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

Resumen en español

Abstract

Resumen en inglés

Índice general

Índice de figuras

Ejemplos de código

Introducción

Aprendizaje de Máquina (conocido en la literatura como *Machine Learning*, ML por sus siglas en inglés) es una rama de la Inteligencia Artificial enfocada en entender y construir programas que "aprendan" y logren buenos resultados en un conjunto de tareas definidas (Mitchell y col. 1990). Este campo vió sus inicios sobre la década de los 50 pero no tuvo auge hasta la primera decada del nuevo milenio con el auge de las nuevas tecnologias que abarataron considerablemente el poder de computo. El Aprendizaje de Máquina comenzó a ocupar espacios donde aplicar algoritmos convencionales resultaba muy díficl o infactible.

Construir un flujo válido de Aprendizaje de Máquina requiere, entre otras cosas, probar con diferentes algoritmos e hyperparámetros. Un proceso automatizable utilizando sistemas de Aprendizajes de Máquinas Automatizado (*Automated Machine Learning*, AutoML). Estos sistemas, en general, reciben como entrada un set de datos y una métrica de evaluación y retornan el flujo de Aprendizaje de Máquina que tenga el mayor rendimiento de acuerdo a dicha métrica. Lo clásico es que es que los flujos producidos se evaluen respecto a una métrica que represente la relevancia de sus resultados.

En el contexto actual, un flujo de Aprendizaje de Máquina que obtenga resultados relevantes ya no es suficiente y se investigan nuevas maneras de lograr un mayor control sobre el flujo final que generan. Una de las propuestas a este problema es permitir la entrada de varias métricas a los sistemas AutoML y optimizar simultáneamente para todas estas.

Motivación

Los modelos de Aprendizaje de Máquina forman cada vez más y más parte de la vida diaria, aplicándose en situaciones directamente relacionadas con la vida de los seres humanos. Es un poblema conocido y estudiado que un modelo puede producir predicciones sesgadas dependiendo a ciertas características de los datos (Mehrabi y col. 2021). Eso es de gran importancia cuando estos datos representan seres humanos y la decisión del modelo tiene impacto directo en un individuo o grupo. No se desea

que el algoritmo juzge por el color de piel o género o región, pero tampoco que sus resultados dejen de ser relevante. Es necesario entonces que el flujo sea capaz de poder optimizar tanto relevancia, como nivel de justeza, un problema multiobjetivo.

También existen casos donde se quiere conocer la *interpretabilidad* de un modelo, que implica entender sin necesidad de ser un experto porque el flujo toma ciertas decisiones; o cuan *robusto* es el sistema, tal que perturbaciones en los datos no cambien radicalmente el resultado de este.

La adición de optimización multibojetivo en AutoML abré las puertas también a métricas que no tienen sentido usarlas individualemente como *precisión* y *recobrado*. *Precisión* mide la proproción 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. Estas métircas usadas por separadas no son represantivas pues basta para tener un recobrado perfecto seleccionar todos los elementos del conjunto de datos, y una precisión casi perfecta seleccionando un pequeño conjunto de elementos. Para lograr un balance entre estas se utiliza *f-score*, que relaciona ambas medidas y permite darle más peso a una o a la otra según determine el investigador, no obstante con optimización multiobjetivo dentro de un framework de AutoML se pueden obtener sistemas ML que optimicen para ambas.

Antecedentes

En el entorno del grupo de Inteligencia Artificial de la Universidad de La Habana se desarrolla AutoGOAL una herramienta de AutoML que permite obtener modelos optimizados con respecto a una sola métrica.

Problemática

Los sistemas AutoML para generar sus soluciones modelan un espacio de búsqueda sobre el que realizan un exploración buscando una solución óptima. Estos espacios son complejos y muchas veces no diferenciables. Optimzizar simúltanemeante para muchos objetivos require de un algoritmo que sea lo más agnóstico posible a este espacio y a la misma vez sea capaz de ofrecer buenos resultados. Se intenta la resolución del problema utilizando algoritmos genéticos que cumplen con estas cualidades.

Objetivo

Objetivo General

Añadir a AutoGOAL la habilidad de resolver problemas de optimización multiobjetivo.

Objetivos Específicos

- Estudiar el estado del arte relacionado con el problema
- Identificar un algoritmo multiobjetivo que aproveche la estructura de AutoGOAL y su implementación de Probabilistic Grammatic Evolution (PGE)
- Estudiar diferencias cuando se resuelve un mismo problema con optimización multibojetivo
- Analizar la efectividad de la solución

Estructura de la Tesis

Capítulo 1

Estado del Arte

El proceso de investigación para formar un flujo de Aprendizaje Automático requiere una experemintación variada donde se combinan dśimeles técnicas como redes neuronales, clasificadores supervistados, algoritmos de agrupamientos entre otras. Además por cada selección de las anteriores se debe optimzar para cada una los hiperparámetros adecuados.

El aprendizje de máquina automatizado es un campo cada vez más creciente cuyo objetivo es encontrar flujos de Aprendizajes Automáticos que sean óptimos respecto a una métrica de evaluación y se adapten a distintos escenarios. Los estudios recientes han llevado a la creación de sistemas de AutoML con espacios de búsqeudas cada vez más complejos y mejores técnicas de selección y modelos de hiperparámetros.

Con el uso de un sistema AutoML el investigador ya no tiene que crear el flujo de Aprendizaje Automático manualmente; puede dedidcarse a otras tareas como el adecuado preprocesamiento de los datos o ingeniería de característcas, no obstante, el proceso de automatización viene con un costo. Cuando al científico no le es suficiente con que el sistema produzca resultados relevantes y desea un mayor control sobre los flujos producidos tal que tengan un buen desempeño con respecto a alguna métrica adicional, los sistmeas AutoML dejan de ser de útiles pués en el estado del arte actual del Aprendizaje Automatizado no existe el soporte para esto.

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 parámteros que resuelven un problema determinado minimizando la dependencia del usuario.

Actualmente los sistemas AutoML se pueden en separar en dos conjuntos dependiendo del problema a resolver. Se encuentran los sistemas orientados a aprendizaje profundo conocido como Búsqueda de Arquitecturas Neuronales (*Neural Arquitectural Search*, NAS) y Auto-ML basado en selección y combinación de modelos que pueden resolver el problema NAS o no. El segundo conjunto intenta resolver el problema de combinación, selección y optimización de hiperparámetros (*Combined Algorith Selection and Hyperparameter Optimization*, CASH) acuñado por Auto-Weka (Thornton y col. 2013). Más formalmente:

Definición 1.1 *Sea* $A = \{a^1, ..., a^k\}$ *un conjunto de algoritmos con espacios de configuración asociados* $\Lambda^1, ..., \lambda^k$. *Sea* D *un conjunto de datos que se divide en* $\{D_t^{(1)}, ..., D_t^{(k)}\}$ *y* $\{D_v^{(1)}, ..., D_v^{(k)}\}$ *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:*

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

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 han habido numerosas propuestas de distintos dominios:

- Optimización Bayesiana(Hutter, Kotthoff y col. 2019)
- Programación Evolutiva (B. Chen y col. 2018)
- 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 itera la búsqueda.

Auto-WEKA (Thornton y col. 2013) Utiliza como algoritmos de aprendizaje las máquinas WEKA, una popular biblioteca de Aprendizaje Automt́ico en Java. Presenta dos optimizadores bayesianos, una propuesta de configuración secuencia de Algortimos basados en Modelos (SMAC por Hutter, Hoos y col. 2011) y un estimador de Parzen con estrucutra de arbol (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 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 búsquda aprendiendo de modelos que funcionaron bien en conjuntos de datos similares y luego un paso de selección de ensemblers 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 bibliteca de aprendizaje subyacente, scikit-learn, utiliza Hyperopt (Bergstra, Yamins y col. 2013) para describir el espacio de búsqueda. Provee una infterfaz de optimziación 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 codficacioón más expresiva de las intuiciones de los expertos con respecto a los valores de los hiperparámetros. Utiliza SMBO (*Sequential Model-based Bayesian Optimization*), consideraando este algoritmo como una componente intercambiable con el fin de que cualquier técninca de búsqueda pueda ser utilizada con cualquier problema de búsqueda.

Auto-Keras (Jin y col. 2018) Un sistema de Auto-ML enfocado en resolver el problema NAS. Está basado en la biblioteca Keras de Python, utilizada para crear redes neuronales. La idea prinicipal de esta propuesta es explorar el espacio mediante la transformación de redes, utilizando un algorimto de Optmización 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 cataloga como un sistema AutoML *single-learner* debido a que solo utiliza un algoritmo de aprendizaje y solamente 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* (T. Chen y Guestrin 2016) debido a su prometedor rendimiento con los paramétros adecuados.

1.1.2. Programación Evolutiva

Los sistemas AutoML basádos en programación evolutiva funcionan creando una población inicial de flujos válidos para luego seleccionar los de mejor rendmiento respecto a cierta métrica y los mejores 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) Utilzando algormiso de aprendizaje de la biblioteca scikit-learn (Pedregosa y col. 2011) junto con el algoritmo xgboost (T. 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) optimzando maximizar exactitud (*accuracy* en la literatura) y minimizando la cantidad de operadores aplicados sobre los flujos de ML por un tema de simplicidad 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 poblacón inicial de flujos. Luego a partir de los mejores individuos de la poblacion le aplican operadores de mutación y cruzamiento que fuerzan el cumplimiento de las reglas de producción de la gramática para generar una nueva población. Utiliza como base la bibliotea scikit-Learn (Pedregosa y col. 2011).

AutoGOAL (Estevez-Velarde y col. 2020) Utiliza una Gramaática Probablísta 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 construir las soluciones. Tiene integrados muchos algorimtos de aprendizaje de diversas bibliotecas de Python y su generalidad permite abarcar problemas de cualquier tipo, incluido problemas de procesamiento de lenguage natural, donde cuenta con algoritmos de 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 le aplica morfismos de red para obtener nuevas redes "vecinas". Se evalúan utilizando una función de evaluación y la que tenga mejor rendimiento se toma como principal y se reptie el proceso. La estrategia de hill climbing que sigue elimina la necesidad de rentrenar las nuevas redes desde cero.

H20 AutoML (LeDell y Poirier 2020) Disponible en varios lenguages, utiliza la biblioteca H20 (Boehmke y Greenwell 2019), una biblioteca de aprendizaje automático. Construye multiples modelos con diferentes juegos de hiperparámetros seleccionados aleatoriamente en base de 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 diferents 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 caracaterísticas automatizada: eventualmente los datos, sin importar del tipo que sean, deben tener representación númerica adecuada (todo un arte de la ciencia 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 optimizción 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. Está 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) Effiecient Arquitectural Search utiliza un meta-controlador basado en aprendizaje por refuerzo (*Reinforcment Learning*, RL, Kaelbling y col. 1996) y el algorimto REINFORCE (Williams 1992) para actualizarlo. En general, este sistema modela el proceso de diseño automático de arquitecutras como un proceso de toma de decisiones secuencial, donde el estado de la red actual y la actuación es la operación de transformación correspondiente. Despueés de una cantidad determinada de pasos, la arquitectura resultante es evaludada 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 dataset pequeño y luego transferir los conocimientos para el entrenamiento de un set más grande, aligerando el costo computacional de aplicarlo directamente sobre un dataset grande.

1.1.5. Modelos Constructivos

Estos métodos exploran el espacio de búsqueda con una forma estructurada pues se definen 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 procesaminto de datos robustos. Construye un ensemble multicapa tomando como entrada la instancia de cada uno de los tipos de modelos predefinidos. Los modelo se entrena de forma secuencial y se añaden 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 MonteCarlo requiere de una definición adecuada de una función que permita un balance entre exploración y explotación. Esta estrategia require un espacio de búsequda donde las decisiones más importantes esten 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 con dos tipos de algoritmos: preprocesadores de datos y de aprendizaje. Sus flujos son un par compuestos por un procesador parametrizado y por 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 simúltaneamente. Es relativamente nueva y un campo activo de investigación.

Ha encontrado abundante campo en la ingeniería y la economía donde muchas veces suelen existir objetivos opuestos. Exsiten muchos aplicaciones en la actualidad que se benefician de aplicar optimziación multiobjetivo, por ejemplo:

- Lograr el 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 distribución adecuada de recursos de radio para satisfacer eficientemente el requerimiento de sus usuarios (Björnson, Jorswieck y col. 2013).
- La inspección de infraestructura compleja es costosa, y muchas veces no es factible cubriendo el área completa. Se busca optimizar la covertura mientras se minimizan los costos (Ellefsen y col. 2017).

Definición 1.2 *Optimización Multibojetivo: Dado m funciones objetivos:* $f_1 : \mathcal{X} \to \mathbb{R}, ..., f_m : \mathcal{X} \to \mathbb{R}$ *que dado un vector en el espacio de decisión* $\mathcal{X}$ *es transformado en un valor de* $\mathbb{R}$*. Se define el PMO como:*

$$\text{mín}\, f_1(x), ..., f_m(x), x \in \mathcal{X}$$

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: Dados dos vectores en el el espacio objetivo,* $x \in \mathbb{R}^m$ *y* $z \in \mathbb{R}^m$*, se dice que x Pareto domina a z (i.e.* $x \prec z$*), si y solo si:*

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

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 técincas 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.

   $$\text{mín} \sum w_i f_i(x), x \in X$$

   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.

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

   Esta enfoque presenta dos dificultades principales: la valores de los ϵ_i, para una adecuada selección requiren 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 $\lambda \in \mathbb{R}^m_{\prec 0}$, donde $\mathbb{R}^m_{\prec 0} = \{x | x \in \mathbb{R}^m, x \prec 0\}$.

   $$\text{mín} \quad \max_{i \in 1,\ldots,m} \lambda_i |f_i(x) - z_i^*|, x \in X$$

   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 de algoritmos geneéticos por descomposición para optener 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 númericos. 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. se puede obtener mas infomración en Hillermeier y col. 2001 y Schütze y col. 2005.

Tambien existen otros metodos 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 númericos 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 algorimtos 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 selccionar 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 invidiuos seguún su indíce 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 cuan cercano es el conjunto actual al frente de Pareto (unario), o cuanto 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 paramtrización. 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 termino Optimización Multiobjetivo se utiliza cuando el número de funciones objetivos son dos o tres. Para un cantidad de métricas 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 enfásis 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 descomposción.

1.3. AutoML y Multiobjetivo

Recientemente ha habido enfásis 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 $M = \{f_1, f_2, ..., f_m\}$, se espera que un sistema AutoML $\mathcal{A}$ retorne un conjunto de flujos, algoritmos o redes $P = \{p_1, p_2, ..., p_k\}$ 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) guíandose por exactitud de las predicciones y número de operadores (i.e. algoritmos de aprendizaje). No obstante el resultado de TPOT no es un conjunto soluciones, sino la solución que mejor rendimiento tuvo 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. Luego la función restultante de aplicar escalarización es optimizada utilizando optimización bayesiana estándar de un sólo criterio. El algoritmo se beneficia de un humano en el proceso de optimización durante la búsqueda especifica través de restricciones a donde se quiere dirigir la búsqueda multiobjetivo. Al estar basados en un sistema AutoML que utiliza un solo algoritmo, el espacio de búsqueda se encuentra limitado y es posble que no incluya configuraciones óptimas.

Capítulo 2

AutoML Multiobjetivo

Se propone como solución al *problema multiobjetivo en AutoML* (1.5) un algoritmo que utilize como caja negra un sistema AutoML que sea capaz de producir flujos respecto a métricas de entrada. El algoritmo selecciona los mejores flujos utilizando un metodo de ordenación multiobjetivo y pide al sistema AutoML que genere y evalue una población a partir de los flujos escogidos como los más aptos. Se repite este paso hasta que se cumpla el criterio de parada, en donde se devuelven las mejores respuestas encontradas hasta el momento con respecto a todas las iteraciones.

2.1. Descripción General

El sistema toma como entrada un sistema de Aprendizaje de Máquina Automatizado $\mathcal{A}$ tal que $\mathcal{A}(D, M, P) = TP$ donde $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ representa un conjunto de datos de entrada, $M = \{f_1, f_2, ..., f_m\}$ un conjunto de métricas a evaluar y $P = \{p_1, p_2, ..., p_k\}$ una serie de modelos de Aprendizaje Automático por los cuales se guía $\mathcal{A}$ para producir una nueva población de flujos TP. El objetivo del sistema es producir una conjutno de flujos de Aprendizaje Automático P que sean una muestra representativa del frente de Pareto utilizando un algoritmo evolutivo multiobjetivo $\mathcal{H}$ tal que $\mathcal{H}(TP) = P$ en cierto número de iteraciones determinado por un criterio de parada.

Algoritmo 2.1: Flujo del Sistema

Entrada: $\mathcal{A}$, D, M
Salida: P

```
1 TP ← A(D, M, ∅) ;          // Se obtiene población inicial aleatoria
2 mientras no se cumplan condiciones de parada hacer
3 |   P = H(TP) ;          // Se extraen los más aptos de una población
4 |   TP = A(D, M, P) ;              // Se genera una nueva población
5 fin
```

2.2. Sistema AutoML

El sistema AutoML necesario para esta propuesta se utiliza como caja negra y debe ser capaz de producir una población inicial aleatoria, al igual que poblaciones basadas en otra subpoblación tal que los nuevos individuos conserven los rasgos de esta subpoblación. Esto último se logra utilizando algoritmos evolutivos que definen operadores de mutación, combinación y selección.

Nuestra propuesta está basada en AutoGOAL (Estevez-Velarde y col. 2020) un sistema de AutoML que modela el espacio de decisión utilizando una Gramática Probabilística Libre del Contexto (*Probabilistic Context Free Grammar*, PCFG).

Las PCFG se definen como una quinterna $PG = (NT, T, S, P, Prob)$ donde NT y T representan los conjuntos disjuntos no vacío de los símbolos no terminales y terminales respectivamente. S es un elemento de NT llamado el axioma que representa el no-terminal inicial tal que expandiendo este se pueden llegar a todas las posibles formas de la gramática. P es el conjunto de reglas de producción que rigen a la grmaática. $Prob$ es un conjunto de probabilidades asocaido con cada regla de la gramática.

Con el fin de aprovechar el espacio de decisión ya definido se utiliza Evolución Gramática Probabilística (*Probabilistic Grammatic Evolution*, PGE, Mégane y col. 2021), un algorimto de Estimación de Distribución (*Estimation of Distribution ALgorithms, EDA*, Larrañaga y Lozano 2001) que remplaza los operadores clásicos de mutación y cruce por un muestreado sobre las probabilidades de distribución de PCFG de acuerdo a las producciones utilizadas por el mejor individuo.

En PCFG, al inicio, las probabilidades están igualmente distribuidas sobre todas las producciones por cada regla de la gramática y se actualizan al aplicar PGE con el individuo más apto donde se alterna entre aumentar ligeramente la probabilidad de las producciones que utilizadas por este y disminuir la probabilidades de los que no fueron utilizadas. Alternar entre estas dos vías evita la utilización del mismo individuo en iteraciones consecutivas balanceando la exploración global con explotación local.

Como se trata de un problema multiobjetivo y no existe el individuo más apto, sino un conjunto de estos las probabilidas se adaptan secuencialmente teniendo en

cuenta cada uno de estos individuos.

2.3. Algoritmo Multiobjetivo

Dado una población de flujos de Aprendizaje Automático es necesario poder elegir los más aptos. Como se habla de un conjunto de individuos, donde pueden haber muchos indiviudos correctos se busca escoger de los mejores de esto una muestra que sea representativa del conjunto total.

Se utiliza NSGA-II (Deb, Pratap y col. 2002) por ser un algoritmo popular en el campo de la optimización multiobjetivo, estar ampliamente probado y tener un buen rendimiento general.

NSGA-II al ser un algoritmo catálogado como un MOEA basado en el frente de pareto (1) y se caracteriza por una ordenació en dos etapas.

En la primera etapa ordena los primeros individuos con respecto a su índice de dominación. Donde dicho ínidice está determinado por la cantidad de soluciones distintas que *Pareto dominan* a esta.

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 $Ind(x) = |Y|$.*

Luego los vectores se agrupan de acuerdo a su índice de dominación de la siguiente manera $\{P^0, ..., P^k\}$ donde $P^i = \{x | Ind(x) = i\}$ y se le conoce como frente de rango i. Idealmente el frente de rango 0 debe ser una muestra representativa al frente de Pareto.

En la segunda etapa los vectores de cada frente obtenido se organizan utilizando *crowding distance* (CD), uno de las contribuciones claves de NSGA-II (Deb, Pratap y col. 2002). El propósito de *crowding distance* es estimar la densidad de las soluciones con respecto sus soluciones vecinas de tal manera que los primeros lugares pertenezcan a los elementos más representativos de dicho frente.

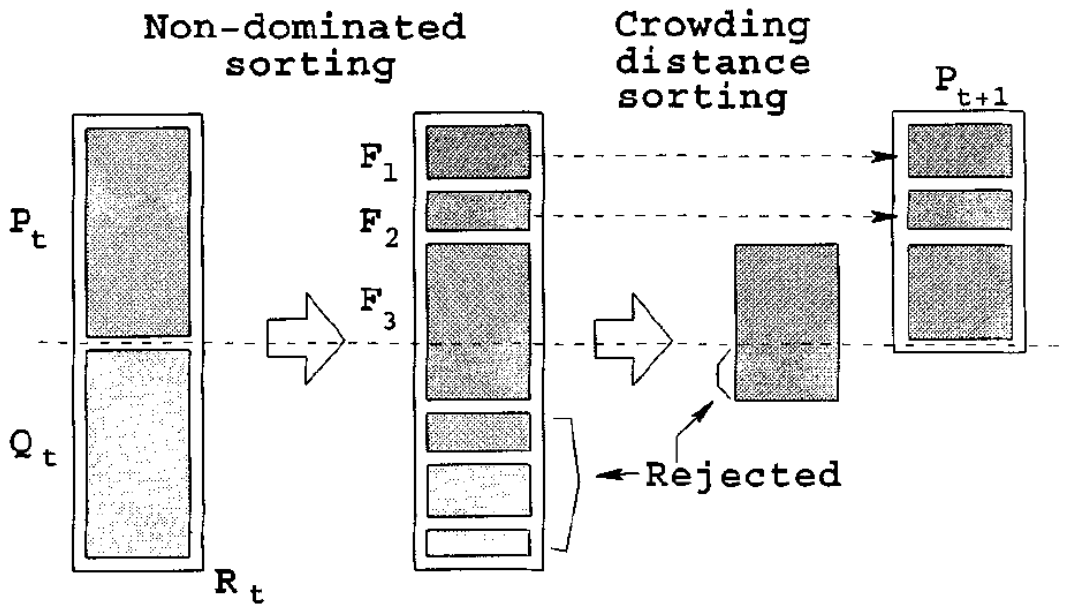
La densidad de una solución se obtiene ordenando las soluciones por cada función objetivo y calculando el promedio de la distancia entre una solución y sus dos adyacentes. Esta distancia resulta siendo el perímetro del cuboide formado usando los vecinos más cercanos como vértices. En el caso de los puntos que representan el mínimo y máximo de al menos alguna función objetivo se les asigna valor infinito. El pseudocódigo del algoritmo se muestra en 2.2.

Algoritmo 2.2: Crowding Distance

```
   // 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 ← 0 ;
 3 fin
 4 para cada funcion objetivo m hacer
 5 |   F ← ordenar(F,m) ;                    // se ordena F con respecto a m
 6 |   F[0].dist ← ∞;
 7 |   F[|F|].dist ← ∞;
 8 |   para i desde 2 hasta |F| − 1 hacer
 9 |   |   F[i].distance = (F[i].distance+(F[i+1].m−F[i−1].m)) / (f_m^max − f_m^min)
10 |   fin
11 fin
12 SF ← ordenar(F,dist) ;                    // se ordena F respecto a CD
```

Luego guíandonos por la figura 2.1 es posible ver el funcionamiento general del algoritmo. Se divide la población inicial en frentes de distinto rango. Luego por cada frente se aplica CD. Según lo que el sistema de AutoML necesite se pasan los subconjutnos. Si un conjutno no se puede pasar completo se garantiza que pasan los elementos más representativos de este gracias a CD.

Figura 2.1: Funcionamineto de NSGA-II



Capítulo 3

Implementación y Experimentos

Se desarrola la propuesta sobre AutoGOAL un sistema de AutoML ideado en el grupo de Inteligencia Artificial de la Universidad de la Habana.

Luego utilizamos el sistema desarrollado sobre dos corupus distintos y con metricas variadas para comprobar el comportamiento de la nueva propuesta.

3.1. Implementación

AutoGOAL logra mantener las características positivas de una poblacióun utilzando PCFG y PGE. Una vez se genera una primera población se escogen los individuos k individuos que según cierta métrica m fueron los más aptos y se aplica PGE sobre cada uno para actualizar el DAG tal que esas producciones tengan mayor probabilidad de escogerse, y por tanto las individuos de las proximas poblaciones tenga más posibilidad de conservar esos traits.

PGE en AutoGOAL se representa una clase que recibe n individuos los ordena y escoge los primeros k con los que actualiza PCFG que se representa como un Sampler. Para escoger los k mejores utiliza la funcion *_inidices_of_the_fittest_* que lo que hace es es producir una lista encabezada por el primero más apto

Se implementa la clase NSPGE (el nombre reminicente a NSGA-II pues se guía por sus mismos principios donde el algoritmo genetico utilzado es PGE) que hereda de PGE y sobre escribe el metodo de ordenar. Esta clase esta preparada para funcionar con todas las métricas que se quieran.

La ordenacion que ocurre aqui, que se cambia ya en vez de tomar los k mas aptos segun una metrica se toman los K mas aptos segun m metricas. La manera de escoger los k mas aptos es ordenarlos primeramente seguún el rango del frente al que pertencen (Non-dominated sorting). Si existe un frente tal que no caben sus soluciones enteras porque la cantidad de inviduos total hasta ahora excede k se ordena utilizando crowding distance para quedarse con los elementos más representativos de

dicho frente.

Mejores índices

```
def _indices_of_fittest(self, fns: List[List[float]]):
  # Se ordenan todas las soluciones segun su orden
  # de dominacion
  fronts = self.non_dominated_sort(fns)
  indices = []
  k = int(self._selection * len(fns))

  for front in fronts:
    if len(indices) + len(front) <= k:
      indices.extend(front)
    else:
      # Cuando solo se utiliza una porcion del frente
      # se aplica crowding distance
      indices.extend(
          sorted(
              front,
              key=lambda i: -self.crowding_distance(fns, front, i)
          )[: k - len(indices)]
      )
      break
  return indices
```

Non Domianted Sort

El dominated sorting se realiza caminado por todo par de soluciones x, y obtenidas. Y se calcula el indice de dominacion de cada uno. Despues se inicializa un frente 0 de soluciones no dominadas. Todas las soluciones que estas soluciones dominan se le resta 1. Luego se pasa buscando todas las soluciones de rango 0 (hay nuevas xq se resto uno). Se le resta 1 al rango de todas las que estas domina y asi sucesivamente. Hasta que no queden soluciones

```
def non_dominated_sort(self, scores: List[List[float]]):
  # fronts almacena los frentes
  #(i.e. fronts[i] es el frente de rango i)
  fronts: List[List[int]] = [[]]

  # domination_rank en i indica la cantidad de soluciones
  # que dominan a la solucion i
  domination_rank = [0] * len(scores)

  # dominated_scores en i alamacena las soluciones dominadas
  # por la solucion i
```

```
  dominated_scores = [list() for _ in scores]

  # revisa todo par de soluciones y se establece
  # quien dominan a quien
  for i, score_i in enumerate(scores):
    for j, score_j in enumerate(scores):
      if self._improves(score_i, score_j):
          dominated_scores[i].append(j)
      elif self._improves(score_j, score_i):
          domination_rank[i] += 1
    if domination_rank[i] == 0:
       fronts[0].append(i)

  # de acuerdo a la informacion sobre quienes
  # se dominan, forma todos los frentes
  front_rank = 0
  while len(fronts[front_rank]) > 0:
    next_front = []
    for i in fronts[front_rank]:
      for dominated in dominated_scores[i]:
        domination_rank[dominated] -= 1
        if domination_rank[dominated] == 0:
          next_front.append(dominated)
    front_rank += 1
    fronts.append(next_front)

  return fronts[:-1]
```

Crowding Distance

Siguiendo el pseudocódigo escrito en (2.2) se hacen m iteraciones donde m es la cantidad de métricas. En cada iteracion se ordena el frente según el valor que tiene respecto a las m. Se establecen valores infinitos para los minimos y los maximos. Luego por cada punto adyacente se calculan las métricas. En este paso el valor de las métricas se normaliza de 0 a 1 utilizando Feature Scaling

```
def crowding_distance(
    self, scores: List[List[float]], front: List[int], index: int
) -> float:
  # Crowding distance usa los vectores normalizados.
  # Se aplica feature scaling para llevar los vectores a [0, 1]
  scaled_scores = feature_scaling(scores)

  crowding_distances: List[float] = [0 for _ in scores]
  for m in range(len(self._maximize)):
    # Se ordena de acuerdo a la metrica m
    front = sorted(front, key=lambda x: scores[x][m])
```

```
12
13      # Se establecen los extremos como infinitos
14      crowding_distances[front[0]] = math.inf
15      crowding_distances[front[-1]] = math.inf
16
17      # Valores de todas las soluciones con respecto a m
18      m_values = [scaled_scores[i][m] for i in front]
19      scale: float = max(m_values) - min(m_values)
20      if scale == 0:
21        scale = 1
22      for i in range(1, len(front) - 1):
23        crowding_distances[i] += (
24          scaled_scores[front[i + 1]][m] - scaled_scores[front[i -
    1]][m]
25        ) / scale
26
27    return crowding_distances[index]
```

Finalmente despues de este proceso quedan los k mejores y se deja el funcionaminto actual de PGE que siga con lo suyo de resamplear para los k mejores.

A parte de la adicion de NSPGE que conforma el grueso de la implementación se modifican las clases AutoML para que tenga en cuenta que los algoritmos de búsqueda pueden retornar uno o más valores y SearchAlgoritm (clase base de PGE) para que guarde las mejores a traves de la iteracionesen vez de una sola que determine como la más apta

3.2. Marco Experimental

En este sección se analiza el desemepeño de la propuesta utilizando AutoGOAL.
El objetivo es ejecutar datasets distintos con diferentes métricas de evaluación.

3.2.1. Métricas Utilizadas

Se utilizan las métricas F-score, Precision y Recall de sklearn.
Se utiliza la métrica accuracy
Se utiliza una métrica que calcula el tiempo estimado con que se entrena el algoritmo Es estimado xq existe un overhead pero es overhead es igual para todos los pipelines.

3.2.2. Corpus de Evaluación

Se utilizan dos corpus de datos para comprobar el comportamiento del sistema cuando optmiza para varias métricas simultáneamente.

CARS

Cars representa una conjunto de carros y con ciertas características catálogadas cualitativamente (ver 3.1). Cada carro se clasifica de acuerdo a sus atributos en inaceptable, aceptable, bueno o muy bueno (3.2). El dataset no tiene valores desconocidos.

Atributos	Valores
buying	v-high, hihg, med, low
maintance	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 3.1: 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 3.2: Distribución de clases en Cars

HAHA

Humor Analysis based on Human Annotation (HAHA), un conjunto de datos que contiene *tweets* en español y se clasifican en si son humorísticos o no. Contiene un total de 30000 tweets donde se utilizan 24000 para entrenaiento 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 3.3: Distribución de clases en HAHA

3.2.3. Configuración Experimental

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

3.3. Resultados y Análisis

3.3.1. Cars

Se utilzo una poblacion de 40 y una hora de autogoal y seleccion 10.

F-Score contra Tiempo de Entrenamiento

Como es un problema multi clase se utiliza f-score con weighted values que se ajusta a las clases que tienen poco cosa (o algo asi)

Este set esta facilon para un clasificador

Despues de ver la forma de los puntos se intuye que le frente de pareto tiene forma rectangular, lo que significa que las métricas no entran en conflicto entre si y es posible optimzar ambas muy bien

No obstante, si se observa el punto en la esquina inferior izquierda hay un punto que tiene muy buen f-score, no obstante se demora mucho en entrenar 5 veces mas en entrenar que otros similares. Utilizar multiobjetvio aqui evita que tu algoritmo se vaya por pileines por los cuales es mas lento la entrenacion

Precision contra Recobrado

En precision contra recobrado AutoGOAL solo encuentra puntos buenos, y aunque no nos da una buena representacion del frente de Pareto, tampoco es necesaria xq solo hay soluciones buenas.

En este caso hay varias soluciones que logran la misma eficiencia (o muy cercanas entre ella) Como la implementacion de Moo las devuelve toda puede ser de interes al investigador ver la diferencia entre algoritmos o hiper parametros de estos.

3.3.2. HAHA

Se utilzo una poblacion de 40 y 8 horas de autogoal y seleccion 10. Debido a la mayor complejidad de este problema y que AutoGOAL se demora mas en encontrar pipelines validos por generacion. Con este tiempo se puede obtener una representacion del frente de Pareto mas nitida.

F-Score contra Tiempo de Entrenamiento

Aqui se parce al ejemplo con Cars, aunque no de una manera tan marcada, es mas suave el punto. Y ocurren cosas similares que las mismas metricas utilizads en Cars

Precisión contra Recobrado

Aqui es diferente al trabajo con Cars las métricas forman un frente de pareto mas parecido a lo que describe la literatura. Se ve que hay un trade off evidente entre maximizar precison o recobrado. Esta exploarcion del espacio.

Conclusiones

Conclusiones

Recomendaciones

Recomendaciones

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. 13).

Back, T. (1996). *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press. (Vid. pág. 13).

Barocas, S., Hardt, M. & Narayanan, A. (2017). Fairness in machine learning. *Nips tutorial*, *1*, 2 (vid. pág. 14).

Bergstra, J., Bardenet, R., Bengio, Y. & Kégl, B. (2011). Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, *24* (vid. pág. 6).

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. 6).

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. 8).

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. 9).

Chen, B., Wu, H., Mo, W., Chattopadhyay, I. & Lipson, H. (2018). Autostacker: A compositional evolutionary learning system. *Proceedings of the genetic and evolutionary computation conference*, 402-409 (vid. pág. 5).

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. 7).

Custódio, A. L., Madeira, J. A., Vaz, A. I. F. & Vicente, L. N. (2011). Direct multisearch for multiobjective optimization. *SIAM Journal on Optimization*, *21*(3), 1109-1140 (vid. pág. 13).

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. pág. 14).

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. 7, 13, 15, 18).

Dellnitz, M., Schütze, O. & Hestermeyer, T. (2005). Covering Pareto sets by multilevel subdivision techniques. *Journal of optimization theory and applications*, *124*(1), 113-136 (vid. pág. 13).

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. 7).

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. 12).

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. 13).

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. 10).

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. 8, 17).

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. 6).

Fleming, P. J., Purshouse, R. C. & Lygoe, R. J. (2005). Many-Objective Optimization: An Engineering Design Perspective. En C. A. Coello Coello, A. Hernán-

dez Aguirre & E. Zitzler (Eds.), *Evolutionary Multi-Criterion Optimization* (pp. 14-32). Springer Berlin Heidelberg. (Vid. pág. 14).

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. 10).

Hillermeier, C. y col. (2001). *Nonlinear multiobjective optimization: a generalized homotopy approach* (Vol. 135). Springer Science & Business Media. (Vid. pág. 13).

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. 6).

Hutter, F., Kotthoff, L. & Vanschoren, J. (2019). *Automated machine learning: methods, systems, challenges*. Springer Nature. (Vid. pág. 5).

Jin, H., Song, Q. & Hu, X. (2018). Efficient neural architecture search with network morphism. *arXiv preprint arXiv:1806.10282*, *9* (vid. pág. 6).

Kaelbling, L. P., Littman, M. L. & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of artificial intelligence research*, *4*, 237-285 (vid. pág. 9).

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. 15).

Komer, B., Bergstra, J. & Eliasmith, C. (2014). Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn. *ICML workshop on AutoML*, *9*, 50 (vid. pág. 6).

Kuhn, H. W. & Tucker, A. W. (2014). Nonlinear programming. *Traces and emergence of nonlinear programming* (pp. 247-258). Springer. (Vid. pág. 13).

Larrañaga, P. & Lozano, J. A. (2001). *Estimation of distribution algorithms: A new tool for evolutionary computation* (Vol. 2). Springer Science & Business Media. (Vid. pág. 17).

LeDell, E. & Poirier, S. (2020). H2o automl: Scalable automatic machine learning. *Proceedings of the AutoML Workshop at ICML*, *2020* (vid. pág. 8).

Mégane, J., Lourenço, N. & Machado, P. (2021). Probabilistic grammatical evolution. *European Conference on Genetic Programming (Part of EvoStar)*, 198-213 (vid. págs. 8, 17).

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. 1).

Miettinen, K. (2012). *Nonlinear multiobjective optimization* (Vol. 12). Springer Science & Business Media. (Vid. pág. 12).

Mitchell, T., Buchanan, B., DeJong, G., Dietterich, T., Rosenbloom, P. & Waibel, A. (1990). Machine learning. *Annual review of computer science*, *4*(1), 417-433 (vid. pág. 1).

Mohr, F., Wever, M. & Hüllermeier, E. (2018). ML-Plan: Automated machine learning via hierarchical planning. *Machine Learning*, *107*(8), 1495-1515 (vid. pág. 10).

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. 7, 15)

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. 12).

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. 6, 7).

Pfisterer, F., Coors, S., Thomas, J. & Bischl, B. (2019). Multi-objective automatic machine learning with autoxgboostmc. *arXiv preprint arXiv:1908.10796* (vid. pág. 15).

Pllana, 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. 10).

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 Informatik. https://doi.org/10.4230/DagSemProc.04461.16. (Vid. pág. 13)
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. 14).

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. 10).

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. 10).

Thomas, J., Coors, S. & Bischl, B. (2018). Automatic gradient boosting. *arXiv preprint arXiv:1807.03873* (vid. págs. 7, 15).

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. 5, 6).

Tovbin, M. (s.f.). Meet TransmogrifAI, open source AutoML that powers einstein predictions. SF Big Analytics Meetup. (Vid. pág. 8).

Wei, T., Wang, C., Rui, Y. & Chen, C. W. (2016). Network morphism. *International conference on machine learning*, 564-572 (vid. pág. 8).

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, *8*(3), 229-256 (vid. pág. 9).

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. 14).

Žilinskas, A. (2013). On the worst-case optimal multi-objective global optimization. *Optimization Letters*, *7*(8), 1921-1928 (vid. pág. 13).

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. 13).

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. 9).