(axis=1), y_predictions.argmax(axis=1))
75 plot_confusion_matrix(conf_mat=matrix, figsize=(3,3), show_normed=False)
76 plt.tight_layout()
```

A.2. Cuaderno Jupyter con el Código de AutoSklearn

```
/**************************************************************************
*
* Fichero .ipynb
*
**************************************************************************
*
* Chesen Castilla Gil
*
* Agosto 2021
*
* Cuaderno Experimentación de AutoSklearn
*
**************************************************************************/
!curl https://raw.githubusercontent.com/automl/auto-sklearn/master
        /requirements.txt | xargs -n 1 -L 1 pip3 install
!pip3 install auto-sklearn
import autosklearn.classification
import sklearn.model_selection
import sklearn.datasets
import sklearn.metrics
import matplotlib.pyplot as plt

# Muestra del dataset de Load Digits
digits = sklearn.datasets.load_digits()
plt.gray()
plt.matshow(digits.images[0])
plt.show()

# Experimentación preliminar
X, y = sklearn.datasets.load_digits(return_X_y=True)
X_train, X_test, y_train, y_test =
    sklearn.model_selection.train_test_split(X, y, random_state=1)

automl = autosklearn.classification.AutoSklearnClassifier()
automl.fit(X_train, y_train)
print(automl.sprint_statistics())

y_hat = automl.predict(X_test)
print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))

# Experimentación con IRIS
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

X,y = load_iris(return_X_y=True)
```

```
X_train, X_test, y_train, y_test =
        train_test_split(X, y, test_size = 0.2)

modelo = autosklearn.classification.AutoSklearnClassifier
        (time_left_for_this_task=300,per_run_time_limit=10)
modelo.fit(X_train, y_train)
print(modelo.sprint_statistics())

y_hat = modelo.predict(X_test)
print("Accuracy score", sklearn.metrics.accuracy_score(y_test, y_hat))

# Matriz de Confusión

from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix

ypred = modelo.predict(X_test)
matriz = confusion_matrix(y_test,ypred)

plot_confusion_matrix(conf_mat=matriz, figsize=(3,3), show_normed=False)
plt.tight_layout()
```

A.3. Cuaderno Jupyter con el Código de TPOT

```
/**************************************************************************
*
* Fichero .ipynb
*
**************************************************************************
*
* Chesen Castilla Gil
*
* Agosto 2021
*
* Cuaderno Experimentación de TPOT
*
**************************************************************************/
!pip install deap update_checker tqdm stopit xgboost
!pip install dask[delayed] dask[dataframe]
        dask-ml fsspec>=0.3.3 distributed>=2.10.0
!pip install scikit-mdr skrebate
!pip install tpot

from tpot import TPOTClassifier
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_digits
```

```
23
24 # Muestra del dataset de Load Digits
25 digits = load_digits()
26 print(digits.data.shape)
27 import matplotlib.pyplot as plt
28 plt.gray()
29 plt.matshow(digits.images[0])
30 plt.show()
31
32 # Experimentación Preliminar
33 digits = load_digits()
34 X_train, X_test, y_train, y_test = train_test_split
35         (digits.data, digits.target,train_size=0.75, test_size=0.25)
36
37 pipeline_optimizer = TPOTClassifier
38         (generations=5, population_size=20,random_state=42, verbosity=2)
39 pipeline_optimizer.fit(X_train, y_train)
40 print(pipeline_optimizer.fitted_pipeline_)
41
42 print("TEST:")
43 print(pipeline_optimizer.score(X_test, y_test))
44 pipeline_optimizer.export('tpot_exported_pipeline.py')
45
46 # Experimentación con IRIS
47 from sklearn.datasets import load_iris
48 import numpy as np
49
50 iris = load_iris()
51 X_train, X_test, y_train, y_test =train_test_split
52     (iris.data.astype(np.float64),iris.target.astype(np.float64),
53     train_size=0.75, test_size=0.25, random_state=42)
54
55 tpot = TPOTClassifier
56         (generations=20, population_size=50, verbosity=2, random_state=42)
57 tpot.fit(X_train, y_train)
58 print(tpot.fitted_pipeline_)
59
60 print("TEST:")
61 print(tpot.score(X_test, y_test))
62 tpot.export('tpot_iris_pipeline.py')
63
64 # Matriz de confusión
65 from mlxtend.plotting import plot_confusion_matrix
66 from sklearn.metrics import confusion_matrix
67
68 ypred = tpot.predict(X_test)
69 matriz = confusion_matrix(y_test,ypred)
```

```
70
71 plot_confusion_matrix(conf_mat=matriz, figsize=(3,3), show_normed=False)
72 plt.tight_layout()
```

Apéndice B

Grafos de arquitectura neuronal

B.1. Grafo de arquitectura neuronal de la experimentación preliminar

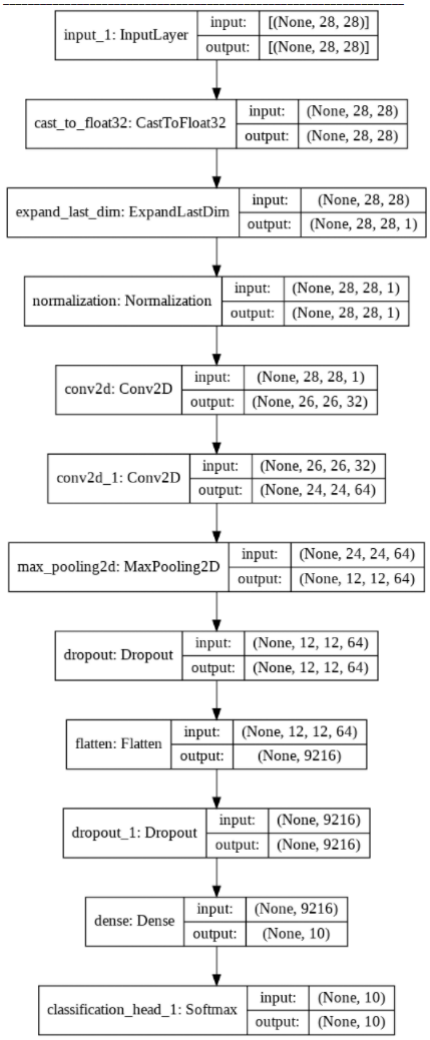


Figura B.1: Grafo Arquitectura Neuronal del mejor modelo de AutoKeras

B.2. Grafo de arquitectura neuronal de la experimentación final

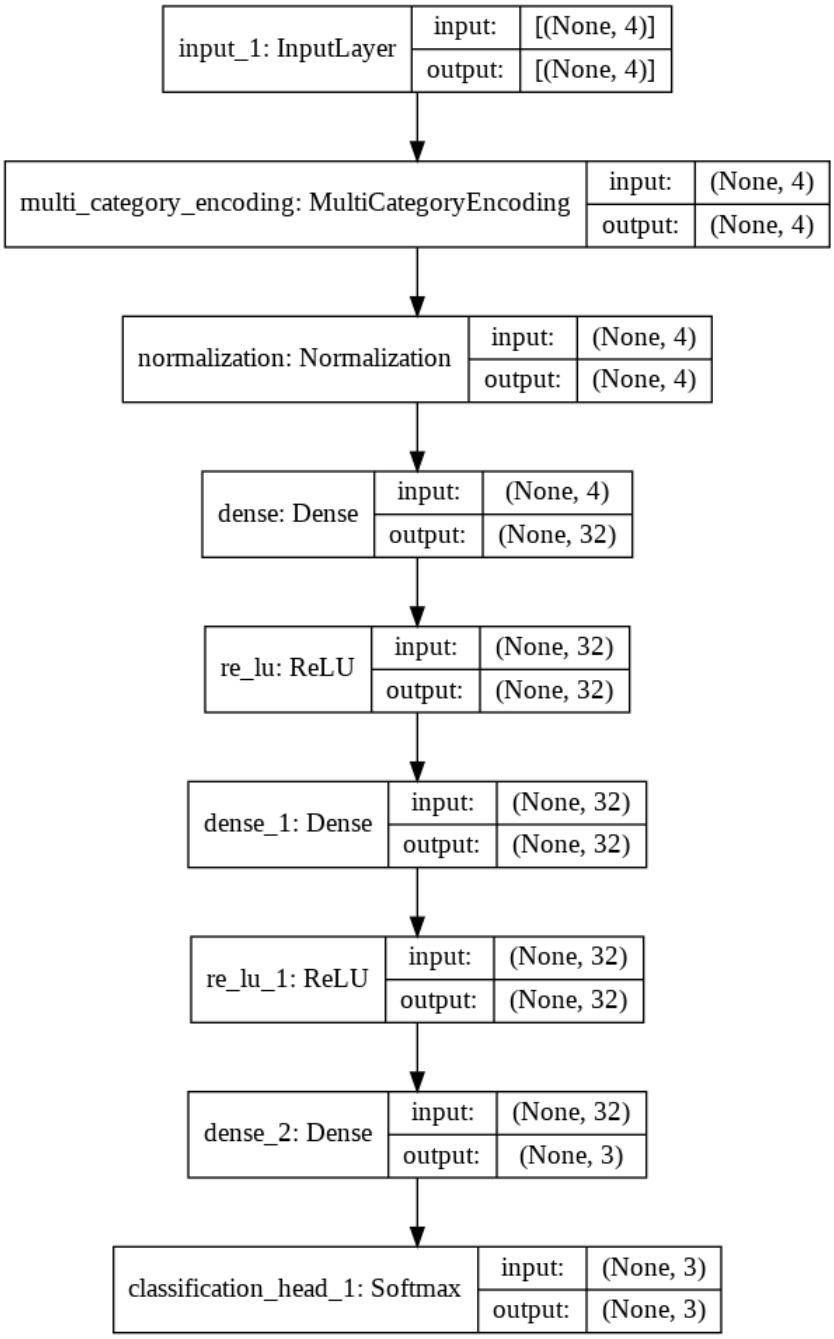


Figura B.2: Grafo Arquitectura Neuronal del mejor modelo de AutoKeras

Bibliografía

[1] CMA ALex Blyakhman, CIA CSCA, et al. The new era of automl. *Strategic Finance*, 102(8):60–61, 2021.

[2] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *The Journal of Machine Learning Research*, 20(1):1997–2017, 2019.

[3] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. Auto-sklearn 2.0. *arXiv:2007.04074 [cs.LG]*, 2020.

[4] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[5] Pieter Gijsbers, Erin LeDell, Janek Thomas, Sébastien Poirier, Bernd Bischl, and Joaquin Vanschoren. An open source automl benchmark. *arXiv preprint arXiv:1907.00909*, 2019.

[6] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, et al. Analysis of the automl challenge series. *Automated Machine Learning*, page 177, 2019.

[7] Xin He, Kaiyong Zhao, and Xiaowen Chu. Automl: A survey of the state-of-the-art. *Knowledge-Based Systems*, 212:106622, 2021.

[8] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[9] Haifeng Jin, Qingquan Song, and Xia Hu. Auto-keras: An efficient neural architecture search system. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1946–1956. ACM, 2019.

[10] Yusuf Kirikkayis. The automl jungle-an overview. In *Proceedings of the 2020 OMI Seminars (PROMIS 2020)*, volume 1, pages 13–1. Universität Ulm, 2021.

[11] Trang T Le, Weixuan Fu, and Jason H Moore. Scaling tree-based automated machine learning to biomedical big data with a feature set selector. *Bioinformatics*, 36(1):250–256, 2020.

[12] Thiloshon Nagarajah and Guhanathan Poravi. An extensive checklist for building automl systems. In *AMIR@ ECIR*, pages 56–70, 2019.

[13] Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 485–492, New York, NY, USA, 2016. ACM.

[14] Randal S. Olson, Ryan J. Urbanowicz, Peter C. Andrews, Nicole A. Lavender, La Creis Kidd, and Jason H. Moore. *Applications of Evolutionary Computation: 19th European Conference, EvoApplications 2016, Porto, Portugal, March 30 – April 1, 2016, Proceedings, Part I*, chapter Automating Biomedical Data Science Through Tree-Based Pipeline Optimization, pages 123–137. Springer International Publishing, 2016.

[15] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. 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*, pages 847–855, 2013.

[16] Chris Thornton, Frank Hutter, Holger H Hoos, Kevin Leyton-Brown, et al. Auto-weka: Automated selection and hyper-parameter optimization of classification algorithms. *CoRR, abs/1208.3719*, 2012.

[17] Ziqiao Weng. From conventional machine learning to automl. In *Journal of Physics: Conference Series*, volume 1207, page 012015. IOP Publishing, 2019.

[18] Doris Xin, Eva Yiwei Wu, Doris Jung-Lin Lee, Niloufar Salehi, and Aditya Parameswaran. Whither automl? understanding the role of automation in machine learning workflows. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–16, 2021.