

Universidad de La Habana
Facultad de Matemática y Computación



Optimización Multiobjetivo para Autogoal

Autor:
Rodrigo Daniel Pino Trueba

Tutores:
Dra. Suilan Estévez Velarde
Lic. Daniel Valdés Pérez

Trabajo de Diploma
presentado en opción al título de
Licenciado en Ciencia de la Computación

Fecha
github.com/rodrigo-pino/Thesis

Dedicación

Agradecimientos

Agradecimientos

Opinión del tutor

Opiniones de los tutores

Resumen

El problema multiobjetivo aplicado a sistemas de Aprendizaje de Máquina Automatizado (AutoML) es un área poco explorada. Este trabajo define el problema de optimización multiobjetivo aplicado a sistemas de AutoML. Propone una solución generalizada para cualquier tipo de sistema aprovechando que sus componentes trabajan en espacios distintos: el espacio de búsqueda donde habitan los flujos de Aprendizaje Automático encontrados y el espacio objetivo donde reside la evaluación de cada uno de estos respecto a diversas métricas. El sistema AutoML genera soluciones sobre el espacio de decisión y el algoritmo multiobjetivo las ordena en el espacio objetivo. La propuesta se inspira en NSGA-II como algoritmo multiobjetivo. Se realiza una implementación en AutoGOAL, un sistema de AutoML orientado a resolver el problema de AutoML Heterógeno. Se realizan pruebas midiendo *f-score* contra tiempo de entrenamiento y precisión contra recobrando obteniendo resultados satisfactorios que prueban la adición de optimización multiobjetivo a sistemas AutoML un beneficio inmediato.

Abstract

Resumen en inglés

Índice general

Introducción	1
1. Estado del Arte	5
1.1. Aprendizaje de Máquina Automatizado	5
1.1.1. Optimización Bayesiana	6
1.1.2. Programación Evolutiva	8
1.1.3. Búsqueda Aleatoria	9
1.1.4. Aprendizaje por Refuerzo	10
1.1.5. Modelos Constructivos	10
1.1.6. Monte Carlo	11
1.2. Optimización Multiobjetivo	11
1.2.1. Técnicas de Escalarización	12
1.2.2. Algoritmos Numéricos	13
1.2.3. Algoritmos Evolucionarios Multiobjetivos	14
1.3. AutoML y Multiobjetivo	15
2. AutoML Heterógeno Multiobjetivo	17
2.1. Descripción General	17
2.1.1. Descripción Formal	18
2.2. MOEA	19
2.2.1. NSGA-II	19
2.2.2. NSGA-II Generalizado	22
3. AutoGOAL Multiobjetivo	23
3.1. Implementación	23
3.1.1. NSPGE	24
3.1.2. Otros cambios	27
3.2. Aplicación de Autogoal Multiobjetivo	28
3.2.1. Métricas	28

4. Experimentación y Resultados	30
4.1. Corpus de Evaluación	30
4.1.1. CARS	31
4.1.2. HAHA	31
4.1.3. MEDDOCAN	32
4.2. Hardware	32
4.3. Resultados y Análisis	32
4.3.1. Cars	32
4.3.2. HAHA	33
4.3.3. MEDDOCAN	36
Conclusiones	40
Recomendaciones	41
Bibliografía	42

Índice de figuras

2.1. Flujo general de la propuesta	18
2.2. Funcionamineto de NSGA-II	21
3.1. Diagrama general	24
4.1. Cars: F-score contra Tiempo de Entrenamiento	33
4.2. Cars: Precisión contra Recobrado	34
4.3. HAHA: F-score contra Tiempo de Entrenamiento (10 segundos máx)	35
4.4. HAHA: F-score contra Tiempo de Entrenamiento (3 minutos máx)	36
4.5. HAHA: Precisión contra Recobrado (10 segundos máx)	36
4.6. HAHA: Precisión contra Recobrado (3 minutos máx)	37
4.7. MEDDOCAN: F-Score contra Tiempo de Entrenamiento	38
4.8. MEDDOCAN: Precisión contra Recobrado	39

Índice de Algoritmos

2.1. Solución a AutoML Heterógeno Multiobjetivo	19
2.2. Crowding Distance Sorting	20

Ejemplos de código

3.1. Nueva ordenación	24
3.2. Non Dominated Sorting	25
3.3. Crowding Distance Sorting	26
3.4. Utilizando NPSGE	28
3.5. Ejemplo de métrica: accuracy contra tiempo	28

Introducción

Vivimos en una sociedad cada vez más digitalizada donde algoritmos de inteligencia artificial se integran progresiva y gradualmente a nuestras vidas. En la medicina para la predicción de enfermedades como el cáncer (Kourou y col. 2015), las noticias recomendadas basadas en nuestros gustos online (Liu y col. 2010), los carros inteligentes que se conducen a si mismos con mínima interacción por parte del piloto (Badue y col. 2021), detección de sentimientos de odio en un texto (Fortuna y Nunes 2018) e incluso en el arte, donde sistemas producen imágenes partiendo de descripciones textuales (Xu y col. 2018).

El Aprendizaje Automático (*Machine Learning* en inglés) es una rama de la inteligencia artificial que compone el núcleo de estas aplicaciones, no obstante, el número de expertos disponibles sobre el tema no es suficiente para la creciente demanda de programas de este tipo. Los modelos o flujos de aprendizaje de máquina requieren para su correcto funcionamiento de un buen diseño determinado por la selección de técnicas de aprendizaje e hiperparámetros que lo componen. La modelación es una ardua tarea que requiere conocimientos, experiencia y abundante prueba y error. Con el objetivo de mitigar esta insuficiencia nace el Aprendizaje de Máquina Automatizado, una línea de investigación dirigida a automatizar esta parte del proceso.

Los programas que resuelven el problema de Aprendizaje Automatizado se le conocen como sistemas AutoML y varían en la parte de trabajo del experto que desean automatizar: desde la selección automática de hiperparámetros (Feurer y Hutter 2019) a la selección de modelos (Thornton y col. 2013) o ambos simultáneamente. El objetivo general de estos sistemas es producir una selección tal que el modelo final maximice lo más posible respecto a cierta métrica o criterio de evaluación. Usualmente esta métrica representa la relevancia de los resultados obtenidos.

En el contexto actual, existen problemas para los cuales ya no es suficiente que un modelo de Aprendizaje Automático obtenga resultados relevantes pues se requiere además un buen desempeño respecto a criterios adicionales. En el campo del Aprendizaje de Máquina Automatizado esto implica la necesidad de incrementar la expresividad para describir los problemas con tal de obtener mayor control sobre las soluciones que estos generan. Entre las propuestas para lograr esto se encuentra la adición de optimización simultánea de múltiples criterios de evaluación a los sistemas AutoML

tal que los modelos de Aprendizaje Automático producidos tengan buen rendimiento en estos.

Motivación

Un problema conocido y estudiado es el sesgo en los flujos de Aprendizaje de Máquina. Existen casos donde ciertas características de los datos pueden provocar que las predicciones de un modelo se encuentren parcializadas en favor de una mayoría (Mehrabi y col. 2021) afectando su capacidad de predicción. Esto tiene mayor importancia cuando los datos representan seres humanos y la decisión del modelo tiene un impacto directo o indirecto en la vida de un individuo o grupo. Se desea que el algoritmo no discrimine por el color de piel, género o región y a la vez sus resultados se mantengan relevantes. Es necesario entonces que el flujo sea capaz de poder optimizar tanto relevancia como nivel de justicia, un problema multiobjetivo.

También hay situaciones donde se desea obtener un modelo que sea relevante a la par de *interpretable*, que implica entender sin necesidad de ser un experto porque el flujo toma ciertas decisiones; o cuan *robusto* es, tal que perturbaciones en los datos de entrada no cambien radicalmente el resultado de este.

La adición de optimización multiboojetivo en AutoML abre las puertas también a métricas que no tienen sentido usarlas individualmente como *precisión* y *recobrado*. *Precisión* mide la proporción entre los valores relevantes identificados y todos los valores identificados mientras que *recobrado* mide la proporción entre los valores relevantes identificados y todos los valores relevantes; métricas que usadas por separado no son representativas pues basta para tener un recobrado perfecto seleccionar todos los elementos del conjunto de datos, y una precisión casi perfecta escogiendo un pequeño conjunto de elementos. *F-score* relaciona ambas medidas y permite el uso de parámetros para priorizar una o la otra según el investigador determine. Un sistema AutoML Multiobjetivo puede prescindir de métricas como *f-score* pues es capaz de optimizar en paralelo para todos los criterios que la componen.

Antecedentes

En el entorno del grupo de Inteligencia Artificial de la facultad de Matemáticas y Computación de la Universidad de La Habana se desarrolla diversas líneas de investigación centradas en el Aprendizaje de Máquina Automatizado. AutoGOAL (Estevez-Velarde y col. 2020) es una herramienta creada por el colectivo con el objetivo de resolver el problema AutoML heterogéneo.

La entidad también desarrolla otras líneas donde se estudia la posible mitigación de sesgos en flujos producidos por sistemas AutoML (Consuegra-Ayala y col. 2022).

Este trabajo es una propuesta que pretende mejorar la expresividad que tienen los usuarios para la búsqueda de sus soluciones permitiendo la utilización de varias métricas.

Problemática

Resolver el problema multiobjetivo en AutoML es un tema poco estudiado y con aplicaciones prácticas en la actualidad. Los sistemas AutoML de mayor reconocimiento en el campo solo son capaces de optimizar para una sola métrica.

Los sistemas de AutoML utilizan espacios de búsqueda caóticos y no diferenciables para los cuales se necesitan algoritmos agnósticos a esos espacios y resistente a formas extrañas del frente de Pareto. Es necesario definir un algoritmo evolutivo multiobjetivo lo suficientemente general que resuelva el problema multiobjetivo y sea compatible con cualquier sistema de AutoML.

Objetivo

Objetivo General

Optimización Multiobjetivo para sistemas AutoML

Objetivos Específicos

- Estudiar el estado del arte sobre sistemas de Aprendizaje de Máquina Automatizado
- Estudiar conceptos y principios de la optimización multobjetivo y sus aplicaciones en el campo del Aprendizaje de Máquina Automatizado
- Identificar e implementar un algoritmo multiobjetivo que aproveche la estructura de AutoGOAL y su implementación de Probabilistic Grammatic Evolution (PGE)
- Experimentar y analizar los resultados de aplicar AutoGOAL Multiobjetivo a problemas conocidos de la literatura.

Estructura de la Tesis

El resto del documento se encuentra organizado de la siguiente manera. En el capítulo 1 se estudia técnicas de AutoML y optimización multobjetivo que conforman el

estado del arte, así como ejemplos de sistemas AutoML que apliquen técnicas de optimización multiobjetivo. En el capítulo 2 se desarrolla una propuesta generalizada para resolver el problema de AutoML Heterogéneo utilizando optimización multobjetivo. El capítulo 3 muestra una implementación de la propuesta utilizando AutoGOAL. En el capítulo 4 se muestran resultados obtenidos tras aplicar la propuesta en tres corpus de datos distintos utilizando optimización multiboojetivo para dos pares de métricas. Finalmente, en el capítulo 5 se presentan las conclusiones de la investigación.

Capítulo 1

Estado del Arte

El proceso de investigación para diseñar un flujo de Aprendizaje Automático requiere una experimentación variada donde se combinan múltiples técnicas como redes neuronales, clasificadores supervisados, algoritmos de agrupamientos, entre otras. Además por cada selección de las anteriores se deben optimizar adecuadamente los hiperparámetros que la componen.

Con el uso de un sistema AutoML el investigador ya no tiene que crear el flujo de Aprendizaje Automático manualmente y puede dedicarse a realizar otras tareas como el adecuado preprocesamiento de los datos o ingeniería de características, no obstante, el proceso de automatización viene con un costo. Cuando el experto necesita más que modelos que produzcan resultados relevantes los sistemas AutoML actuales se tornan insuficientes pues no posee el suficiente control para comunicarle al sistema su intención de guiar la búsqueda en otros sentidos.

En la sección 1.1 se hace un estudio del estado del arte en AutoML. Luego en la sección 1.2 se estudia los diferentes paradigmas de Optimización Multiobjetivo y se muestran diferentes propuestas de cada uno. Finalmente en la sección 1.3 se analiza casos conocidos donde al problema AutoML se le ha aplicado optimización multiobjetivo.

1.1. Aprendizaje de Máquina Automatizado

El Aprendizaje de Máquina Automatizado (Automated Machine Learning, AutoML) consiste en la generación automática de flujos, algoritmos o parámetros que resuelven un problema determinado minimizando la dependencia del usuario.

Actualmente los sistemas AutoML se pueden separar en dos conjuntos dependiendo del problema a resolver. Se encuentran los sistemas orientados a aprendizaje profundo que proponen resolver el problema de Búsqueda de Arquitecturas Neuronales (*Neural Architectural Search*, NAS) y sistemas AutoML basado en selección y

combinación; este último puede solucionar o no el problema NAS. El segundo conjunto intenta resolver el problema de combinación, selección y optimización de hiperparámetros (*Combined Algorithm Selection and Hyperparameter Optimization*, CASH) acuñado por Auto-Weka (Thornton y col. 2013). Más formalmente:

Definición 1.1 *Sea $A = \{a^1, \dots, a^k\}$ un conjunto de algoritmos con espacios de configuración asociados $\Lambda^1, \dots, \Lambda^k$. Sea D un conjunto de datos que se divide en $\{D_t^{(1)}, \dots, D_t^{(k)}\}$ y $\{D_v^{(1)}, \dots, D_v^{(k)}\}$ tal que $\forall i, 1 \leq i \leq k$ se cumple que $D_t^{(i)} = D \setminus D_v^{(i)}$. El problema CASH se define como el computo de:*

$$a_{\lambda^*}^* \in \operatorname{argmin}_{a^{(j)} \in A, \lambda \in \Lambda^{(j)}} \frac{1}{k} \sum_{i=1}^k \mathcal{L}(a^{(j)}_{\lambda}, D_t^{(i)}, D_v^{(i)})$$

Donde $\mathcal{L}(a, D_t, D_v)$ es la función de pérdida al ser entrenado en D_t y evaluado en D_v .

En el campo del Aprendizaje de Máquina Automatizado cuenta con numerosas propuestas de distintos dominios:

- Optimización Bayesiana
- Programación Evolutiva
- Búsqueda Aleatoria
- Aprendizaje por Refuerzo
- Método de Gradiente
- Modelos Constructivos
- Monte Carlo

1.1.1. Optimización Bayesiana

Las propuestas basadas en optimización bayesiana están divididas en dos componentes principales: un modelo estadístico bayesiano para modelar la función objetivo, y una función de adquisición que dirige la búsqueda en el espacio. Esos sistemas utilizan el modelo estadístico para seleccionar el mejor candidato a evaluar. Luego de la evaluación se actualiza el modelo y se repite el proceso.

Auto-WEKA (Thornton y col. 2013) Utiliza como algoritmos de aprendizaje las máquinas WEKA, una popular biblioteca de Aprendizaje Automático en Java. Presenta dos optimizadores bayesianos, una propuesta de configuración secuencial de Algoritmos basados en Modelos (SMAC por Hutter y col. 2011) y un estimador de Parzen con estructura de árbol (TPE por Bergstra, Bardenet y col. 2011), ambos se ejecutan en paralelo y se retorna la función con menor error en validación de entre todas las descubiertas.

Auto-skelearn (Feurer, Klein y col. 2015) Utiliza como biblioteca de aprendizaje automático a scikit-learn (Pedregosa y col. 2011) y extiende la propuesta de AutoWEKA introduciendo mejoras como la inclusión de un paso de meta-aprendizaje que reduce el espacio de búsqueda aprendiendo de modelos que obtuvieron un buen desempeño en conjuntos de datos similares y luego un paso de selección de ensamblers que le permite reusar los flujos de Aprendizaje de Máquina que tuvieron mejor rendimiento.

HyperOpt-Sklearn (Komer y col. 2014) Similar a Auto-Skelearn con respecto a la biblioteca de aprendizaje subyacente, scikit-learn, utiliza Hyperopt (Bergstra, Yamins y col. 2013) para describir el espacio de búsqueda. Provee una interfaz de optimización que distingue un espacio de configuraciones y una función de evaluación que asigna valores de pérdida reales a puntos dentro de dicho espacio. Hyperopt requiere que el espacio esté definido como una distribución de probabilidad, permitiendo una codificación más expresiva de las intuiciones de los expertos con respecto a los valores de los hiperparámetros. Utiliza SMBO (*Sequential Model-based Bayesian Optimization*), considerando este algoritmo como una componente intercambiable con el fin de que cualquier técnica de búsqueda pueda ser utilizada con cualquier problema de búsqueda.

Auto-Keras (Jin y col. 2018) Un sistema de AutoML enfocado en resolver el problema NAS. Está basado en la biblioteca Keras de Python, utilizada para crear redes neuronales. La idea principal de esta propuesta es explorar el espacio mediante la transformación de redes, utilizando un algoritmo de Optimización Bayesiana eficiente. Dichas transformaciones permiten generar redes manteniendo las configuraciones alcanzadas con entrenamientos anteriores, disminuyendo el uso de los recursos de cómputo. Auto-Keras a nivel técnico está estructurado para el uso eficiente de la memoria y la ejecución en paralelo sobre el CPU y el GPU.

autogxgboost (Thomas y col. 2018) Se catáloga como un sistema AutoML *single-learner* debido a que solo utiliza un algoritmo de aprendizaje y optimiza los hiperparámetros para este utilizando optimización bayesiana. Requiere que el algoritmo

sea un metodo de aumento de gradiente con árboles (*Gradient Boosting with Trees*, GBT) como *xgboost* (Chen y Guestrin 2016) debido a su prometedor rendimiento con los paraméetros adecuados.

1.1.2. Programación Evolutiva

Los sistemas AutoML basados en programación evolutiva funcionan creando una población inicial de flujos válidos para luego seleccionar los de mejor rendimiento respecto a cierta métrica. Los más aptos son utilizados para crear la población de la próxima iteración. Las propuestas difieren principalmente en como modelan el espacio de búsqueda y que estrategias usan para crear los nuevos individuos.

TPOT (Olson y Moore 2016) Utiliza algoritmos de aprendizaje de la biblioteca scikit-learn (Pedregosa y col. 2011) junto con el algoritmo xgboost (Chen y Guestrin 2016), es uno de los sistemas AutoML evolutivos más reconocidos. Cada algoritmo de Aprendizaje Automático a su disposición se le considera un operador, además de tener un operador especial de cruce. Permite tener varias copias del dataset y aplicar operadores sobre ellos simultáneamente para luego recombinarlos y crear flujos con los operadores que hayan tenido mejor rendimiento. TPOT además realiza una búsqueda multiobjetivo sobre las soluciones encontradas utilizando NSGA-II (Deb, Pratap y col. 2002) optimizando simultáneamente la relevancia de los resultados y minimizando la cantidad de operadores aplicados sobre los flujos de Aprendizaje Automático para mantener las soluciones simples y evitar el sobreajuste a los datos.

RECIPE (de Sá y col. 2017) Utiliza programación genética basado en gramáticas (*Grammar Genetic Programming*, GGP) para generar sus flujos. El sistema recibe como entrada un conjunto de datos y una gramática libre del contexto (CFG) que utiliza para formar una población inicial de flujos. Partiendo de los mejores individuos aplica operadores de mutación y cruce sobre estos tal que no se violen las reglas de la gramática para la obtención de una nueva población. Utiliza como base de aprendizaje a la biblioteca scikit-Learn (Pedregosa y col. 2011).

AutoGOAL (Estevez-Velarde y col. 2020) Utiliza una Gramática Probabilística Libre del Contexto (*Probabilistic Context-Free Grammars*) para modelar el espacio de búsqueda y Evolución de Gramática Probabilística (*Probabilistic Grammatic Evolution*, Mégane y col. 2021) para dirigir la búsqueda de acuerdo a los individuos más aptos de cada iteración. AutoGOAL está compuesto por gran cantidad de algoritmos de aprendizaje de diversas bibliotecas de Python que le permite abarcar problemas de cualquier tipo, incluido problemas de procesamiento de lenguaje natural, donde cuenta con algoritmos pre-procesadores de datos.

1.1.3. Búsqueda Aleatoria

Consiste en la exploración del espacio de búsqueda, ya sea de forma secuencial o en paralelo mediante la selección de soluciones aleatorias. Este método es utilizado muchas veces como comparación frente a estrategias más avanzadas.

NASH (Wei y col. 2016) Neural Arquitecture Search by Hillclimbing es una propuesta de solución al problema NAS. Partiendo de una red aleatoria o predeterminada a la cual le aplica *morfismos de red* para obtener nuevas redes “vecinas”. Se evalúan de acuerdo a una métrica y la que tenga mejor rendimiento se toma como principal y se repite el proceso partiendo de esta. La estrategia de *hill climbing* que sigue elimina la necesidad de rentrenar las nuevas redes desde cero.

H2O AutoML (LeDell y Poirier 2020) Disponible en varios lenguajes, utiliza los algoritmos de Aprendizaje Automático definidos en la biblioteca H2O (Boehmke y Greenwell 2019). Construye múltiples modelos con diferentes juegos de hiperparámetros seleccionados aleatoriamente en base a ciertas heurísticas. En función de la disponibilidad de recursos de cómputo la búsqueda se detiene y los mejores modelos se utilizan para construir un ensemble.

TransmogrifAI (Tovbin s.f.) Ideado con el objetivo de tener un buen rendimiento en datasets grandes, fue diseñado para ejecutarse sobre Apache Spark. Encapsula cinco componentes del proceso de aprendizaje automático en diferentes pasos:

1. Realiza un preprocesamiento de los datos donde obliga al científico a especificar un esquema y establecer un sistema de tipos sobre estos. Transmogrif los analiza siendo capaz de inferir el tipo de algunos.
2. Aplica ingeniería de características automatizada: busca obtener la representación numérica adecuada para cualquier tipo de datos.
3. Hace una validación de las características para mitigar la probable explosión de dimensionalidad ocurrida en el paso anterior y los problemas que ello acarrea.
4. Resuelve el problema de selección automática de modelos ejecutando un torneo de varios algoritmos de Aprendizaje Automático, y escoge el mejor según el error de validación promedio.
5. En cada uno de los pasos anteriores se ejecuta además un paso subyacente de optimización de hiperparámetros, contrario a muchos sistemas AutoML que solo optimizan solamente para el modelo solución.

1.1.4. Aprendizaje por Refuerzo

El aprendizaje por refuerzo como estrategia de búsqueda consiste en entrenar un agente que realiza modificaciones sobre una solución con el objetivo de maximizar una recompensa que depende del rendimiento de dicha solución. Esta estrategia es utilizada mayormente en sistemas AutoML que buscan resolver el problema NAS, donde el agente puede realizar acciones como añadir, remover, o modificar una capa o sus hiperparámetros. Las propuestas difieren según el diseño del agente.

EAS (Cai y col. 2018) Efficient Architectural Search utiliza un meta-controlador basado en aprendizaje por refuerzo (*Reinforcement Learning*, RL, Kaelbling y col. 1996) y el algoritmo REINFORCE (Williams 1992) para actualizarlo. En general, este sistema modela el proceso de diseño automático de arquitecturas como un proceso de toma de decisiones secuencial, donde verifica el estado de la red actual y realiza la operación de transformación correspondiente. Despues de una cantidad determinada de pasos, la arquitectura resultante es evaluada para obtener la señal de recompensa, que luego es utilizada para actualizar el meta-controlador, maximizando la validación.

NASNet (Zoph y col. 2018) Utiliza aprendizaje por refuerzo para la construcción óptima de una red neural para cualquier conjunto de imágenes. La contribución clave de esta propuesta es que el diseño del espacio de búsqueda permite construir la arquitectura en un corpus de datos pequeño y luego transferir los conocimientos para el entrenamiento de un conjunto de datos más grande, aligerando el costo computacional de aplicarlo directamente sobre este último.

1.1.5. Modelos Constructivos

Estos métodos exploran el espacio de búsqueda con una forma estructurada pues se define de antemano los posibles modelos y formas de combinarlos. Se establece un orden de evaluación de las soluciones y la búsqueda finaliza cuando todo el espacio ha sido explorado. Esta estrategia es útil cuando el espacio de búsqueda es pequeño y consta de modelos bien definidos, variados y con un buen rendimiento.

AutoGluon (Erickson y col. 2020) Realiza procesamiento de datos avanzados, aprendizaje profundo y ensamblaje de modelos multicapa. Reconoce automáticamente el tipo de datos de cada columna para un procesamiento de datos robustos, luego construye un ensemble multicapa tomando como entrada la instancia de cada uno de los tipos de modelos predefinidos. Cada modelo se entrena de forma secuencial y se añade al ensemble. Para finalizar, se realiza un proceso de post-optimización para destilar el ensemble resultante en un modelo más pequeño con rendimiento similar.

1.1.6. Monte Carlo

Esta estrategia se emplea en espacios de búsquedas jerárquicos para explorar eficientemente el árbol que describe todas las posibles soluciones. En cada iteración se deciende por una rama del árbol hasta llegar a un nodo hoja, el camino representa una posible solución. Como todo algoritmo de Monte Carlo requiere de una definición adecuada de una función que permita un balance entre exploración y explotación. Esta estrategia requiere un espacio de búsqueda donde las decisiones más importantes estén posicionadas al principio en el árbol para ser efectiva.

ML-Plan (Mohr y col. 2018) Un sistema novedoso basado en redes de tareas jerárquicas (HTN, Erol y col. 1994). La búsqueda es aleatoria, construyendo flujos parciales junto con un mecanismo que evita el sobreajuste. Cuenta con dos tipos de algoritmos: preprocesadores de datos y de aprendizaje. Sus flujos son un par compuestos por un procesador parametrizado y un algoritmo de aprendizaje.

1.2. Optimización Multiobjetivo

Optimización Multiobjetivo es la rama de la Ciencia y la Matemática que se dedica a optimizar varias funciones objetivos simultáneamente. Cuanta con abundante campo las ingenierías y la economía donde muchas veces suelen existir conflictos entre los objetivos a optimizar. Su aplicación es amplia y variada:

- La necesidad de lograr un balance entre energía producida y combustible utilizado por una termoeléctrica (Shirazi, Aminyavari y col. 2012 y Shirazi, Najafi y col. 2014).
- El diseño óptimo de una estructura como la configuración de gabinete de control (Pllana y col. 2019) o una granja solar (Ganesan y col. 2013).
- La inspección de infraestructura complejas, que es muy costosa y no resulta factible cubrir el área completa por lo que hay que buscar una solución óptima que haga concesiones entre área cubierta y costos (Ellefsen y col. 2017).
- La distribución adecuada de recursos de radio para satisfacer eficientemente el requerimiento de sus usuarios (Björnson, Jorswieck y col. 2013).

El problema multiobjetivo consiste en optimizar varias funciones y lograr soluciones que hagan las concesiones pertinentes entre los criterios que tengan conflictos entre si.

Definición 1.2 Optimización Multibojetivo: Dado m funciones objetivos: $f_1 : \mathcal{X} \rightarrow \mathbb{R}, \dots, f_m : \mathcal{X} \rightarrow \mathbb{R}$ que transforman un vector x del espacio de decisión \mathcal{X} a un valor de \mathbb{R} . Se define el problema multiobjetivo como:

$$\min f_1(x), \dots, f_m(x), x \in \mathcal{X}$$

Cuando se optimizan varias funciones, la mejor solución no es un escalar, sino un vector y no necesariamente único pues al haber más de dos medidas de evaluación un vector puede ser superior a otro sobre ciertas componentes pero no en todas. Un vector se dice mejor o que *Pareto domina* a otro cuando es superior en al menos una componente y no peor en las restantes. Se conoce formalmente en la literatura como *Pareto dominación*.

Definición 1.3 Pareto Dominación: Dado dos vectores en el espacio objetivo, $x, z \in \mathcal{Y}$, se dice que x Pareto domina a z (i.e. $x \prec z$), si y solo si:

$$\forall i \in \{1, \dots, m\} : x_i \leq z_i \text{ y } \exists j \in \{1, \dots, m\} : x_j < z_j$$

Cuando se tiene un subconjunto de vectores, dentro de todo el espacio, a los que ningún otro vector domina, se le llama Frente de Pareto y es el conjunto solución del problema multiobjetivo.

Definición 1.4 Frente de Pareto: Todos los vectores x del espacio objetivo \mathcal{Y} tal que no exista un vector $y \in \mathcal{Y}$ que Pareto domine a x .

$$\mathcal{P} = \{x | x, y \in \mathcal{Y}, \neg \exists y \prec x\}$$

El problema multiobjetivo se ha intentado resolver utilizando tres enfoques de diferentes campos de la Computación y la Matemática:

1. Técnicas de Escalarización.
2. Métodos Numéricos.
3. Algoritmos Evolucionarios.

1.2.1. Técnicas de Escalarización

Las técnicas de escalarización han sido las más utilizadas para resolver el problema multiobjetivo (Miettinen 2012). Estas técnicas consisten en agregar funciones objetivos o reformularlas como restricciones en una sola función sobre la cual se aplica un método de optimización estándar de un solo objetivo. La nueva función objetivo además se parametriza con un vector de pesos, que dependiendo de sus valores resulta en un punto distinto del frente de pareto.

1. Linear Weighting: La técnica más conocida, todas las funciones objetivos se combinan en una sola y a cada una se le asigna un peso distinto.

$$\min \sum w_i f_i(x), x \in X$$

Se garantiza que una solución de esta nueva función objetivo siempre está sobre el frente de Pareto y si este es convexo se puede hallar cualquier punto de este con el correcto vector de peso (Emmerich y Deutz 2018). El problema yace cuando el frente de Pareto tiene forma cóncava donde *Linear Weighting* resulta insuficiente por no ser capaz obtener los puntos de esta sección del frente.

2. ϵ -constrain: Se selecciona una función objetivo como principal y las demás se establecen como restricciones de esta tal que sean menor que cierto ϵ_i por cada función objetivo.

$$\begin{aligned} & \min f_1(x), x \in X, \text{ sujeto a:} \\ & g_i(x) \leq \epsilon_i \quad \forall i, 2 \leq i \leq n \end{aligned}$$

Esta enfoque presenta dos dificultades principales: la valores de los ϵ_i , para una adecuada selección requieren conocimiento previo del frente de Pareto y al igual que *Linear Weighting*, no es capaz de detectar soluciones en las partes cóncavas del frente (Emmerich y Deutz 2018).

3. Chebychev Distance (CSP): Se establece un punto de referencia z^* y se utiliza la distancia de Chebychev de los vectores objetivos hacia este como función objetivo utilizando un vector de pesos $\lambda \in \mathbb{R}_{\prec 0}^m$, donde $\mathbb{R}_{\prec 0}^m = \{x | x \in \mathbb{R}^m, x \prec 0\}$.

$$\min \max_{i \in 1, \dots, m} \lambda_i |f_i(x) - z_i^*|, x \in X$$

CSP dado un punto de referencia y vector de pesos adecuados puede encontrar cualquier punto del frente de Pareto, no importa su forma (Emmerich y Deutz 2018).

Recientemente escalarización ha encontrado uso en algoritmos genéticos por descomposición para obtener muestras distintas del frente de pareto. En Paria y col. 2020 se propone un algorimto capaz de muestrear un area determinada del frente de Pareto utilizando una estrategia basada en escalarización aleatoria.

1.2.2. Algoritmos Numéricos

En principio todos los algoritmos de escalarización se pueden resolver utilzando métodos numéricos. Además existen otros métodos que intentan resolver el problema

haciendo cumplir las condiciones de Karush-Kuhn-Tucker (KKT) definidas por Kuhn y Tucker 2014.

La idea va de encontrar al menos una solución al sistema de ecuaciones creado tras intentar resolver el problema KKT. Luego se utilizan métodos de continuación y homotopía para añadir al conjunto solución soluciones cercanas a esta. Este tipo de enfoque no es ideal para espacios de búsqueda complejos pues requiere que las soluciones satisfagan las condiciones de convexidad local y diferenciabilidad (Hillermeier y col. 2001 y Schütze y col. 2005).

Existen otros métodos para la búsqueda de mínimos globales como *Técnicas de Subdivisión* por Dellnitz y col. 2005, *Optimización Global Bayesiana* por Emmerich, Yang y col. 2016 y *Optimización de Lipschitz* por Žilinskas 2013. Además se investiga métodos numéricos que no necesiten diferenciar para su resolución utilizando técnicas de búsquedas directas. Se pueden encontrar ejemplos de estos algoritmos en Custódio y col. 2011 y Audet y col. 2010.

1.2.3. Algoritmos Evolucionarios Multiobjetivos

Los algoritmos genéticos utilizan paradigmas extraídos de la naturaleza, tal como selección natural, mutación y recombinación para dirigir una población (o conjunto de vectores de decisión) hacia una solución óptima (Back 1996).

Los algoritmos evolucionarios multiobjetivos (MOEA) generalizan esta idea, y son diseñados para acercarse en cada iteración al frente de Pareto. Como en este caso no existe solución única, la manera de seleccionar los individuos cambia fundamentalmente. Dentro de los MOEA existen tres paradigmas principales:

1. **MOEA basados en el frente de Pareto:** Se identifican por dividir el proceso de selección en dos etapas. En la primera se organizan los individuos según su índice de dominación y se acomodan en distintos subconjuntos según la cantidad de soluciones que los dominen (e.g. un subconjunto para las soluciones a las cuales nadie domina, otro para los son dominados por alguna solución etc.). En la segunda ordenación cada subconjunto se ordena buscando que los primeros lugares sean los elementos más representativos de cada subconjunto. NSGA-II (Deb, Pratap y col. 2002) y SPEA2 (Zitzler y Thiele 1999) son algoritmos de este tipo.
2. **Basados en indicador:** Estos utilizan un indicador para calcular cuán cercano es el conjunto actual al frente de Pareto (unario), o cuánto mejora el nuevo conjunto de soluciones respecto a la iteración anterior (binario). Ejemplo de este es SMS-EMOEA (Emmerich, Beume y col. 2005) que suele converger al frente de Pareto con soluciones igualmente distribuidas.

3. **Basados en descomposición:** La idea principal consiste en descomponer el problema en pequeños subproblemas cada una correspondiente a una sección del frente de Pareto. Por cada sub-problema se resuelve utilizando escalarización con diferente parametrización. El método de escalarización más usado en estos casos suele ser CSP debido a ser capaz de obtener cualquier punto del frente de Pareto. Ejemplo de este paradigma son MOEA/D (Zhang y Li 2007), NSGA-III (Deb y Jain 2013) y MOMSA (Sharifi y col. 2021).

El término Optimización Multiobjetivo se utiliza cuando el número de funciones objetivos son dos o tres. Para un cantidad de criterios mayor se le conoce coloquialmente en la literatura como Optimización para Muchos Objetivos o *many-objective optimization* en inglés, propuesto inicialmente por Fleming y col. 2005. Se hace enfasis en esta diferenciación pues al aumentar el número de métricas a optimizar:

1. ya no es posible visualizar el frente de Pareto;
2. la computación de indicadores o de selección para muchos algoritmos se convierte en problemas NP-duros;
3. existe un rápido crecimiento de puntos no dominados, mientras mayor número de objetivos, la probabilidad de que un punto sea no dominado en un set con distribución normal tiende exponencialmente a 1.

Los algoritmos que mejor han tenido resultado en esta área son los algoritmos basados en descomposición.

1.3. AutoML y Multiobjetivo

Recientemente ha habido enfasis en la comunidad FatML (*Fairness, Accountability and Transparency in Machine Learning*) sobre como optimizar utilizando una sola métrica acarrea problemas que pueden ser evitados optimizando simultáneamente para múltiples métricas (Barocas y col. 2017). No obstante, es escaso el estudio sobre sistemas de Aprendizaje de Máquina Automatizado a los que se les aplica optimización multiobjetivo.

Definimos el problema multiobjetivo aplicado a sistemas AutoML de la siguiente manera:

Definición 1.5 *Dado un conjuntos de datos D y un conjunto de métricas a evaluar $M = \{f_1, f_2, \dots, f_m\}$, se espera que un sistema AutoML \mathcal{A} retorne un conjunto de flujos, algoritmos o redes $P = \{p_1, p_2, \dots, p_k\}$ tal que su evaluación con respecto a las métricas en M sean una aproximación del frente de Pareto en el espacio objetivo \mathcal{Y} .*

$$\mathcal{A}(D, M) = P$$

TPOT (Olson y Moore 2016) es un ejemplo de Sistema AutoML que aplica multiobjetivo para su resolución. Después de cada iteración ordena los algoritmos utilizando NSGA-II (Deb, Pratap y col. 2002) guiándose por exactitud de las predicciones y número de operadores utilizados (i.e. algoritmos de aprendizaje). No obstante el resultado de TPOT no es un conjunto soluciones representativas del frente de Pareto, sino la solución que mejor rendimiento tuvo respecto a una métrica de relevancia durante las pruebas de validación. El uso de optimización multiobjetivo en TPOT se utiliza principalmente como medida para evitar el sobre ajuste del modelo a los datos.

Pfisterer y col. 2019 propone una solución al problema multiobjetivo basándose en *autoxgboost* (Thomas y col. 2018) y utilizando parEgo (Knowles 2006), un sistema de escalarización multiobjetivo utilizando CSP con un vector de peso uniforme. La función resultante tras aplicar CSP sobre las múltiples métricas es optimizada utilizando optimización bayesiana estándar de un sólo objetivo. El algoritmo se beneficia de un humano en el proceso de optimización que ayuda a guiar la búsqueda aplicando restricciones sobre el espacio de búsqueda interactivamente. Al estar basados en *autoxgboost* (Thomas y col. 2018) un sistema AutoML que utiliza un solo algoritmo, el espacio de búsqueda se encuentra limitado y es posible que no incluya configuraciones óptimas.

Capítulo 2

AutoML Heterógeno Multiobjetivo

AutoML Heterógeno Multiobjetivo es una propuesta dirigida a resolver el problema definido en 1.5. El núcleo fundamental de la idea radica en trabajar sobre dos espacios bien definidos. El sistema busca y obtiene soluciones en el espacio de decisión \mathcal{X} y el resultado de evaluar cada una de ellas se representa como un vector numérico donde cada una de sus componentes es la evaluación con respecto a cierta métrica. Este vector representa un punto en el espacio objetivo \mathcal{Y} con $\mathcal{Y} \subseteq \mathbb{R}^m$ donde m representa el número de métricas a evaluar. Cuando $m = 1$ es trivial la búsqueda del más apto. Cuando $m \geq 2$ se complejiza la ordenación pues pueden existir varias soluciones buenas y no es posible hablar de una mejor solución, en cambio se habla de *Pareto dominación* definido en (1.3). La selección de los vectores más aptos del espacio de decisión se realiza utilizando un algoritmo evolutivo multiobjetivo pues además de ser agnósticos al espacio de decisión han demostrado ser los más resistentes al frente de Pareto cuando tiene formas extrañas.

2.1. Descripción General

La solución propuesta (ver 2.1) está compuesta por dos elementos fundamentales:

1. Un sistema AutoML capaz de generar y evaluar múltiples soluciones basado en un corpus de datos y un conjunto de métricas. Generar nuevas soluciones partiendo de las soluciones más aptas de la generación anterior.
2. Un algoritmo de ordenación multiobjetivo capaz de funcionar solamente con el espacio objetivo y obtenga soluciones igualmente distribuidas a lo largo del frente de Pareto.

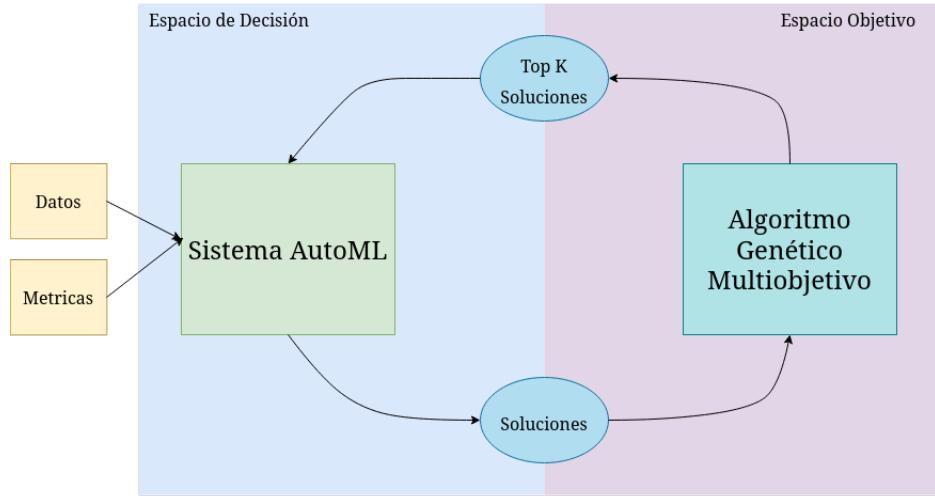


Figura 2.1: Flujo general de la propuesta

La propuesta en cada iteración utiliza la evaluación de las soluciones generadas por el sistema de AutoML. La evaluación de una solución se representa como un vector v de m componentes donde v_i es la evaluación de la solución con respecto a la métrica i . El algoritmo ordena estos vectores utilizando algún MOEA y retorna una lista ordenada de estos. Este proceso se repite hasta que se cumpla algún criterio de parada.

La separación por espacios existente entre ambas componentes de la propuesta permite que sean intercambiables. Todo sistema de AutoML que cumpla con estas condiciones es apto para la propuesta, no importa si está enfocado en resolver el problema NAS o el problema CASH y toda algoritmo multiobjetivo capaz de producir ordenaciones efectivas utilizando solamente el espacio objetivo puede ser utilizado.

2.1.1. Descripción Formal

El sistema propuesto está compuesto por un sistema de Aprendizaje de Máquina Automatizado $\mathcal{A}(D, M, P)$ y un algoritmo evolutivo multiobjetivo $\mathcal{H}(TP)$. Los parámetros de entrada $D = \{(x_1, y_1), \dots, (x_n, y_n)\}$ representa el conjunto de datos de entrada, $M = \{f_1, f_2, \dots, f_m\}$ un conjunto de métricas a evaluar, $P = \{p_1, p_2, \dots, p_k\}$ el conjunto de soluciones de mejor rendimiento producidas por \mathcal{A} en la iteración anterior. \mathcal{H} toma como entrada TP que representa el conjunto de soluciones producidas en la iteración actual tras aplicar $\mathcal{A}(D, M, P)$.

Algoritmo 2.1: Solución a AutoML Heterógeno Multiobjetivo

Entrada: \mathcal{A}, D, M

Salida: P

```

1  $TP \leftarrow \mathcal{A}(D, M, \emptyset)$  ;           // Se obtiene población inicial aleatoria
2 mientras no se cumplan condiciones de parada hacer
3   |    $P = \mathcal{H}(TP)$  ;           // Se extraen los más aptos de una población
4   |    $TP = \mathcal{A}(D, M, P)$  ;           // Se genera una nueva población
5 fin

```

2.2. MOEA

Esencialmente cualquier tipo de MOEA que sea suficiente información sobre espacio objetivo se puede utilizar. Específicamente para esta propuesta utilizamos una adaptación de NSGA-II (Deb, Pratap y col. 2002) por ser un algoritmo que sigue una idea intuitiva y tiene un buen rendimiento general, suficiente para demostrar la efectividad de la propuesta.

2.2.1. NSGA-II

NSGA-II (Deb, Pratap y col. 2002) es un algoritmo multiobjetivo basados en el frente de Pareto característicos por dividir el proceso de selección de los individuos más aptos en dos etapas:

- *Non Dominated Sorting* donde divide a la población basado en sus diferentes niveles de dominación.
- *Crowding Distance Sorting* donde se dedicada a preservar la diversidad entre los individuos encontrados.

Non Dominated Sorting

Durante esta etapa se agrupan las soluciones de acuerdo a su índice o nivel de dominación. Dicho índice está determinado por la cantidad de soluciones diferentes que la dominan.

Definición 2.1 *Dado un vector x y un conjunto Y de vectores en el espacio objetivo \mathcal{Y} tal que los vectores en Y dominan a x (i.e. $Y = \{y | y \succ x\}$) se dice que $Ind(x) = |Y|$.*

Definición 2.2 *El conjunto de todas la soluciones con un mismo índice de dominación i se les dice frente de rango i :*

$$F^i = \{x | x \in \mathcal{Y}, Ind(x) = i\}$$

El primer paso de *Non Dominated Sorting* es calcular el índice de dominación de todas las soluciones revisando todos los pares de vectores $x, y \in \mathcal{Y}$ a la vez que se crea un grafo dirigido entre estos tal que existe una arista entre las soluciones x y y si $x \prec y$. Luego partiendo de las soluciones con nivel de dominación 0, visita todas las soluciones que estas dominan y les reduce el índice en uno. El proceso se repite hasta que no queden soluciones por analizar. El resultado de esta primera etapa es una lista ordenada $F = \{F^0, \dots, F^k\}$ de todos los frentes de distinto rango obtenidos.

Crowding Distance Sorting

Crowding Distance Sorting(CD), uno de los aportes claves hechos por NSGA-II, se utiliza sobre F^i un frente de rango i obtenido en la etapa anterior con el objetivo de ordenar sus elementos según cuán representativos del conjunto sean. Esta parte es vital pues permite que en los casos donde haya que utilizar solo la porción de un frente, se utilice la más representativa de este manteniendo la diversidad del conjunto.

Calcular el CD de cada solución requiere ordenarlas según su valor normalizado por cada función objetivo y calcular el promedio de la distancia entre una solución y sus dos adyacentes respecto a dicha función objetivo. Esta distancia es el perímetro del cuboide formado utilizando los vecinos más cercanos como vértices. Los puntos que representan el mínimo y máximo de al menos alguna función objetivo se les asigna *crowding distance* infinita. El algoritmo queda descrito en 2.2.

Algoritmo 2.2: Crowding Distance Sorting

```

// F como entrada representa un frente de rango i
Entrada: F
// SF como salida representa el frente ordenado según CD
Salida: SF

1 para  $i$  desde 0 hasta  $|F|$  hacer
2   |    $F[i].dist \leftarrow 0$  ;
3 fin
4 para cada función objetivo  $m$  hacer
5   |    $F \leftarrow ordenar(F, m)$  ;           // se ordena F con respecto a m
6   |    $F[0].dist \leftarrow \infty$  ;
7   |    $F[|F|].dist \leftarrow \infty$  ;
8   |   para  $i$  desde 2 hasta  $|F|-1$  hacer
9     |     |    $F[i].distance = F[i].distance + \frac{(F[i+1].m - F[i-1].m)}{f_m^{max} - f_m^{min}}$ 
10    |   fin
11 fin
12  $SF \leftarrow ordenar(F, dist)$  ;           // se ordena F respecto a CD

```

Ciclo Principal

Inicialmente se crea una población aleatoria P_0 de soluciones de tamaño N . Cada solución es ordenada de acuerdo a *Non Dominated Sorting* y se aplica un torneo binario utilizando operadores genéticos de selección, recombinación y mutación para crear una población descendiente Q_0 de tamaño N .

Luego se forma una población $R_t = P_t \cup Q_t$ de tamaño $2N$. La población R_t es ordenada de acuerdo al nivel de dominación de cada individuo obteniéndose una lista de frentes de distinto rango $F = \{F^0, \dots, F^k\}$. Las soluciones en F^0 representan las mejores soluciones encontradas por el algoritmo hasta el momento y es necesario que pasen a formar parte de la población de la siguiente iteración P_{t+1} . Si $|F^0| < N$ el frente completo pasa formar parte de P_{t+1} . Los miembros restantes de la nueva población son escogidos de los demás frentes acorde a su índice de dominación. Si F^k no capaz de pertencer completo a P_{t+1} debido a que $\sum_0^k |F^i| > N$ se aplica *Crowding Distance Sorting* sobre F^k y se llena las posiciones restantes con los individuos de mayor distancia.

La nueva población P_{t+1} se utiliza para producir una población Q_{t+1} con operadores de selección, cruzamiento y mutación. El procedimiento general de NSGA-II se muestra en la figura 2.2.

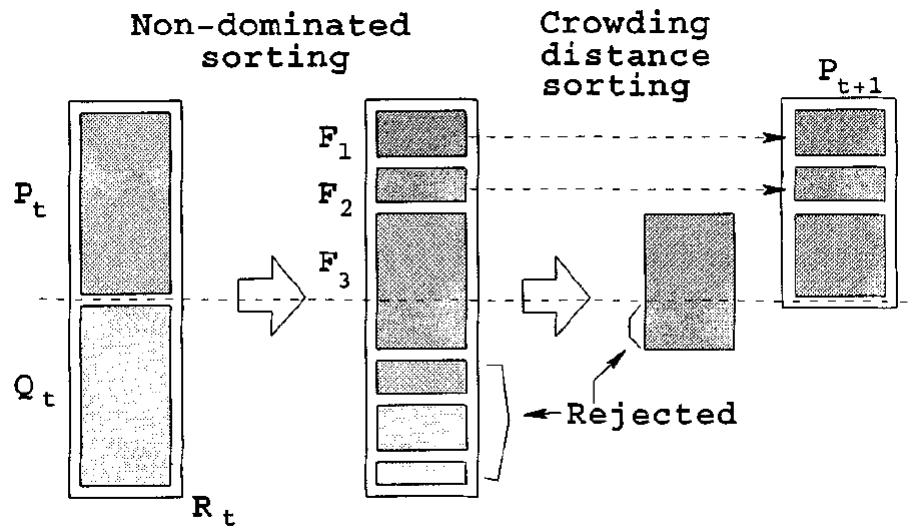


Figura 2.2: Funcionamiento de NSGA-II

2.2.2. NSGA-II Generalizado

NSGA-II realiza una serie de asunciones sobre las poblaciones con las que trabajan que no lo hacen perfectamente integrable con nuestra propuesta. Proponemos una adaptación del algoritmo que conserva sus dos etapas de ranking fundamentales pero con un ciclo principal más sencillo que no aplica operadores genéticos ni modificaciones sobre la población limitándolo al espacio objetivo \mathcal{Y} .

Ciclo Principal

El ciclo de nuestra propuesta recibe una población inicial P de tamaño N y pasa directamente a aplicar *Non Dominated Sorting* con los que se obtiene una lista de frentes de distinto rango $F = \{F^0, \dots, F^l\}$ con el objetivo de formar un conjunto P' de tamaño k que representen los mejores elementos de P . Las soluciones que pasan a formar parte de P' se escojen añadiendo en grupos por cada frente F^i hasta que cierto frente F^q que no pueda añadirse completo. Los puestos restantes se llenan con los elementos de mayor distancia de F^q tras aplicar *Crowding Distance*.

Se simplifica el ciclo respetando la idea de la propuesta general. Los sistemas AutoML son los que deciden como generar soluciones y que hacer con la información de los más aptos, permitiendo que se puedan utilizar como una caja negra intercambiable.

Capítulo 3

AutoGOAL Multiobjetivo

AutoGOAL (Estevez-Velarde y col. 2020) es un sistema AutoML Heterógeno modular y fácilmente extensible. Modela su espacio de búsqueda utilizando Gramáticas Probabilística Libre del Contexto (PCFG, Mégane y col. 2021), una extensión de las Gramáticas Libres del Contexto al asignar a cada producción una probabilidad. Con el fin de dirigir la búsqueda utiliza *Probabilistic Grammatical Evolution* (PGE), una modificación de *Grammatical Evolution* para ser usada con PCFG.

PGE está basado en Algoritmos de Estimación de Distribución (EDAs) una técnica evolutiva que remplaza las operaciones de mutación y cruce por un muestreado sobre las probabilidades de las producciones de PCFG. Dado una gramática G y una cadena $c \in G$, PGE modifica las probabilidades asociadas a cada producción de G para que las próximas generaciones que utilicen G tengan mayor probabilidad de utilizar las mismas producciones que c . Cuando c representa el individuo más apto entonces las próximas generaciones tienen mayor probabilidad de conservar sus características. Este es un algoritmo sencillo pero efectivo que logra un balance entre exploración y explotación local haciéndolo resistente a mínimos locales y a la generación de soluciones iguales.

3.1. Implementación

La manera de AutoGOAL de representar su PCFG utilizada para modelar su espacio de decisión es utilizando un grafo acíclico dirigido (DAG) donde cada nodo representa una posible cadena de la gramática y sus aristas apuntan a todas las posibles cadenas a las que se puede expandir sustituyendo algún No Terminal por una producción de la Gramática correspondiente. Cuando una cadena se encuentra completamente expandida, i.e. compuesta por solo símbolos terminales, se tiene una solución válida del sistema.

Inicialmente cada arista del DAG se inicializa con una probabilidad igual a todas

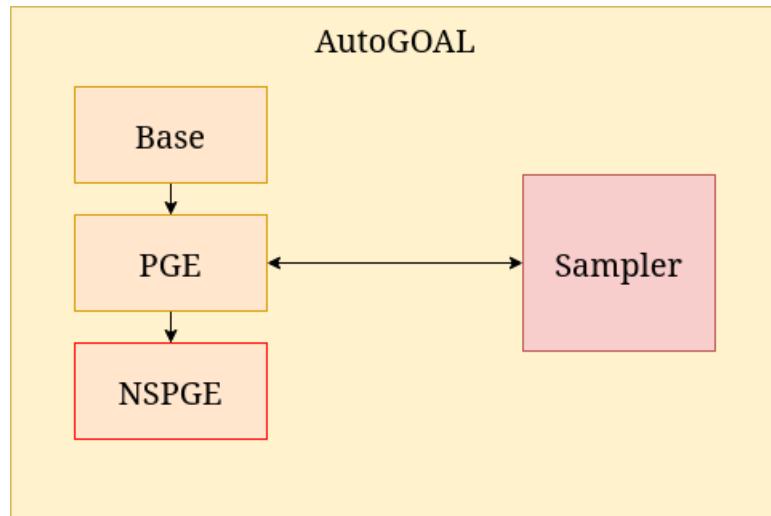


Figura 3.1: Diagrama general

sus aristas hermanas (i.e. todas las aristas que salen de su mismo nodo) tal que la suma de sus probabilidades sea 1. Un camino se escoge utilizando una variable uniforme donde su valor determina la arista a seleccionar. Una vez que se tiene un individuo apto AutoGOAL aplica PGE sobre el grafo actualizando las probabilidades de todas las aristas según el camino utilizado por dicho individuo. Esta operación se realiza sobre los k individuos más aptos de la población generada.

3.1.1. NSPGE

NSPGE, un algoritmo de búsqueda basado en nuestra propuesta, define una nueva ordenación sobre las soluciones generadas por AutoGOAL. Se define como una clase que hereda directamente de PGE con el objetivo de extender la lógica de PGE encargada de seleccionar los k mejores individuos de acuerdo a m métricas, con $m \geq 2$. PGE define su simple lógica de selección de individuos dentro del método `_indices_of_the_fittest_` el cual NSPGE sobreescribe.

La redefinición `_indices_of_the_fittest_` incluye las dos etapas de ordenación definida por Deb, Pratap y col. 2002. Dado n soluciones se agrupan según su índice de dominación (i.e. Non Dominated Sorting), donde cada agrupamiento se le llama frente de rango i , donde i representa la cantidad de soluciones que dominan a cada una de estas. Una vez establecidos estos frentes se extraen las primeras k soluciones, en caso de que no se pueda seleccionar un frente completo, se aplica Crowding Distance para seleccionar los individuos más representativa de este.

```

1 def _indices_of_fittest(self, fns: List[List[float]]):
2     # Se ordenan todas las soluciones segun su orden
  
```

```

3   # de dominacion
4   fronts = self.non_dominated_sort(fns)
5   indices = []
6   k = int(self._selection * len(fns))
7
8   for front in fronts:
9     if len(indices) + len(front) <= k:
10       indices.extend(front)
11     else:
12       # Cuando solo se utiliza una porcion del frente
13       # se aplica crowding distance
14       indices.extend(
15         sorted(
16           front,
17           key=lambda i: -self.crowding_distance(fns, front, i)
18         )[: k - len(indices)]
19       )
20       break
21   return indices

```

Ejemplo de código 3.1: Nueva ordenación

Non Dominated Sort

La ordenación se conforma por dos pasos lógicos fundamentales:

1. Se verifica todo par de soluciones x, y encontradas y se aplica $x \prec y$ con el objetivo de calcular el índice de dominación de cada solución. Además se construye un DAG conformado por las soluciones donde existe una arista entre las soluciones x y y si $x \prec y$. La raíz de dicho DAG está conformado por las soluciones que nadie domina.
2. Se recorre DAG utilizando una versión de BFS (*Breadth First Search*) partiendo por las soluciones que tienen índice de dominación 0. Todas las soluciones visitadas se le es reducido el índice de dominación en uno. Si llega a 0 se añade al frente de Pareto que se está formando. Luego se comienza el proceso nuevamente partiendo por las soluciones a las cuales su índice se redujo a 0.

```

1 def non_dominated_sort(self, scores: List[List[float]]) :
2   # fronts almacena los frentes
3   #(i.e. fronts[i] es el frente de rango i)
4   fronts: List[List[int]] = [[]]
5
6   # domination_rank en i indica la cantidad de soluciones
7   # que dominan a la solucion i
8   domination_rank = [0] * len(scores)

```

```

9      # dominated_scores en i alamacena las soluciones dominadas
10     # por la solucion i
11     dominated_scores = [list() for _ in scores]
12
13
14     # revisa todo par de soluciones y se establece
15     # quien dominan a quien
16     for i, score_i in enumerate(scores):
17         for j, score_j in enumerate(scores):
18             if self._improves(score_i, score_j):
19                 dominated_scores[i].append(j)
20             elif self._improves(score_j, score_i):
21                 domination_rank[i] += 1
22             if domination_rank[i] == 0:
23                 fronts[0].append(i)
24
25     # de acuerdo a la informacion sobre quienes
26     # se dominan, forma todos los frentes
27     front_rank = 0
28     while len(fronts[front_rank]) > 0:
29         next_front = []
30         for i in fronts[front_rank]:
31             for dominated in dominated_scores[i]:
32                 domination_rank[dominated] -= 1
33                 if domination_rank[dominated] == 0:
34                     next_front.append(dominated)
35         front_rank += 1
36         fronts.append(next_front)
37
38     return fronts[:-1]

```

Ejemplo de código 3.2: Non Dominated Sorting

Crowding Distance

Se sigue la idea del algoritmo propuesto en (2.2). Dado m métricas a evaluar se realizan m iteraciones, donde en cada una se ordena un frente de rango k de acuerdo a la metrica m_i . Las soluciones que con respecto a la métrica m_i tienen el mínimo y mayor valor se les asigna distancia infinita y luego se calculan los valores intermedios. En Deb, Pratap y col. 2002 requieren que las componentes de los vectores solución esten normalizadas antes de ejecutar *Crowding Distance*, en nuestra implementación utilizamos *feature scaling* para normalizar los valores entre 0 y 1.

```

1 def crowding_distance(
2     self, scores: List[List[float]], front: List[int], index: int
3 ) -> float:
4     # Crowding distance usa los vectores normalizados.

```

```

5   # Se aplica feature scaling para llevar cada
6   # componente de los vectores al rango [0, 1]
7   scaled_scores = feature_scaling(scores)
8
9   crowding_distances: List[float] = [0 for _ in scores]
10  for m in range(len(self._maximize)):
11      # Se ordena de acuerdo a la métrica m
12      front = sorted(front, key=lambda x: scores[x][m])
13
14      # Se establecen los extremos como infinitos
15      crowding_distances[front[0]] = math.inf
16      crowding_distances[front[-1]] = math.inf
17
18      # Valores de todas las soluciones con respecto a m
19      m_values = [scaled_scores[i][m] for i in front]
20      scale: float = max(m_values) - min(m_values)
21      if scale == 0:
22          scale = 1
23      for i in range(1, len(front) - 1):
24          crowding_distances[i] += (
25              scaled_scores[front[i + 1]][m] - scaled_scores[front[i - 1]][m]
26          ) / scale
27
28  return crowding_distances[index]

```

Ejemplo de código 3.3: Crowding Distance Sorting

3.1.2. Otros cambios

Además de la adición de la clase NSPGE, la infraestructura interna de AutoGOAL sufrió ciertos cambios para permitir la optimización multiobjetivo:

- La clase AutoML retorna el conjunto de mejores soluciones encontradas y su evaluación en vez de la mejor.
- El algoritmo base que explora el espacio de búsqueda utiliza el concepto de *Pareto dominación* para comparar la evaluación de las soluciones.
- En cada iteración se almacena los k mejores individuos encontrados y no el más apto respecto a una sola métrica.
- El algoritmo se detiene cuando no encuentra ninguna solución mejor que alguna de las k mejores.
- La respuesta del algoritmo es F^0 el frente de rango 0 de las mejores soluciones encontradas durante todo el ciclo de ejecución.

3.2. Aplicación de AutogoaL Multiobjetivo

AutoGOAL Multiobjetivo mantiene la misma interfaz que AutoGOAL. Para utilizar esta nueva ordenación se necesita señalar el algoritmo de búsqueda a utilizar con el parámetro `search_algorithm`. Además es necesario definir una métrica que retorne 2 o más valores en `score_metric` y especificar por cada una si se desea maximizar o minimizar utilizando el parámetro `maximize`.

```

1 from autogoaL.datasets import cars
2 from autogoaL.kb import (
3     MatrixContinuousDense,
4     Supervised,
5     VectorCategorical
6 )
7 from autogoaL.ml import AutoML
8 from autogoaL.search import NSPESearch
9
10 automl = AutoML(
11     input=(MatrixContinuousDense, Supervised[VectorCategorical]),
12     output=VectorCategorical,
13     search_algorithm=NSPESearch, # Algoritmo de Búsqueda
14     score_metric=accuracy_vs_time, # Métrica de varios criterios
15     maximize=(True, True), # Componentes de la métrica a maximizar
16 )
17
18 # Entrena el modelo
19 x, y = cars.load()
20 automl.fit(x, y)
21
22 # Imprime todas las soluciones encontradas
23 print(automl.best_pipeline_)
```

Ejemplo de código 3.4: Utilizando NPSGE

3.2.1. Métricas

La definición de las métricas es similar a como se definen en AutoGOAL estándar con la diferencia de que estas retornan tuplas de elementos.

```

1 from time import time
2 import autogoaL.ml.metrics as metrics
3
4 def acc_vs_train_time(*args, **kwargs):
5     t1 = time()
6     acc = metrics.accuracy(*args, **kwargs)
7     t2 = time()
8     score = [acc, t1 - t2]
```

```
9     return score
```

Ejemplo de código 3.5: Ejemplo de métrica: accuracy contra tiempo

Capítulo 4

Experimentación y Resultados

En este capítulo se evaluan los resultados obtenidos por AutoGOAL Multiobjetivo tras aplicarse sobre tres corpus de datos utilizando los pares de métricas precisón contra recobrado y *f-score* contra tiempo de entrenamiento:

- Precisión es una métrica que mide las muestras clasificadas correctamente entre todas las muestras extraídas.
- Recobrado mide las muestras clasificadas correctamente sobre todas las extraídas.
- *F-score* es el promedio harmónico entre precisión y recobrado.
- *Tiempo de entrenamiento* es un estimado que mide cuanto tiempo toma para una flujo de Aprendizaje Automático en realizar el proceso de entrenamiento.

Se utiliza el par de metricas precisión y recobrado pues son crietrios que pueden tener comportamientos variados. Existen problemas donde es posible maximizar ambas y otros donde es obligatorio realizar consesiones entre estas tal que no exista una mejor solución sino un conjutno de estas.

Se utilza *f-score* contra tiempo de entrenamiento para probar si es posible dirigir efectivamente la búsqueda maximizando la relevancia y minimizando el tiempo de ejecución. Es posible utilzar también precisión y recobrado junto con tiempo de entrenamiento pero con el objetivo de mantener estos primeros experimentos más sencillos se utiliza *f-score*.

4.1. Corpus de Evaluación

Se utilizan tres corpus de datos para comprobar el comportamiento del sistema cuando optmiza para varias métricas simultáneamente. Los corpus se escogen de dis-

tintos tamaños y cada uno representa versiones distintas del problema de aprendizaje supervisado.

4.1.1. CARS

Cars (Dua y Graff 2017) es un corpus que representa un conjunto de carros con ciertas características catalogadas cualitativamente (ver 4.1). El objetivo con este corpus es aprender a clasificar los carros de acuerdo a sus atributos en inaceptable, aceptable, bueno o muy bueno (4.2). El dataset no tiene valores desconocidos.

Atributos	Valores
buying	v-high, hihg, med, low
maintenance	v-high, hihg, med, low
doors	2, 3, 4, 5-more
persons	2, 4, more
lug_boot	small, med, big
safety	low, med, high

Tabla 4.1: Tipos de Atributos en Cars

Clases	N	%
unacc	1210	70.023 %
acc	384	22.222 %
good	69	3.993 %
v-good	65	3.762 %

Tabla 4.2: Distribución de clases en Cars

4.1.2. HAHA

Humor Analysis based on Human Annotation (Chiruzzo y col. 2019) representa un conjunto de datos que contiene *tweets* en español en texto plano codificado en utf-8. Se quiere construir un clasificador que pueda catalogarlos correctamente entre si son humorísticos o no. Contiene un total de 30000 tweets donde se utilizan 24000 para entrenamiento y 6000 para evaluación.

	Entrenamiento	Evaluación	Total
Tweets	24000	6000	30000
Graciosos	9253	2342	11595
No graciosos	14 757	3 658	18 405
Puntuación Promedio	1305	275	1580

Tabla 4.3: Distribución de clases en HAHA

4.1.3. MEDDOCAN

Spanish Clinical Case Corpus - Medical Document Anonymization (MEDDOCAN, Marimon y col. 2019) representa colección de 1000 casos clínicos y sus anotaciones PHI. Cada uno está conformado con aproximadamente de 33 oraciones o 495 palabras. Los caso clínico están codificados en texto plano en utf-8 y las anotaciones están en formato BRAT. El objetivo es predecir dichas anotaciones.

4.2. Hardware

Los experimentos fueron ejecutados en un equipo con las siguientes propiedades: CPU AMD R5 3550h y 32 GB de RAM.

4.3. Resultados y Análisis

En esta sección se muestran los resultados obtenidos tras aplicar AutoGOAL Multiojetivo a los distintos corpus. En las gráficas solo se tiene en cuenta la puntuación de relevancia obtenida durante entrenamiento y no con respecto a los datos de prueba. Los puntos grises marcan las mejores soluciones encontradas en cada generación. La tonalidad de los puntos varía de acuerdo en que iteración del algoritmo fue encontrado; a medida que avanzan las generaciones estos se oscurecen. Los puntos marcados con cruces rojas representan las mejores soluciones encontradas por el algoritmo y conforman una aproximación del frente de Pareto.

En cada conjunto de datos se utilizaron parámetros distintos debido fundamentalmente a que cada corpus representa problemas con diferente complejidad y cantidad de datos.

4.3.1. Cars

La aplicación de AutoGOAL en Cars utiliza una población total de 40 individuos por generación, 1 hora de tiempo máximo y 10 segundos de tiempo límite por *pipeline*. Se utilizaron las implementaciones de *f-score*, *precision* y *recall* de Scikit-Learn (Pedregosa y col. 2011) con el promedio *weighted* utilizado para problemas de clasificación multiclas que pueden tener un desbalance en las clases existentes.

F-Score contra Tiempo de Entrenamiento

La mayoría de las soluciones encontradas componen un frente de Pareto con una forma extraña aparentemente cóncava pero bien distribuida sobre el espacio (ver

figura 4.1). El sistema encuentra múltiples soluciones de muy buen *f-score* la mayoría dirigidas a minimizar el tiempo de entrenamiento.

Los algoritmos que logran una puntuación de 1.0 tiene en común que todos utilizan *Support Vector Machines* (SVC). En el rango de 0.7 a 0.9 se encuentran pipelines con distintas técnicas como *Nearest Centroid*, *Multinomial Naive Bayes*, *Bernoulli Naive Bayes* y SVC líneal.

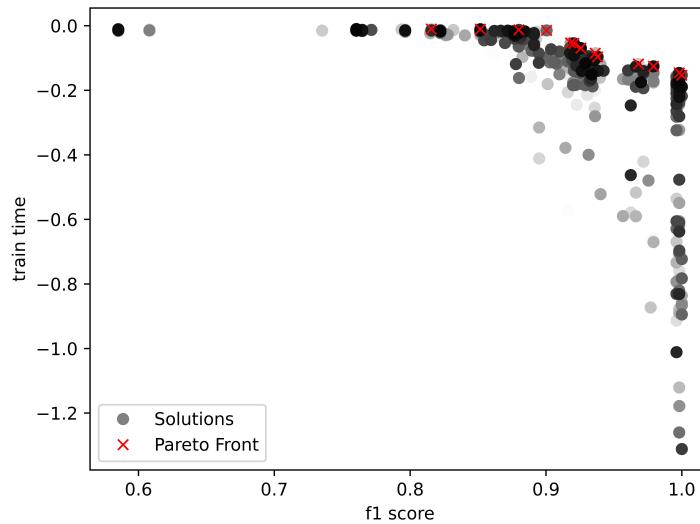


Figura 4.1: Cars: F-score contra Tiempo de Entrenamiento

Precisión contra Recobrado

Durante esta prueba ocurre un hecho interesante, ni precisión ni recobrado entrán en conflicto y por tanto no es necesario hacer concesiones entre una o la otra durante la optimización. Se puede observar en la figura 4.2 que el frente de Pareto esta constituido por un sólo punto donde se maximizan precisión y recobrado. También se nota como las soluciones van mejorando con el transcurso de las iteraciones.

AutoGOAL encontró 5 soluciones compuestas entre otras técnicas por SVC con un rendimiento perfecto. En este caso hay flujos más complejos con respecto a la prueba anterior pues no se discrimina por tiempo de entrenamiento.

4.3.2. HAHA

Se utilizó una población total de 40 individuos, 8 horas de tiempo máximo y 10 segundos por evaluación. HAHA es un problema de clasificación binaria y las métricas

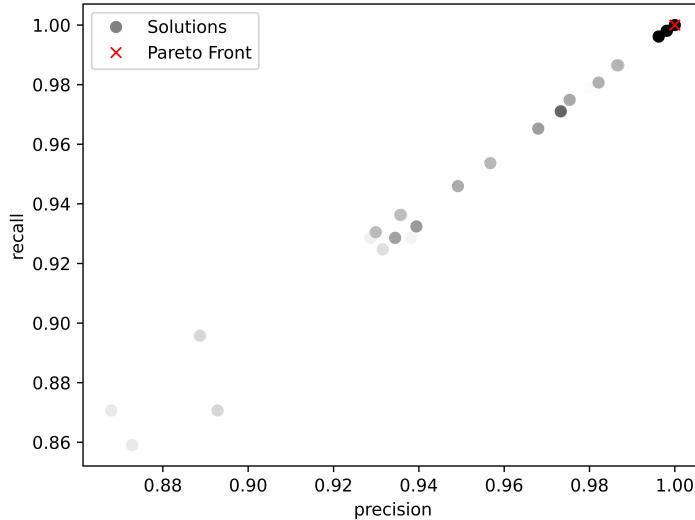


Figura 4.2: Cars: Precisión contra Recobrado

f-score, precisión y recobrado son las implementadas en Scikit-Learn (Pedregosa y col. 2011) utilizando promedio binario.

F-Score contra Tiempo de Entrenamiento

En esta prueba el frente obtenido (fig. 4.3) posee una forma aparentemente similar al de Cars (fig. 4.1). Los soluciones de AutoGOAL están conformados por dos algoritmos principales. Los que contienen una precisión sobre los 0.5 pero son más rápidos están conformados utilizando técnicas de *Hashing Vectorizer* y *Nearest Centroid* mientras que los de mayor precisión están compuestos por *CountVectorizer* y regresores logísticos.

En la esquina superior izquierda se ve como AutoGOAL busca soluciones que tienen f-score 0, un desperdicio de tiempo de cómputo pues el sistema es agnóstico al significado de las métricas. Una posible mitigación a esto es definir no solo las métricas a evaluar si no establecer relaciones entre estas tal como: *if f-score == 0 → retorna -∞*. Estas restricciones reducen el espacio objetivo y tienen un impacto directo en la búsqueda dando aún más expresividad a los usuarios de AutoGOAL.

Otra de las cosas a notar es que hay algoritmos que toman 10 segundos para ejecutarse y posiblemente las soluciones del sistema se vean limitadas por el tiempo máximo asignado a cada *pipeline*. Se realiza una segunda prueba extendiendo esta restricción hasta 3 minutos con el fin de verificar si aumentando el espacio de objetivo se puede obtener una representación distinta del frente de Pareto. En este caso la

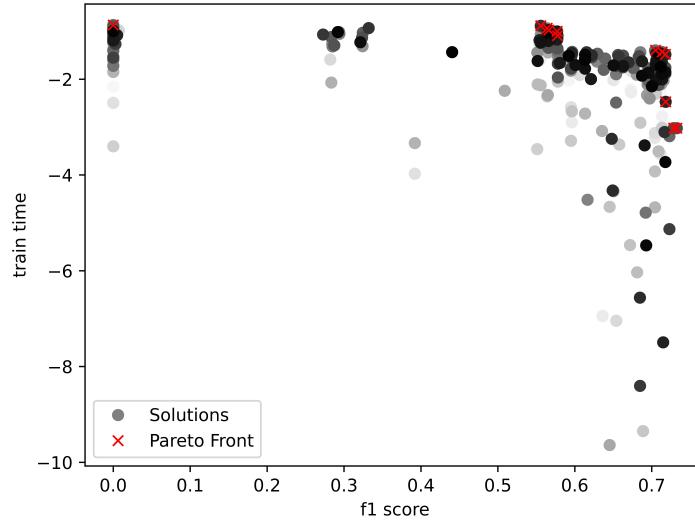


Figura 4.3: HAHA: F-score contra Tiempo de Entrenamiento (10 segundos máx)

estructura extraña del frente se mantiene excepto por la aparición de una nueva solución que destaca sobre el resto con un *f-score* de 0.76 pero con un tiempo de entrenamiento de 31 segundos utilizando un *Count Vectorizer* sin tokenizar y un clasificador SGD.

Precisión contra Recobrado

Contrario a lo que muestra la evaluación de Cars con respecto a precisión y recobrado, durante esta prueba no es posible encontrar un flujo que maximice ambas métricas y es necesario hacer concesiones entre una y la otra. El frente, como se observa en la figura 4.5 tiene una forma convexa, consecuencia del conflicto que se crea al optimizar para estas métricas en este conjunto de datos.

Los flujos obtenidos durante la ejecución de esta fase presentan mucha mayor diversidad con respecto a todas las pruebas anteriores. Los flujos están compuesto en su mayoría por una tupla de técnicas de Aprendizaje: un método de vectorización y un clasificador.

Cuando se optimiza con tiempo máximo de tres minutos (ver figura 4.6) no se muestra ningún cambio notable con respecto a la anterior prueba.

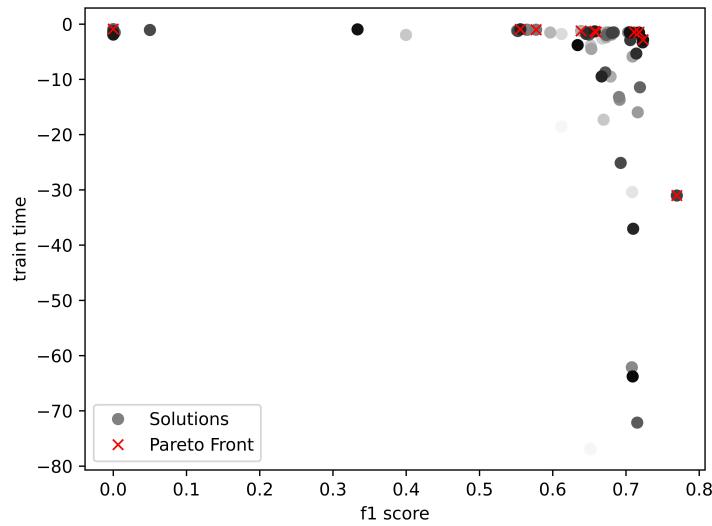


Figura 4.4: HAHA: F-score contra Tiempo de Entrenamiento (3 minutos máx)

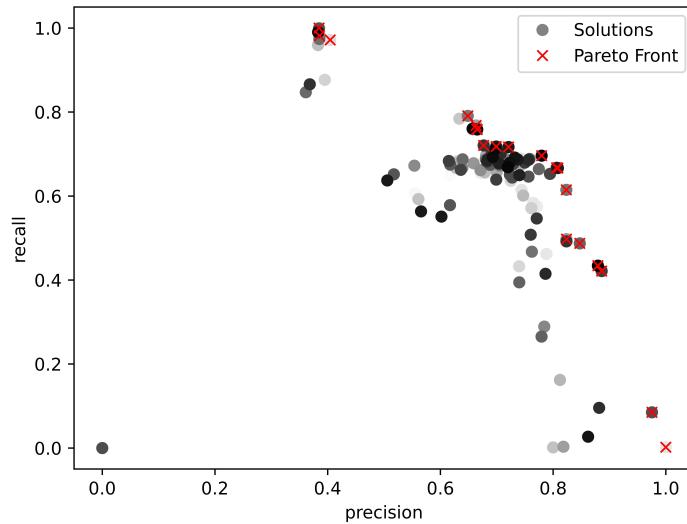


Figura 4.5: HAHA: Precisión contra Recobrado (10 segundos máx)

4.3.3. MEDDOCAN

Este corpus representa el conjunto de datos más grande de los tres y fue necesario aumentar el tiempo de evaluación por flujo a 10 minutos con el fin de encontrar

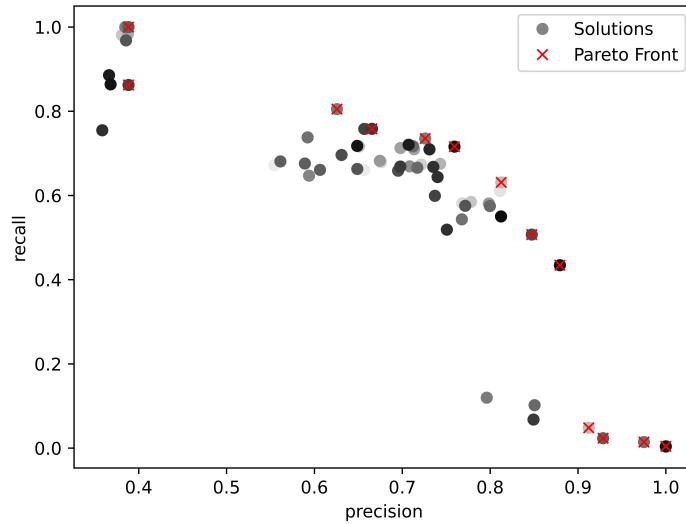


Figura 4.6: HAHA: Precisión contra Recobrado (3 minutos máx)

soluciones válidas, estas tomando tiempos de entre 2 y 6 minutos. Las pruebas fueron ejecutadas con una población total de 50 individuos, 10 horas de tiempo máximo y 10 minutos por evaluación. Las implementaciones de f -score, precisión y recobrado son las implementaciones del módulo de Python *meddocan*. El entrenamiento en esta prueba resultaron ser más lentos y no fue posible realizar muchas iteraciones por lo que no es posible apreciar completamente los frentes.

F-Score contra Tiempo de Entrenamiento

En esta prueba se ve un comportamiento similar a las anteriores como puede observar en la figura 4.7 un poco más acentuado en lo que se refiere a métricas de tiempo de entrenamiento pues la diferencia pasa de ser de segundos a minutos. Existe una solución con un tiempo de 160 segundos y una precisión de 0.83 contra una de 230 segundos y una precisión cercana a 0.9. Existe variedad de técnicas y tamaño en los flujos generados .

Precisión contra Recobrado

En esta prueba, representada en la figura 4.8, debido a la poca cantidad de iteraciones se obtiene una representación muy pobre del frente de Pareo. No queda claro si estamos frente a un caso parecido al de precisión contra recobrado de Cars (4.2) donde las métricas no entran en conflicto y es posible optimizar para ambas, o mas

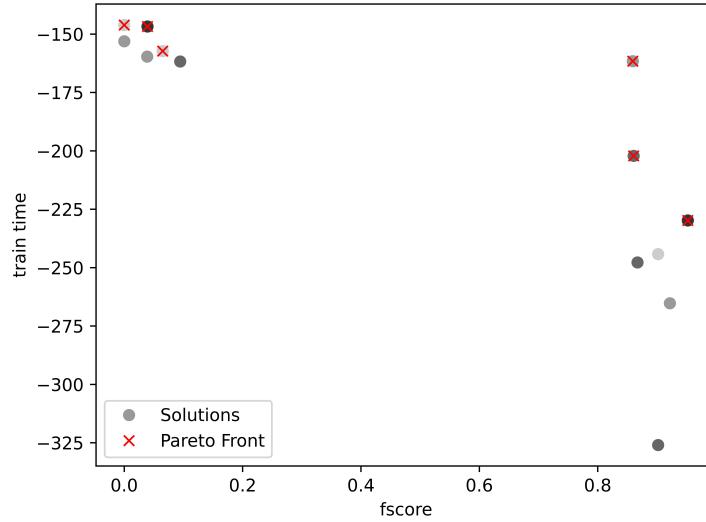


Figura 4.7: MEDDOCAN: F-Score contra Tiempo de Entrenamiento

parecido al caso en HAHA (4.5) donde es obligatorio realizar concesiones.

Un dato interesante de esta prueba en comparación a cuando se optimiza *f-score* contra tiempo de entrenamiento es que esta realizo menos iteraciones en el mismo rango de tiempo, visible por la diferencia de puntos graficados entre ambas. Esto sugiere que al optimizar utilizando el tiempo como una métrica hace que el sistema priorize flujos rápidos de entrenar afectando positivamente la velocidad de búsqueda de AutoGOAL.

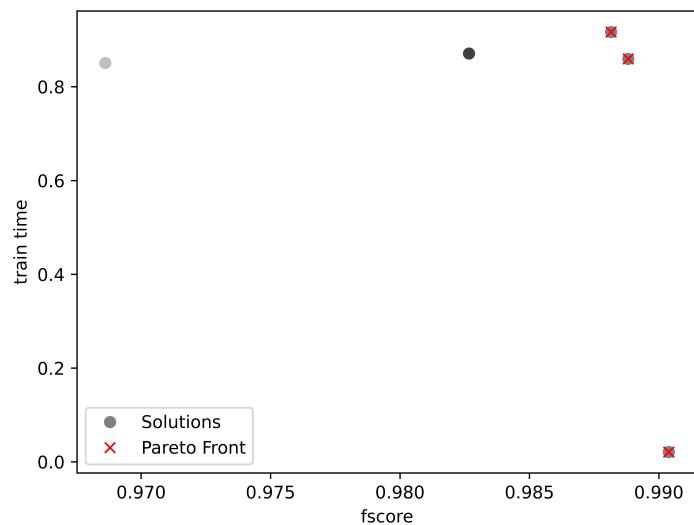


Figura 4.8: MEDDOCAN: Precisión contra Recobrado

Conclusiones

La adición de optimización multiobjetivo a problemas AutoML incrementa exponencialmente sus usos. Los usuarios obtienen una forma efectiva de dirigir la búsqueda sin tener que participar activamente en esta. Problemas previamente sin resolución por sistemas AutoML, pueden ser atacados.

Separar las componentes del sistema tal que trabajen en espacios separados posibilita la integración intuitiva y no invasiva de este mecanismo a los sistemas de AutoML. Además la utilización de algoritmos evolutivos multiobjetivo garantiza soluciones igualmente distribuidas sobre el frente de Pareto, no importa cuan extraña sea su forma.

Autogoal Multiobjetivo, una adaptación de la propuesta, es muestra de integración sencilla. La implementación mantiene la misma interfaz sencilla de AutoGOAL estándar que permite su fácil uso a usuarios de cualquier ámbito de experiencia. Se invita a utilizar esta nueva característica de AutoGOAL.

Los experimentos muestran resultados prometedores y el beneficio que significa la optimización en paralelo de múltiples criterios incluso para problemas que solo utilizan una sola métrica. Abre interrogantes interesantes sobre como se afectarían experimentos previamente realizados con AutoGOAL después de la adición de una métrica adicional como tiempo de entrenamiento o complejidad del flujo.

Recomendaciones

Se identifican varias líneas de investigación y experimentos a realizar para verificar el alcance y eficacia de la propuesta y su implementación en AutoGOAL.

- Investigar en profundidad como la propuesta pudiera adaptarse a sistemas que no utilizan algoritmos evolutivos para definir su espacio de búsqueda tales como los sistemas bayesianos o de aprendizaje por refuerzo.
- Probar la implementación utilizando algoritmos de Aprendizaje No Supervisado.
- Replicar las pruebas realizadas con AutoGOAL en Estevez-Velarde y col. 2020 utilizando como métrica adicional al tiempo de entrenamiento y e investigar los resultados obtenidos.
- Mejorar el algoritmo multiobjetivo en el que se basa la propuesta. NSGA-II sufre un defecto característico de muchos algoritmos multiobjetivos cuya rendimiento se ve crecientemente afectado mientras se eleva el número de métricas. Es posible mejorar el sistema utilizando como base algoritmos evolutivos multiobjetivos por descomposición tales como NSGA-III (Deb y Jain 2013) que no sufre de este defecto.
- Crear nuevas métricas multiobjetivos que analizan *interpretabilidad* y *resistencia* a perturbaciones de los datos. Evaluar los resultados con corpus de datos relevantes.

Bibliografía

- Audet, C., Savard, G. & Zghal, W. (2010). A mesh adaptive direct search algorithm for multiobjective optimization. *European Journal of Operational Research*, 204(3), 545-556 (vid. pág. 14).
- Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press. (Vid. pág. 14).
- Badue, C., Guidolini, R., Carneiro, R. V., Azevedo, P., Cardoso, V. B., Forechi, A., Jesus, L., Berriel, R., Paixao, T. M., Mutz, F. y col. (2021). Self-driving cars: A survey. *Expert Systems with Applications*, 165, 113816 (vid. pág. 1).
- Barocas, S., Hardt, M. & Narayanan, A. (2017). Fairness in machine learning. *Nips tutorial*, 1, 2 (vid. pág. 15).
- Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. (2011). Algorithms for hyperparameter optimization. *Advances in neural information processing systems*, 24 (vid. pág. 7).
- Bergstra, J., Yamins, D., Cox, D. D. y col. (2013). Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. *Proceedings of the 12th Python in science conference*, 13, 20 (vid. pág. 7).
- Björnson, E., Jorswieck, E. y col. (2013). Optimal resource allocation in coordinated multi-cell systems. *Foundations and Trends® in Communications and Information Theory*, 9(2–3), 113-381 (vid. pág. 11).
- Boehmke, B. & Greenwell, B. (2019). *Hands-on machine learning with R*. Chapman; Hall/CRC. (Vid. pág. 9).
- Cai, H., Chen, T., Zhang, W., Yu, Y. & Wang, J. (2018). Efficient architecture search by network transformation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1) (vid. pág. 10).
- Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, 785-794 (vid. pág. 8).
- Chiruzzo, L., Castro, S., Etcheverry, M., Garat, D., Prada, J. J. & Rosá, A. (2019). Overview of haha at iberlef 2019: Humor analysis based on human annotation. *IberLEF@ SEPLN* (vid. pág. 31).

- Consuegra-Ayala, J. P., Gutiérrez, Y., Almeida-Cruz, Y. & Palomar, M. (2022). Intelligent ensembling of auto-ML system outputs for solving classification problems. *Information Sciences*, 609, 766-780 (vid. pág. 2).
- Custódio, A. L., Madeira, J. A., Vaz, A. I. F. & Vicente, L. N. (2011). Direct multi-search for multiobjective optimization. *SIAM Journal on Optimization*, 21(3), 1109-1140 (vid. pág. 14).
- Deb, K. & Jain, H. (2013). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE transactions on evolutionary computation*, 18(4), 577-601 (vid. págs. 15, 41).
- Deb, K., Pratap, A., Agarwal, S. & Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on evolutionary computation*, 6(2), 182-197 (vid. págs. 8, 14, 16, 19, 24, 26).
- Dellnitz, M., Schütze, O. & Hestermeyer, T. (2005). Covering Pareto sets by multi-level subdivision techniques. *Journal of optimization theory and applications*, 124(1), 113-136 (vid. pág. 14).
- de Sá, A. G., Pinto, W. J. G., Oliveira, L. O. V. & Pappa, G. L. (2017). RECIPE: a grammar-based framework for automatically evolving classification pipelines. *European Conference on Genetic Programming*, 246-261 (vid. pág. 8).
- Dua, D. & Graff, C. (2017). UCI Machine Learning Repository. <http://archive.ics.uci.edu/ml>. (Vid. pág. 31)
- Ellefsen, K. O., Lepikson, H. A. & Albiez, J. C. (2017). Multiobjective coverage path planning: Enabling automated inspection of complex, real-world structures. *Applied Soft Computing*, 61, 264-282 (vid. pág. 11).
- Emmerich, M., Beume, N. & Naujoks, B. (2005). An EMO algorithm using the hypervolume measure as selection criterion. *International Conference on Evolutionary Multi-Criterion Optimization*, 62-76 (vid. pág. 14).
- Emmerich, M. & Deutz, A. H. (2018). A tutorial on multiobjective optimization: fundamentals and evolutionary methods. *Natural computing*, 17(3), 585-609 (vid. pág. 13).
- Emmerich, M., Yang, K., Deutz, A., Wang, H. & Fonseca, C. M. (2016). A multicriteria generalization of bayesian global optimization. *Advances in Stochastic and Deterministic Global Optimization* (pp. 229-242). Springer. (Vid. pág. 14).
- Erickson, N., Mueller, J., Shirkov, A., Zhang, H., Larroy, P., Li, M. & Smola, A. (2020). Autogluon-tabular: Robust and accurate automl for structured data. *arXiv preprint arXiv:2003.06505* (vid. pág. 10).
- Erol, K., Hendler, J. A. & Nau, D. S. (1994). UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. *Aips*, 94, 249-254 (vid. pág. 11).
- Estevez-Velarde, S., Piad-Morffis, A., Gutiérrez, Y., Montoyo, A., Munoz, R. & Almeida-Cruz, Y. (2020). Solving heterogeneous automl problems with AutoGOAL.

- ICML Workshop on Automated Machine Learning (AutoML@ ICML)* (vid. págs. 2, 8, 23, 41).
- Feurer, M. & Hutter, F. (2019). Hyperparameter optimization. *Automated machine learning* (pp. 3-33). Springer, Cham. (Vid. pág. 1).
- Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M. & Hutter, F. (2015). Efficient and robust automated machine learning. *Advances in neural information processing systems*, 28 (vid. pág. 7).
- Fleming, P. J., Purshouse, R. C. & Lygoe, R. J. (2005). Many-Objective Optimization: An Engineering Design Perspective. En C. A. Coello Coello, A. Hernández Aguirre & E. Zitzler (Eds.), *Evolutionary Multi-Criterion Optimization* (pp. 14-32). Springer Berlin Heidelberg. (Vid. pág. 15).
- Fortuna, P. & Nunes, S. (2018). A survey on automatic detection of hate speech in text. *ACM Computing Surveys (CSUR)*, 51(4), 1-30 (vid. pág. 1).
- Ganesan, T., Elamvazuthi, I., Shaari, K. Z. K. & Vasant, P. (2013). Hypervolume-driven analytical programming for solar-powered irrigation system optimization. *Nostradamus 2013: Prediction, Modeling and Analysis of Complex Systems* (pp. 147-154). Springer. (Vid. pág. 11).
- Hillermeier, C. y col. (2001). *Nonlinear multiobjective optimization: a generalized homotopy approach* (Vol. 135). Springer Science & Business Media. (Vid. pág. 14).
- Hutter, F., Hoos, H. H. & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. *International conference on learning and intelligent optimization*, 507-523 (vid. pág. 7).
- Jin, H., Song, Q. & Hu, X. (2018). Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, 9 (vid. pág. 7).
- Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4, 237-285 (vid. pág. 10).
- Knowles, J. (2006). ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems. *IEEE Transactions on Evolutionary Computation*, 10(1), 50-66 (vid. pág. 16).
- Komer, B., Bergstra, J. & Eliasmith, C. (2014). Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. *ICML workshop on AutoML*, 9, 50 (vid. pág. 7).
- Kourou, K., Exarchos, T. P., Exarchos, K. P., Karamouzis, M. V. & Fotiadis, D. I. (2015). Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal*, 13, 8-17 (vid. pág. 1).
- Kuhn, H. W. & Tucker, A. W. (2014). Nonlinear programming. *Traces and emergence of nonlinear programming* (pp. 247-258). Springer. (Vid. pág. 14).
- LeDell, E. & Poirier, S. (2020). H2o automl: Scalable automatic machine learning. *Proceedings of the AutoML Workshop at ICML, 2020* (vid. pág. 9).

- Liu, J., Dolan, P. & Pedersen, E. R. (2010). Personalized news recommendation based on click behavior. *Proceedings of the 15th international conference on Intelligent user interfaces*, 31-40 (vid. pág. 1).
- Marimon, M., Gonzalez-Agirre, A., Intxaurrendo, A., Rodriguez, H., Martin, J. L., Villegas, M. & Krallinger, M. (2019). Automatic De-identification of Medical Texts in Spanish: the MEDDOCAN Track, Corpus, Guidelines, Methods and Evaluation of Results. *IberLEF@ SEPLN*, 618-638 (vid. pág. 32).
- Mégane, J., Lourenço, N. & Machado, P. (2021). Probabilistic grammatical evolution. *European Conference on Genetic Programming (Part of EvoStar)*, 198-213 (vid. págs. 8, 23).
- Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K. & Galstyan, A. (2021). A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)*, 54(6), 1-35 (vid. pág. 2).
- Miettinen, K. (2012). *Nonlinear multiobjective optimization* (Vol. 12). Springer Science & Business Media. (Vid. pág. 12).
- Mohr, F., Wever, M. & Hüllermeier, E. (2018). ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8), 1495-1515 (vid. pág. 11).
- Olson, R. S. & Moore, J. H. (2016). TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning. En F. Hutter, L. Kotthoff & J. Vanschoren (Eds.), *Proceedings of the Workshop on Automatic Machine Learning* (pp. 66-74). PMLR. https://proceedings.mlr.press/v64/olson_tpot_2016.html. (Vid. págs. 8, 16)
- Paria, B., Kandasamy, K. & Póczos, B. (2020). A flexible framework for multi-objective bayesian optimization using random scalarizations. *Uncertainty in Artificial Intelligence*, 766-776 (vid. pág. 13).
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V. y col. (2011). Scikit-learn: Machine learning in Python. *the Journal of machine Learning research*, 12, 2825-2830 (vid. págs. 7, 8, 32, 34).
- Pfisterer, F., Coors, S., Thomas, J. & Bischl, B. (2019). Multi-objective automatic machine learning with autoxgboostmc. *arXiv preprint arXiv:1908.10796* (vid. pág. 16).
- Plana, S., Memeti, S. & Kolodziej, J. (2019). Customizing Pareto simulated annealing for multi-objective optimization of control cabinet layout. *2019 22nd International Conference on Control Systems and Computer Science (CSCS)*, 78-85 (vid. pág. 11).
- Schütze, O., Dell'Aere, A. & Dellnitz, M. (2005). On Continuation Methods for the Numerical Treatment of Multi-Objective Optimization Problems. En J. Branke, K. Deb, K. Miettinen & R. E. Steuer (Eds.), *Practical Approaches to Multi-Objective Optimization* (pp. 1-15). Schloss Dagstuhl – Leibniz-Zentrum für In-

formatik. <https://doi.org/10.4230/DagSemProc.04461.16>. (Vid. pág. 14)

Keywords: multi-objective optimization, continuation, k-manifolds

- Sharifi, M. R., Akbarifard, S., Qaderi, K. & Madadi, M. R. (2021). A new optimization algorithm to solve multi-objective problems. *Scientific Reports*, 11(1), 1-19 (vid. pág. 15).
- Shirazi, A., Aminyavari, M., Najafi, B., Rinaldi, F. & Razaghi, M. (2012). Thermal-economic-environmental analysis and multi-objective optimization of an internal-reforming solid oxide fuel cell-gas turbine hybrid system. *International Journal of Hydrogen Energy*, 37(24), 19111-19124 (vid. pág. 11).
- Shirazi, A., Najafi, B., Aminyavari, M., Rinaldi, F. & Taylor, R. A. (2014). Thermal-economic-environmental analysis and multi-objective optimization of an ice thermal energy storage system for gas turbine cycle inlet air cooling. *Energy*, 69, 212-226 (vid. pág. 11).
- Thomas, J., Coors, S. & Bischl, B. (2018). Automatic gradient boosting. *arXiv preprint arXiv:1807.03873* (vid. págs. 7, 16).
- Thornton, C., Hutter, F., Hoos, H. H. & Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 847-855 (vid. págs. 1, 6, 7).
- Tovbin, M. (s.f.). Meet TransmogrifAI, open source AutoML that powers einstein predictions. SF Big Analytics Meetup. (Vid. pág. 9).
- Wei, T., Wang, C., Rui, Y. & Chen, C. W. (2016). Network morphism. *International conference on machine learning*, 564-572 (vid. pág. 9).
- Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectivist reinforcement learning. *Machine learning*, 8(3), 229-256 (vid. pág. 10).
- Xu, T., Zhang, P., Huang, Q., Zhang, H., Gan, Z., Huang, X. & He, X. (2018). AttnGAN: Fine-grained text to image generation with attentional generative adversarial networks. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1316-1324 (vid. pág. 1).
- Zhang, Q. & Li, H. (2007). MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6), 712-731 (vid. pág. 15).
- Žilinskas, A. (2013). On the worst-case optimal multi-objective global optimization. *Optimization Letters*, 7(8), 1921-1928 (vid. pág. 14).
- Zitzler, E. & Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4), 257-271 (vid. pág. 14).
- Zoph, B., Vasudevan, V., Shlens, J. & Le, Q. V. (2018). Learning transferable architectures for scalable image recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, 8697-8710 (vid. pág. 10).