Universidad de La Habana

Facultad de Matemática y Computación

Asignatura: Aprendizaje Automático

Meta-Learning

Integrantes

Jabel Resendiz Aguirre
Amalia Beatriz Valiente Hinojosa
Noel Pérez Calvo
Melanie Forsythe Matos
Jorge Alejandro Echevarría Brunet
Arianne Camila Palancar Ochando

Carrera: Ciencia de la Computación

9 de enero de 2026

Abstract

Este trabajo aborda el problema de AutoML, proponiendo un enfoque basado en meta-learning que aprende meta-features de datasets (MetaFeatX) y utiliza transfer-learning para predecir configuraciones óptimas de hiperparámetros. Se presenta la motivación del proyecto, los problemas que resuelve, y se muestran resultados experimentales en comparación con sistemas AutoML tradicionales. La combinación de MetaFeatX y transfer-learning permite un AutoML más eficiente, interpretable y con menor costo computacional.

Índice

1. Introducción

El **meta-learning**, o "aprendizaje a aprender", es un área del aprendizaje automático que busca desarrollar algoritmos capaces de **aprender de experiencias pasadas para mejorar el rendimiento en nuevas tareas**. A diferencia del aprendizaje tradicional, donde un modelo se entrena para una tarea específica, el meta-learning se centra en **extraer conocimiento generalizable** que pueda transferirse a problemas desconocidos, acelerando el proceso de aprendizaje y mejorando la eficiencia.

En el contexto de **AutoML** (Automated Machine Learning), el meta-learning permite **seleccionar automáticamente algoritmos y configuraciones de hiperparámetros** adecuadas para un conjunto de datos dado, basándose en información obtenida de datasets anteriores. Esto reduce significativamente el tiempo de experimentación y evita la necesidad de un ajuste manual exhaustivo de los modelos.

Conceptos Clave: Para comprender meta-learning y el enfoque que se presenta en este reporte, es importante familiarizarse con los siguientes conceptos:

- **Meta-features:** Son características que describen datasets. Por ejemplo, el número de instancias, el número de atributos, la distribución de las clases o medidas estadísticas de los atributos. Las meta-features permiten comparar datasets y predecir qué algoritmos o configuraciones funcionarán mejor.
- **Pipeline de Machine Learning:** Es la secuencia de pasos que se aplican a un dataset, desde la preparación de datos (preprocesamiento, normalización) hasta el entrenamiento y evaluación de un modelo. Cada etapa puede tener múltiples opciones e **hiperparámetros** que afectan el rendimiento final.
- **Hiperparámetros:** Son parámetros de un algoritmo de ML que no se aprenden directamente a partir de los datos, sino que deben fijarse antes del entrenamiento. Ejemplos incluyen la profundidad máxima de un árbol de decisión, la tasa de aprendizaje de un modelo de redes neuronales o el número de vecinos en un k-NN. La correcta elección de hiperparámetros es crucial para el desempeño de un modelo.
- **Tareas de aprendizaje:** Se refiere a un problema específico de aprendizaje automático que se desea resolver, definido por un dataset y un objetivo de predicción (por ejemplo, clasificación o regresión). En meta-learning, cada tarea puede considerarse como una instancia en la que el sistema debe seleccionar un algoritmo y sus hiperparámetros adecuados. El aprendizaje meta busca **transferir conocimiento entre tareas** para mejorar la eficiencia en tareas nuevas.
- **AutoML:** Se refiere a la automatización del proceso de selección de algoritmos y ajuste de hiperparámetros. Los sistemas AutoML tradicionales requieren probar múltiples configuraciones y evaluar su desempeño, lo que puede ser costoso en tiempo y recursos.

Motivación y Problema El principal desafío que aborda este proyecto es: *cómo lograr que un sistema AutoML prediga de manera eficiente qué algoritmos y configuraciones funcionarán bien en un nuevo dataset*, sin tener que evaluar exhaustivamente todas las posibilidades. Los problemas principales incluyen:

- **Cold-start:** Para un dataset nuevo, no se conoce previamente qué configuraciones funcionarán, lo que obliga a probar muchas combinaciones costosas.
- **Diseño de meta-features:** No siempre las meta-features manuales garantizan buena generalización entre datasets.
- **Espacio de datasets complejo:** La diversidad de datasets y algoritmos hace difícil estimar qué configuraciones funcionarán bien.

Aunque existen sistemas como AutoSkLearn, PMF u OBOE que implementan estrategias de selección y optimización, estos enfoques requieren lanzar experimentos para cada nuevo dataset (fase cold-start), lo que limita la eficiencia. Además, el diseño manual de meta-features no siempre garantiza una buena generalización entre datasets.

Por esta razón, surge la necesidad de **aprender meta-features que capturen la relación entre datasets y configuraciones óptimas de hiperparámetros**. Estas meta-features permiten:

- Establecer una topología confiable del espacio de datasets, donde datasets cercanos comparten configuraciones de hiperparámetros similares.
- Reducir costos computacionales al usar configuraciones de datasets "vecinos" en nuevos datasets.
- Obtener información interpretable sobre cuándo un algoritmo funciona bien, proporcionando intuición sobre la naturaleza del problema.

Modelo 1: MetaFeatX MetaFeatX es un sistema de meta-learning que aprende automáticamente meta-features representativas de los datasets para guiar la selección de algoritmos y configuraciones de hiperparámetros. La idea central es construir un **embedding** de los datasets que capture la relación entre sus características y los hiperparámetros que producen los mejores resultados.

Su funcionamiento se basa en:

1. Aprender nuevas meta-features mediante un procedimiento de **Transporte Óptimo**, alineando las meta-features manuales con la distribución de configuraciones óptimas.
2. Estimar la **topología** del espacio de datasets y la **dimensionalidad intrínseca** del espacio de problemas para cada algoritmo o pipeline.
3. Identificar los datasets más **similares** a la tarea actual, generando un ranking de algoritmos prometedores.

Modelo 2: Transfer-Learning para Hiperparámetros Este segundo modelo recibe el ranking de algoritmos de MetaFeatX y predice los **valores de hiperparámetros** más prometedores para cada algoritmo, basándose en los datasets vecinos en el embedding aprendido. Esto permite una inicialización eficiente de AutoML y evita el costo computacional de explorar todas las configuraciones posibles desde cero.

Conexión con AutoML La combinación de ambos modelos permite realizar tareas típicas de AutoML de manera más eficiente:

- Selección de algoritmo: basada en la similitud de datasets en el embedding de MetaFeatX.
- Optimización de hiperparámetros: predicha por el modelo de transfer-learning usando vecinos.

En resumen, este enfoque de meta-learning permite reducir la fase de *cold-start*, mejorar la eficiencia de AutoML, y proporcionar interpretabilidad sobre cuándo y por qué un algoritmo funciona bien para un dataset específico.

2. MetaFeatX

Obtener el máximo rendimiento de un portafolio de algoritmos para una instancia de problema específica se reconoce como un cuello de botella importante en dominios que van desde la Programación por Restricciones y Satisfacibilidad hasta el Aprendizaje Automático. Los enfoques iniciales investigaron el uso de modelos de rendimiento generales [Rice, 1976], que estiman a priori el desempeño de cualquier algoritmo sobre cualquier instancia de problema, donde cada instancia se describe mediante un vector de *meta-features*, y el modelo de rendimiento se aprende en este espacio de meta-features.

En el contexto del Aprendizaje Automático supervisado, muchas meta-features han sido diseñadas manualmente para describir datasets. Tras varios desafíos internacionales de AutoML, destinados a automatizar la selección y ajuste de pipelines de ML [Hutter et al., 2019, Guyon et al., 2019], se ha observado que un modelo de rendimiento general y preciso difícilmente puede basarse únicamente en estas meta-features [Misir and Sebag, 2017]. Por ejemplo, el AutoSkLearn [Feurer et al., 2015a] se basa en optimización bayesiana y aprende iterativamente un modelo de rendimiento específico para cada dataset; PMF [Fusi et al., 2018] utiliza un enfoque probabilístico de filtrado colaborativo, y OBOE [Yang et al., 2019] combina filtrado colaborativo con aprendizaje activo.

El enfoque **MetaFeatX** se inspira en el trabajo de Rakotoarison et al. [Rakotoarison et al., 2022], que propone aprender meta-features capaces de capturar la topología del espacio de datasets en relación con el desempeño de un algoritmo de ML. MetaFeatX considera dos representaciones de los datasets: la básica, compuesta por 135 meta-features diseñadas manualmente, y la objetivo, que representa un dataset mediante la distribución de configuraciones de hiperparámetros de A que producen los mejores rendimientos. Para alinear ambas representaciones se utiliza Transporte Óptimo, de modo que la distancia euclidiana entre las meta-features aprendidas imite la distancia Wasserstein-Gromov sobre la representación objetivo [Cuturi, 2013, Peyré and Cuturi, 2019, Mémoli, 2011]. Este procedimiento permite derivar meta-features que pueden ser computadas desde cero para nuevos datasets, sin necesidad de un arranque en frío como en Yang et al. [2019], Fusi et al. [2018].

Entre los aspectos más relevantes de MetaFeatX se destacan:

1. Las meta-features definen una topología eficiente del espacio de datasets, útil para identificar regiones prometedoras de hiperparámetros.

2. Pueden ser empleadas como espacio de representación para inicializar otros métodos de AutoML o transfer-learning.

3. Permiten estimar la dimensionalidad intrínseca del espacio de datasets respecto a un algoritmo, lo que proporciona información sobre la complejidad de la tarea. Por ejemplo, se puede comparar la dimensión intrínseca de OpenML CC-18 respecto a AutoSkLearn, SVM , o Random Forest .

Esta sección se centra en presentar los aspectos más importantes del trabajo de Rakotoarison et al. (2022) y su relevancia para la construcción de meta-features en tareas de AutoML.

2.1. AutoML y Meta-Features

Las meta-features son estadísticas que describen datasets supervisados y se han diseñado manualmente considerando información descriptiva, teoría de la información, es-

tructura geométrica y *landmarking* (por ejemplo, desempeño de clasificadores simples). En dominios relacionados, como Satisfacibilidad (SAT) o Programación por Restricciones (CP), también existen meta-features manuales.

En AutoML, estas meta-features se utilizan principalmente para inicializar la búsqueda de optimización [Feurer et al., 2015a], pero no siempre garantizan un modelo de rendimiento preciso [Misir and Sebag, 2017]. Por ello, se han explorado enfoques de meta-features aprendidas, usando modelos de rendimiento complejos [Hazan et al., 2018] o redes neuronales que representan cada dataset como función de su distribución. Sin embargo, estas redes requieren grandes cantidades de datos, mientras que los benchmarks de AutoML incluyen relativamente pocos datasets. Esto motiva métodos como MetaFeatX, que construyen representaciones efectivas basándose en meta-features existentes.

2.2. Transporte Óptimo

MetaFeatX se apoya en el Transporte Óptimo (OT) para medir la similitud entre datasets en dos representaciones: la básica (meta-features existentes) y la objetivo (rendimiento óptimo de configuraciones de hiperparámetros).

Sea (Ω_x, d_x) y (Ω_y, d_y) espacios métricos compactos con distribuciones x y y. El conjunto de distribuciones con márgenes x y y se denota $\Gamma(x, y)$, y $c : \Omega_x \times \Omega_y \to \mathbb{R}^+$ es la función de costo de transporte.

El problema de OT busca una distribución $\gamma \in \Gamma(x, y)$ que minimice el costo esperado [Peyré and Cuturi, 2019]:

$$d_W^q(x, y) = \left(\min_{\gamma \in \Gamma(x,y)} \mathbb{E}_{(x,y)\sim\gamma}[c^q(x, y)] \right)^{1/q}, \tag{1}$$

definiendo la distancia de Wasserstein de orden q.

La distancia **Gromov-Wasserstein (GW)** mide qué tan bien se preservan las relaciones internas de cada dominio [Mémoli, 2011]:

$$d_{GW}^q(x, y) = \left(\min_{\gamma \in \Gamma(x,y)} \mathbb{E}_{(x,y),(x',y')\sim\gamma} \left| d_x(x, x') - d_y(y, y') \right|^q \right)^{1/q}. \tag{2}$$

La **Fused Gromov-Wasserstein (FGW)** combina ambas [Titouan et al., 2019]:

$$\begin{aligned} d_{FGW;\alpha}^q(x, y) = \min_{\gamma \in \Gamma(x,y)} \; & (1 - \alpha) \underbrace{\int c^q(x, y)\, d\gamma(x, y)}_{\text{Wasserstein Loss}} \\ & + \alpha \underbrace{\int\int |d_x(x, x') - d_y(y, y')|^q \, d\gamma(x, y) d\gamma(x', y')}_{\text{Gromov-Wasserstein Loss}}, \end{aligned} \tag{3}$$

donde $\alpha \in [0, 1]$ controla el balance: $\alpha = 0$ corresponde a Wasserstein, $\alpha = 1$ a Gromov-Wasserstein.

Estas distancias permiten evaluar la similitud entre datasets considerando tanto las meta-features como la estructura de rendimiento de los algoritmos, y constituyen la base de MetaFeatX para construir representaciones que conectan la información básica con la objetivo, facilitando la transferencia de conocimiento entre datasets. Esta metodología se ha usado previamente en adaptación de dominio y transfer learning [Courty et al., 2017, Alvarez-Melis and Fusi, 2020] y en la consistencia de espacios latentes de autoencoders [Xu et al., 2020, Nguyen et al., 2024], y constituye la base de MetaFeatX para vincular la información básica con la objetivo.

2.3. Principio de MetaFeatX y Argumentación del Benchmark

MetaFeatX busca aprender nuevas meta-features para algoritmos de ML a partir de meta-features básicas y representaciones objetivo, siguiendo un enfoque basado en Transporte Óptimo.

Cada dataset se puede representar de dos formas:

1. **Representación básica:** Un vector de las D meta-features manualmente diseñadas, fácil de calcular para cualquier dataset.

2. **Representación objetivo:** La distribución de configuraciones de hiperparámetros que producen los mejores resultados para un dataset, disponible solo para un subconjunto de datasets del benchmark.

La idea de MetaFeatX es construir un *puente* entre estas dos representaciones. Para ello:

- Se proyecta la representación objetivo en un espacio de dimensión más baja d mediante un método que preserve distancias, como Multi-Dimensional Scaling (MDS). Esto produce vectores $u_i \in \mathbb{R}^d$ para cada dataset [Cox and Cox, 2001]

- Se aprende un mapeo $\psi : \mathbb{R}^D \to \mathbb{R}^d$ que transforma la representación básica al espacio proyectado, de manera que la distancia euclidiana entre $\psi(x_i)$ refleje la topología de los u_i, usando la distancia **Fused Gromov-Wasserstein**.

El resultado de este mapeo son las **meta-features MetaFeatX**, que:

- Se pueden calcular de manera económica a partir de las meta-features básicas.

- Definen una distancia euclidiana que refleja la proximidad de datasets en términos de desempeño de hiperparámetros, facilitando tareas de AutoML como inicialización de optimización o transferencia de conocimiento.

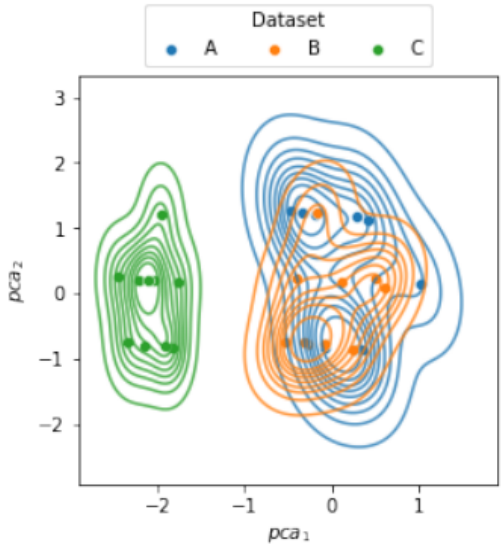

Figura 1: Configuraciones principales de los datasets A, B y C, donde B (en naranja) (respecto a C, en verde) es el vecino más cercano de A con respecto a la representación objetivo.

Argumentación del benchmark.
Como los benchmarks de AutoML (por ejemplo, OpenML CC-18) contienen relativamente pocos datasets etiquetados con representaciones objetivo, existe riesgo de sobreajuste al aprender las meta-features. MetaFeatX aborda este problema mediante un procedimiento de *bootstrap* [Efron, 1979], que genera datasets adicionales a partir de los existentes para densificar el espacio de meta-features. Este procedimiento puede incluir la adición de ruido siguiendo una distribución gaussiana, lo que permite explorar más exhaustivamente el espacio de meta-features y mejorar la generalización del modelo.

2.4. Algoritmo MetaFeatX

El algoritmo MetaFeatX se entrena sobre $p = 1000 \times n$ datasets de entrenamiento del benchmark, previamente aumentados mediante *bootstrap* (ver sección anterior y Apéndice B). Las meta-features de MetaFeatX se construyen en un "procedimiento de tres pasos" ilustrado en la Figura 2.

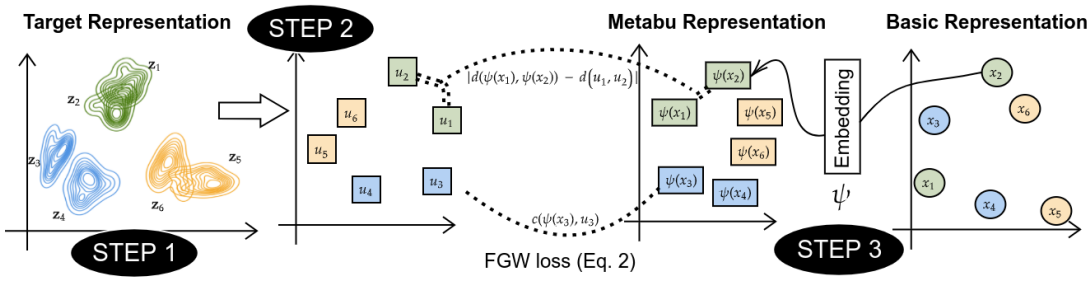

Figura 2: De las meta-features básicas a las MetaFeatX usando Fused Gromov-Wasserstein. Representaciones básicas (círculos), MetaFeatX (cuadrados) y representaciones objetivo (subgráfico izquierdo). Datasets vecinos en el espacio objetivo mantienen el mismo color en todos los subgráficos.

Paso 1: Representación objetivo y distancia de Wasserstein. Para cada dataset de entrenamiento i, se define $\Theta_i \subset \mathcal{T}$ como el conjunto de configuraciones de hiperparámetros cuyo desempeño se encuentra en el top-L de configuraciones conocidas ($L = 20$ en los experimentos). La *representación objetivo* z_i se define como la distribución discreta con soporte Θ_i. La distancia entre datasets se mide mediante la "distancia 1-Wasserstein":

$$d_W^1(z_i, z_j) = \min_{\gamma \in \Gamma(z_i, z_j)} \mathbb{E}_{(x,y)\sim\gamma}[c(x, y)], \tag{4}$$

donde c es el costo de transporte Euclidiano en el espacio de configuraciones.

Paso 2: Proyección de la representación objetivo en $\mathbb{R}^d$. La representación z_i se proyecta en $\mathbb{R}^d$, donde d se estima usando una medida de dimensionalidad intrínseca (ver más abajo). Se utiliza "Multi-Dimensional Scaling (MDS)" para que la distancia euclidiana entre las proyecciones u_i y u_j aproxime la distancia 1-Wasserstein:

$$d(u_i, u_j) \approx d_W^1(z_i, z_j). \tag{5}$$

Estas proyecciones u_i se definen hasta una isometría, pero preservan la topología relativa de los datasets en el espacio objetivo.

Paso 3: Aprendizaje de las MetaFeatX. Se define la distribución discreta uniforme sobre las representaciones básicas:

$$x = \frac{1}{p} \sum_{i=1}^{p} \delta_{x_i}, \tag{6}$$

y sobre las proyecciones objetivo:

$$u = \frac{1}{n} \sum_{i=1}^{n} \delta_{u_i}. \tag{7}$$

Las MetaFeatX se construyen encontrando un mapeo $\psi : \mathbb{R}^D \to \mathbb{R}^d$ que minimice la distancia "Fused Gromov-Wasserstein" entre la distribución push-forward $\psi_{\#}x$ y u:

$$\psi^* = \arg\min_{\psi \in \Psi} d^q_{FGW;\alpha}(\psi_{\#}x, u) + \lambda \|\psi\|, \tag{8}$$

donde λ es un parámetro de regularización y $\|\psi\|$ es la norma de la función ψ. Se considera ψ lineal para evitar sobreajuste y facilitar la interpretación respecto a las meta-features básicas.

La optimización se realiza mediante un esquema de "bilevel optimization" [**?**]:

- **Problema interno:** minimizar $d_{FGW;\alpha}(\psi_{\#}x, u)$ usando un método de gradiente proximal [Xu et al., 2019], refinando la matriz de transporte γ con el "algoritmo de Sinkhorn" [Cuturi, 2013].
- **Problema externo:** optimizar ψ considerando γ como constante, usando el optimizador "ADAM" [Kingma and Ba, 2015] con tasa de aprendizaje 0.01, $\alpha = 0{,}5$ y $\lambda = 0{,}001$.

Dimensión intrínseca del espacio de datasets. El parámetro principal de MetaFeatX es d, el número de meta-features necesarias para aproximar la representación objetivo. Se estima usando un método basado en vecinos más cercanos : para cada muestra x, se calculan sus primeras distancias a los vecinos $x^{(1)}$ y $x^{(2)}$, y se define

$$\mu(x) = \frac{d(x, x^{(2)})}{d(x, x^{(1)}) + \epsilon}. \tag{9}$$

Ordenando las muestras por $\mu(x_i)$, d se obtiene como la pendiente de la aproximación lineal de la curva

$$\{(\log \mu(x_i), -\log(1 - \frac{i}{(m+1)})), 1 \leq i \leq m\}, \tag{10}$$

proporcionando una estimación garantizada de la dimensionalidad intrínseca del espacio donde habitan los datasets.

Este procedimiento asegura que las MetaFeatX reflejen la topología del espacio objetivo y puedan ser computadas de manera económica a partir de las meta-features básicas, facilitando tareas de AutoML como inicialización de optimización y transferencia de conocimiento.

3. Transfer-Learning de Hiperparámetros

4. Experimentación

La experimentación en este trabajo se organiza en torno a tres objetivos principales: Validar si las meta-características aprendidas por METABU capturan adecuadamente la topología "verdadera" inducida por las configuraciones de hiperparámetros; evaluar su utilidad práctica para AutoML, tanto sin como con modelo de rendimiento; y analizar la sensibilidad del método y extraer información sobre la estructura del espacio de conjuntos de datos y los "nichos" de distintos algoritmos. La experimentación se fundamenta en la idea de que unas buenas meta-características deben inducir una métrica sobre el espacio de conjuntos de datos tal que la cercanía entre conjuntos de datos signifique también similitud en los conjuntos de configuraciones de hiperparámetros que proporcionan buen rendimiento.

Esta noción se formaliza mediante representaciones objetivo de cada conjunto de datos como distribuciones discretas sobre configuraciones de hiperparámetros de alto rendimiento, y distancias de Wasserstein y Gromov-Wasserstein entre dichas distribuciones. Desde este punto de vista, el propósito de la experimentación es triple: (1) comprobar si la distancia euclídea en el espacio de meta-características METABU aproxima bien la distancia de Wasserstein entre representaciones objetivo; (2) verificar si esta métrica permite diseñar estrategias de muestreo inicial de configuraciones más eficaces que los meta-features manuales clásicos (AutoSklearn, Landmark, SCOT) o el muestreo uniforme; y (3) estudiar, a partir de las meta-características aprendidas, la dimensión intrínseca del espacio de conjuntos de datos y la importancia relativa de distintas meta-características humanas para varios algoritmos.

4.1. Configuración experimental y tareas

El banco de pruebas es la suite OpenML CC-18 de clasificación tabular, con 72 conjuntos de datos, de los cuales 64 disponen de información suficiente para construir la representación objetivo a partir de configuraciones evaluadas. Para evitar sobreajuste dada la escasez de conjuntos de datos, los autores amplían la colección mediante *bootstrap*: para cada conjunto de datos original se generan 1000 versiones remuestreadas con reemplazo, calculando de nuevo las meta-características básicas pero heredando la misma representación objetivo, lo que densifica el espacio de meta-features sin distorsionar la estructura de rendimiento.

Sobre esta base, se definen tres tareas experimentales, todas validadas con un esquema *leave-one-out* sobre los 64 conjuntos de datos con representación objetivo. La primera tarea evalúa la capacidad de capturar la topología objetivo; la segunda estudia el comportamiento de AutoML sin modelo de rendimiento explícito, en un escenario de inicialización basada en vecindario; y la tercera analiza el uso de las meta-características como mecanismo de inicialización para sistemas AutoML con modelo de rendimiento, concretamente AutoSklearn y Probabilistic Matrix Factorization (PMF).

4.1.1. Tarea 1: Captura de la topología objetivo

En la primera tarea, el objetivo es cuantificar hasta qué punto el vecindario de un conjunto de datos en el espacio de meta-características METABU coincide con su vecin-

dario real en el espacio de distribuciones de configuraciones de alto rendimiento. Para cada conjunto de datos de prueba se calcula, por un lado, la lista ordenada de vecinos más cercanos según la distancia de Wasserstein entre sus representaciones objetivo, y, por otro, la lista de vecinos según la distancia euclídea en el espacio de meta-características METABU y en los espacios de referencia (AutoSklearn, Landmark, SCOT).

La similitud entre ambas listas se mide mediante la métrica NDCG@k (normalized discounted cumulative gain) sobre los primeros k vecinos, con valores de k entre 5 y 35, y se considera como indicador el NDCG@k medio sobre todos los conjuntos de datos de prueba. Teóricamente, un valor elevado de NDCG indica que la métrica inducida por las meta-características preserva la estructura de vecindad que es relevante para AutoML, es decir, la inducida por el comportamiento de las configuraciones de hiperparámetros sobre los distintos conjuntos de datos.

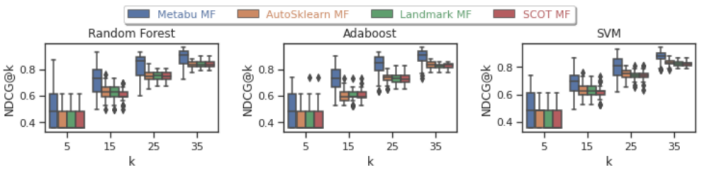

Figura 3:

4.1.2. Tarea 2: AutoML sin modelo de rendimiento

En la segunda tarea se evalúa la capacidad de METABU para guiar el muestreo inicial de configuraciones sin recurrir a un modelo de rendimiento explícito, en un escenario puramente basado en la información de vecindario. Para cada conjunto de datos de prueba y para cada conjunto de meta-características (METABU, AutoSklearn, Landmark, SCOT) se define una distribución sobre configuraciones, denotada $\hat{z}^{mf}$, calculada como mezcla ponderada de las distribuciones objetivo de los diez vecinos más cercanos en el espacio correspondiente, ponderando cada vecino con un peso decreciente $e^{-\ell}$ según su orden y normalizando para obtener una distribución de probabilidad.

A partir de $\hat{z}^{mf}$, se muestrean iterativamente configuraciones de hiperparámetros, se entrena el modelo con un presupuesto de 15 minutos de CPU y menos de 8 GB de memoria, y se registra el rendimiento de la mejor configuración encontrada hasta el instante t. Para cada t, se ordenan los distintos conjuntos de meta-características, junto con un muestreador uniforme de referencia, según el rendimiento obtenido, produciendo una curva de rango $r(t, mf)$ promediada sobre todos los conjuntos de datos, lo que permite evaluar la eficacia de cada métrica a la hora de explotar la información de experiencias previas sin aprendizaje adaptativo sobre el problema actual.

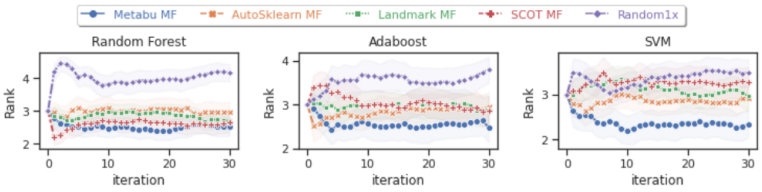

Figura 4:

4.1.3. Tarea 3: AutoML con modelo de rendimiento

La tercera tarea estudia el papel de las meta-características como mecanismo de inicialización de sistemas de AutoML basados en modelos de rendimiento, en particular AutoSklearn y PMF. En estos sistemas, la búsqueda de configuraciones se realiza de forma adaptativa: el método selecciona sucesivamente configuraciones, observa su rendimiento y actualiza un modelo probabilístico que orienta las decisiones posteriores de muestreo.

En este caso, las meta-características se utilizan para seleccionar un conjunto inicial de configuraciones prometedoras. En AutoSklearn, se toman las mejores configuraciones observadas en los diez vecinos más cercanos del conjunto de datos, según la métrica en el espacio de meta-características, y se evalúan en el conjunto de datos objetivo para alimentar el modelo de rendimiento inicial sobre el que se ejecuta la optimización bayesiana. En PMF, se realiza un procedimiento análogo con cinco vecinos, usando sus mejores configuraciones para rellenar parcialmente la fila correspondiente al conjunto de datos en la matriz de rendimientos y así inicializar su representación latente.

A partir de esta fase de inicialización, el sistema continúa su búsqueda estándar y el rendimiento se resume mediante curvas de rango en función del número de iteraciones, comparando versiones originales de AutoSklearn y PMF con sus variantes híbridas METABU+AutoSklearn y METABU+PMF, así como con muestreadores uniformes reforzados que seleccionan el mejor de varios candidatos aleatorios por iteración.

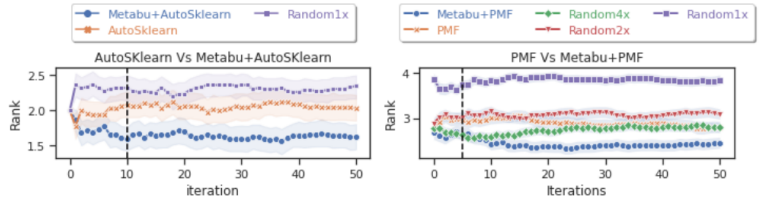

Figura 5:

4.2. Algoritmos, hiperparámetros y protocolo

Los experimentos se realizan para tres algoritmos de aprendizaje supervisado (AdaBoost, RandomForest y SVM, usando implementaciones de `scikit-learn`) y dos pipelines de AutoML (AutoSklearn y PMF). Para cada algoritmo se define un espacio de hiperparámetros de dimensión moderada, incluyendo por ejemplo el número de árboles, la profundidad máxima y los criterios de división en RandomForest, así como parámetros como C, tipo de *kernel*, grado y γ en el caso de SVM, y un conjunto amplio de opciones de preprocesamiento y clasificación en AutoSklearn.

Las representaciones objetivo de los conjuntos de datos se construyen a partir del conjunto de configuraciones con mejor rendimiento disponible: las veinte mejores sobre decenas de miles de configuraciones evaluadas en OpenML para AdaBoost, RandomForest y SVM, quinientas configuraciones por conjunto de datos para AutoSklearn y las veinte mejores entradas de la matriz de filtrado colaborativo en el caso de PMF. El entrenamiento y evaluación de cada configuración se realiza sobre las particiones de entrenamiento, validación y prueba provistas por OpenML, recurriendo a validación cruzada de cinco pliegues para estimar el rendimiento.

En todas las tareas, los indicadores se calculan mediante el esquema *leave-one-out*: para cada conjunto de datos que dispone de representación objetivo, se entrena METABU

sobre el resto de conjuntos de datos y sus versiones de bootstrap, y se evalúa sobre el conjunto excluido; adicionalmente, los ocho conjuntos de datos sin representación objetivo se incorporan como casos de prueba en las tareas de muestreo de configuraciones, donde solo es necesario observar el rendimiento y no la estructura de la representación objetivo.

4.3. Resultados y análisis

En la Tarea 1, las curvas de NDCG@k muestran que la métrica inducida por METABU se alinea mejor que las referencias con la topología objetivo, especialmente a medida que aumenta el valor de k. Aunque la varianza es mayor debido a la naturaleza entrenable de METABU, los resultados indican que supera de forma significativa a las meta-características de AutoSklearn, Landmark y SCOT para todos los valores de k y para todos los espacios de hiperparámetros considerados, lo que respalda la idea de que las meta-características aprendidas capturan de forma más fiel la estructura relevante para AutoML.

En la Tarea 2, las curvas de rango evidencian una mejora sustancial en la calidad del muestreo de configuraciones. Para RandomForest, las meta-características SCOT resultan competitivas en las primeras iteraciones, pero METABU pasa a dominar en fases posteriores; para AdaBoost, las meta-características de AutoSklearn pueden ofrecer ligeras ventajas en las tres primeras configuraciones evaluadas, mientras que METABU se vuelve significativamente mejor a partir de ese punto; para SVM, METABU supera de forma clara y consistente a todas las alternativas a lo largo de toda la búsqueda. Estos resultados indican que la topología inducida por METABU permite explotar con mayor eficacia los vecindarios de conjuntos de datos a la hora de seleccionar regiones prometedoras del espacio de configuraciones, incluso en ausencia de un modelo de rendimiento adaptativo.

En la Tarea 3, las variantes METABU+AutoSklearn y METABU+PMF mejoran los rangos obtenidos por las versiones originales de AutoSklearn y PMF, pese a que la única diferencia entre ellas reside en la fase de inicialización. En particular, METABU+AutoSklearn sobrepasa al pipeline AutoSklearn puro a partir de un número moderado de iteraciones, y METABU+PMF consigue superar al muestreo uniforme reforzado que elige el mejor de cuatro configuraciones aleatorias a partir de aproximadamente la décima iteración. Este comportamiento sugiere que las meta-características de METABU no solo favorecen la explotación, al identificar una buena región inicial del espacio de configuraciones, sino que también facilitan una exploración eficaz cuando se combinan con modelos de rendimiento que se actualizan de manera incremental.

4.4. Sensibilidad e interpretabilidad

El análisis de sensibilidad se centra en los parámetros α, que pondera las componentes Wasserstein y Gromov-Wasserstein en la distancia FGW, y λ, que controla el peso de la regularización L1 sobre la transformación lineal ψ. Se evalúa la diferencia NDCG@10(METABU) − NDCG@10(AutoSklearn) sobre una rejilla de valores de α y λ, observándose que dicha diferencia es positiva en todo el dominio considerado y estadísticamente significativa para niveles de significación convencionales. Además, METABU muestra baja sensibilidad cuando $\lambda \leq 10^{-3}$ y α se sitúa entre 0.3 y 0.7, mientras que valores extremos de α que anulan prácticamente una de las dos componentes de la distancia conducen a una degradación apreciable del rendimiento.

En cuanto a la dimensión intrínseca del espacio de conjuntos de datos, definida a partir de la distribución de distancias de Wasserstein de orden uno entre representaciones objetivo, se estima que es aproximadamente 6 para AutoSklearn, 8 para AdaBoost, 9 para RandomForest y 14 para SVM, con un análisis adicional de estabilidad que muestra cómo esta dimensión crece al incorporar más conjuntos de datos. Esta medida proporciona una interpretación cuantitativa de la complejidad del problema de AutoML desde la perspectiva de cada algoritmo: cuanto más variada es la respuesta de un algoritmo a los distintos conjuntos de datos, mayor es la dimensión intrínseca asociada.

Por último, se analiza la interpretabilidad de las meta-características METABU proyectando las representaciones aprendidas mediante análisis de componentes principales y midiendo la importancia de cada meta-característica manual como contribución a la primera componente principal. Representando cada meta-característica como un punto en el plano definido por sus importancias para dos algoritmos distintos, se obtiene una visión de qué propiedades de los datos son compartidas o específicas de cada algoritmo, observándose, por ejemplo, que el índice de Dunn y las medidas de importancia de atributos resultan relevantes tanto para RandomForest como para AdaBoost, mientras que la proporción de instancias con valores perdidos y el desbalanceo de clases afectan más a AdaBoost, y la dispersidad y la asimetría de los atributos tienen un peso mayor en SVM que en RandomForest.

5. Discusión

6. Conclusiones

Anexos

Material Suplementario

El material suplementario incluye detalles adicionales sobre:

- La ampliación del benchmark OpenML (Apéndice A)
- Pseudo-código del algoritmo (Apéndice B)
- Configuración experimental, indicadores de desempeño y procedimiento de validación (Apéndice C)
- Espacio de configuración de hiperparámetros (Apéndice D)
- Lista de meta-features básicas y conjuntos de meta-features de referencia (Apéndice E)
- Detalles sobre el tiempo computacional (Apéndice F)
- Información sobre el problema AutoML proporcionada por el enfoque: dimensionalidad intrínseca (Apéndice G.1) y visualización de los nichos de los algoritmos ML considerados (Apéndice G.2)
- Resultados detallados con desviación estándar en las tres tareas (Apéndice H)

- Comparación par a par de MetaFeatX con meta-features de referencia (Apéndice I)
- Análisis de sensibilidad de la dimensión d (Apéndice J)
- Curvas de desempeño de la Tarea 2 (Apéndice K)

Ampliación del benchmark OpenML

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Pseudo-código del algoritmo

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Configuración experimental y procedimiento de validación

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Espacios de Configuración de Hiperparámetros

Las configuraciones de hiperparámetros utilizadas para Adaboost, Random Forest y SVM, así como sus rangos, se detallan en la Tabla 1. Para AutoSkLearn, solo se incluyó la lista de hiperparámetros considerados; sus rangos están detallados en [Feurer et al., 2015b]. El espacio de hiperparámetros usado en PMF es el mismo que en AutoSkLearn. La implementación de MetaFeatX utiliza la librería `ConfigSpace` [Lindauer et al., 2019] para gestionar los hiperparámetros.

Clasificador	**Hiper-Parámetro (HP)**	**Rango**
Adaboost	imputation	mean, median, most frequent
	n_estimators	[50, 500]
	algorithm	SAMME, SAMME.R
	max_depth	[1, 10]
	learning_rate	[0.01 , 2.0]
Random Forest (RF)	imputation	mean, median, most frequent
	criterion	gini, entropy
	max_features	(0, 1]
	min_samples_split	[2, 20]
	min_samples_leaf	[1, 20]
	bootstrap	True, False
SVM	imputation	mean, median, most frequent
	C	[0.03125, 32768]
	kernel	rbf, poly, sigmoid
	degree	[1, 5]
	gamma	$[3.0517578125 \times 10^{-5}, 8]$
	coef0	[-1, 1]
	shrinking	True, False
	tol	$[10^{-5}, 10^{-1}]$
	max_iter	-1

Cuadro 1: Rangos de hiperparámetros para Adaboost, Random Forest y SVM.

Método	**Parámetros**
balancing	strategy
adaboost	learning_rate, max_depth, n_estimators
bernoulli_nb	fit_prior
decision_tree	max_depth_factor, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf
extra_trees	criterion, max_depth, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf
gradient_boosting	l2_regularization, learning_rate, loss, max_bins, max_depth, max_leaf_nodes, min_samples_leaf, scoring, tol, n_iter_no_change, validation_fraction
k_nearest_neighbors	p, weights
lda	tol, shrinkage_factor
liblinear_svc	dual, fit_intercept, intercept_scaling, loss, multi_class, penalty, tol
libsvm_svc	gamma, kernel, max_iter, shrinking, tol, coef0, degree
mlp	alpha, batch_size, beta_1, beta_2, early_stopping, epsilon, hidden_layer_depth, learning_rate_init, n_iter_no_change, num_nodes_per_layer, shuffle, solver, tol, validation_fraction
multinomial_nb	fit_prior
passive_aggressive	average, fit_intercept, loss, tol
qda	reg_param
random_forest	criterion, max_depth, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf
sgd	average, fit_intercept, learning_rate, loss, penalty, tol, epsilon, eta0, l1_ratio, power_t
extra_trees_preproc_for_classification	criterion, max_depth, max_features, max_leaf_nodes, min_impurity_decrease, min_samples_leaf, min_samples_split, min_weight_fraction_leaf, n_estimators
fast_ica	fun, whiten, n_components
feature_agglomeration	linkage, n_clusters, pooling_func
kernel_pca	n_components, coef0, degree, gamma
kitchen_sinks	n_components
liblinear_svc_preprocessor	dual, fit_intercept, intercept_scaling, loss, multi_class, penalty, tol
nystroem_sampler	n_components, coef0, degree, gamma
pca	whiten
polynomial	include_bias, interaction_only
random_trees_embedding	max_depth, max_leaf_nodes, min_samples_leaf, min_samples_split, min_weight_fraction_leaf, n_estimators
select_percentile_classification	score_func
select_rates_classification	score_func, mode

Cuadro 2: Lista de hiperparámetros considerados en la pipeline de AutoSkLearn.

Lista de meta-características básicas y conjuntos de meta-características de referencia

La lista de meta-features utilizadas en los experimentos se detalla en las Tablas 3 y 4. Las meta-features se extraen con PyMFE [Alcobaça et al., 2020], excepto las meta-features de AutoSkLearn, SCOT y Landmark, que se calculan a partir de la biblioteca AutoSkLearn.

Cuadro 3: Lista de meta-features utilizadas en los experimentos (1/2).

Meta-feature	**Descripción**
best_node	Rendimiento del mejor nodo individual de un árbol de decisión.
elite_nn	Rendimiento del clasificador Elite Nearest Neighbor.
linear_discr	Rendimiento del clasificador Linear Discriminant.
naive_bayes	Rendimiento del clasificador Naive Bayes.
one_nn	Rendimiento del clasificador 1-Nearest Neighbor.
random_node	Nodo aleatorio del árbol de decisión.
worst_node	Nodo menos informativo del árbol de decisión.
one_itemset	Meta-feature basada en un solo itemset.
two_itemset	Meta-feature basada en dos itemsets.
c1	Entropía de las proporciones de clase.
c2	Razón de desbalance de clases.
cls_coef	Coeficiente de clustering.
density	Densidad promedio del grafo.
f1	Máxima razón discriminante de Fisher.
f1v	Razón discriminante direccional de Fisher.
f2	Volumen de la región de solapamiento.
f3	Máxima eficiencia individual de característica.
f4	Eficiencia colectiva de características.
hubs	Hub score de la red.
l1	Suma de errores por programación lineal.
l2	Tasa de error OVO del clasificador lineal.
l3	No linealidad de clasificador lineal.
lsc	Cardinalidad promedio del conjunto local.
n1	Fracción de puntos borderline.
n2	Razón de distancias intra/extraclase NN.
n3	Tasa de error del k-NN.
n4	Fracción de hiperesferas que cubren los datos.
t1	Promedio de características por dimensión.
t2	Promedio de dimensiones PCA por punto.
t3	Ratio de dimensión PCA a dimensión original.
t4	Índice de Calinski y Harabasz.
ch	Índice de Calinski y Harabasz.
int	INT index.
nre	Entropía relativa normalizada.
pb	Pearson entre matching de clase e instancias.
sc	Número de clusters menores a un tamaño dado.
sil	Valor medio de silhouette.
vdb	Índice de Davies-Bouldin.
vdu	Índice de Dunn.
leaves	Número de nodos hoja en el DT.
leaves_branch	Tamaño de ramas del DT.
leaves_corrob	Corroboración de hojas del DT.
leaves_homo	Homogeneidad de hojas.
leaves_per_class	Proporción de hojas por clase.
nodes	Número de nodos no-hoja en el DT.
nodes_per_attr	Ratio nodos/atributos.
nodes_per_inst	Ratio nodos/no-hojas por instancia.
nodes_per_level	Ratio nodos por nivel.
nodes_repeated	Nodos repetidos.
tree_depth	Profundidad de árbol.
tree_imbalance	Desbalance de árbol.
tree_shape	Forma del árbol.
var_importance	Importancia de características del DT.

Continuación en la siguiente página

Meta-feature	Descripción
can_cor	Correlaciones canónicas.
cor	Correlación absoluta entre pares de columnas.
cov	Covarianza absoluta entre pares de atributos.
eigenvalues	Valores propios de la matriz de covarianza.
g_mean	Media geométrica.
gravity	Distancia entre clases minoritaria y mayoritaria.
h_mean	Media armónica.
iq_range	Rango intercuartílico (IQR).
kurtosis	Curtosis.
lh_trace	Lawley-Hotelling trace.
mad	MAD ajustada.
max	Máximo.
mean	Media.
median	Mediana.
min	Mínimo.
nr_cor_attr	Número de pares altamente correlacionados.
nr_disc	Número de atributos discretos.
nr_norm	Número de atributos normales.
nr_outliers	Número de outliers.
p_trace	Pillai's trace.
range	Rango (max-min).
roy_root	Raíz más grande de Roy.
sd	Desviación estándar de cada atributo.
sd_ratio	Prueba estadística de homogeneidad de covarianzas.
skewness	Sesgo de cada atributo.
sparsity	Medida de sparsity (posiblemente normalizada).
t_mean	Media recortada de cada atributo.
var	Varianza de cada atributo.
w_lambda	Valor de Wilks' Lambda.
attr_conc	Coeficiente de concentración entre pares de atributos.
attr_ent	Entropía de Shannon de cada atributo predictivo.
class_conc	Coeficiente de concentración entre atributo y clase.
class_ent	Entropía de Shannon del atributo objetivo.
eq_num_attr	Número de atributos equivalentes para la tarea.
joint_ent	Entropía conjunta entre cada atributo y la clase.
mut_inf	Información mutua entre cada atributo y la clase.
ns_ratio	Medida de ruido de atributos.
cohesiveness	Distancia ponderada mejorada que mide densidad de distribución.
conceptvar	Variación del concepto estimando la variabilidad de clases.
impconceptvar	Variación mejorada del concepto estimando variabilidad de clases.
wg_dist	Distancia ponderada de la distribución de ejemplos.
attr_to_inst	Ratio entre número de atributos y número de instancias.
cat_to_num	Ratio entre número de atributos categóricos y numéricos.
freq_class	Frecuencia relativa de cada clase.
inst_to_attr	Ratio entre número de instancias y atributos.
nr_attr	Número total de atributos.
nr_bin	Número de atributos binarios.
nr_cat	Número de atributos categóricos.
nr_class	Número de clases distintas.
nr_inst	Número de instancias (filas) del dataset.
nr_num	Número de atributos numéricos.
num_to_cat	Número de atributos numéricos y categóricos.
PCASkewnessFirstPC	Sesgo de ejemplos en el primer componente principal.
PCAKurtosisFirstPC	Curtosis de ejemplos en el primer componente principal.
PCAFracOfCompFor95Per	Fracción de componentes explicando 95 % de varianza.
Landmark1NN	Rendimiento del clasificador 1-NN.
LandmarkRandomNodeLearner	Rendimiento de Random Node Learner.
LandmarkDecisionNodeLearner	Rendimiento de Decision Node Learner.
LandmarkDecisionTree	Rendimiento de Decision Tree.
LandmarkNaiveBayes	Rendimiento de Naive Bayes.
LandmarkLDA	Rendimiento de LDA.
SkewnessSTD	Desviación estándar del sesgo de características.
SkewnessMean	Media del sesgo de características.
SkewnessMax	Máximo del sesgo de características.
SkewnessMin	Mínimo del sesgo de características.
KurtosisSTD	Desviación estándar de curtosis de características.
KurtosisMean	Media de curtosis de características.
KurtosisMax	Máximo de curtosis de características.

Continuación en la siguiente página

Meta-feature	**Descripción**
KurtosisMin	Mínimo de curtosis de características.
SymbolsSum	Suma de símbolos de características categóricas.
SymbolsSTD	Desviación estándar de símbolos de características categóricas.
SymbolsMean	Media de símbolos de características categóricas.
SymbolsMax	Máximo de símbolos de características categóricas.
SymbolsMin	Mínimo de símbolos de características categóricas.
ClassProbabilitySTD	Desviación estándar de probabilidades de clase.
ClassProbabilityMean	Media de probabilidades de clase.
ClassProbabilityMax	Máximo de probabilidades de clase.
ClassProbabilityMin	Mínimo de probabilidades de clase.
InverseDatasetRatio	Inverso del ratio dataset.
DatasetRatio	Ratio dataset.
RatioNominalToNumerical	Ratio de atributos nominales a numéricos.
RatioNumericalToNominal	Ratio de atributos numéricos a nominales.
NumberOfCategoricalFeatures	Número de atributos categóricos.
NumberOfNumericFeatures	Número de atributos numéricos.
NumberOfMissingValues	Número de valores faltantes.
NumberOfFeaturesWithMissingValues	Número de atributos con valores faltantes.
NumberOfInstancesWithMissingValues	Número de instancias con valores faltantes.
NumberOfFeatures	Número total de atributos.
NumberOfClasses	Número de clases.
NumberOfInstances	Número de instancias.
LogInverseDatasetRatio	Logaritmo del inverso del ratio dataset.
LogDatasetRatio	Logaritmo del ratio dataset.
PercentageOfMissingValues	Porcentaje de valores faltantes.
PercentageOfFeaturesWithMissingValues	Porcentaje de atributos con valores faltantes.
PercentageOfInstancesWithMissingValues	Porcentaje de instancias con valores faltantes.
LogNumberOfFeatures	Logaritmo del número de atributos.
LogNumberOfInstances	Logaritmo del número de instancias.

Detalles del tiempo computacional

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Perspectivas sobre el problema de AutoML

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Visualización de los nichos de los algoritmos de ML

Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Donec odio elit, dictum in, hendrerit sit amet, egestas sed, leo. Praesent feugiat sapien aliquet odio. Integer vitae justo. Aliquam vestibulum fringilla lorem. Sed neque lectus, consectetuer at, consectetuer sed, eleifend ac, lectus. Nulla facilisi. Pellentesque eget lectus. Proin eu metus. Sed porttitor. In hac habitasse platea dictumst. Suspendisse eu lectus. Ut mi mi, lacinia sit amet, placerat et, mollis vitae, dui. Sed ante tellus, tristique ut, iaculis eu, malesuada ac, dui. Mauris nibh leo, facilisis non, adipiscing quis, ultrices a, dui.

Resultados detallados en las tres tareas

Morbi luctus, wisi viverra faucibus pretium, nibh est placerat odio, nec commodo wisi enim eget quam. Quisque libero justo, consectetuer a, feugiat vitae, porttitor eu, libero. Suspendisse sed mauris vitae elit sollicitudin malesuada. Maecenas ultricies eros sit amet ante. Ut venenatis velit. Maecenas sed mi eget dui varius euismod. Phasellus aliquet volutpat odio. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Pellentesque sit amet pede ac sem eleifend consectetuer. Nullam elementum, urna vel imperdiet sodales, elit ipsum pharetra ligula, ac pretium ante justo a nulla. Curabitur tristique arcu eu metus. Vestibulum lectus. Proin mauris. Proin eu nunc eu urna hendrerit faucibus. Aliquam auctor, pede consequat laoreet varius, eros tellus scelerisque quam, pellentesque hendrerit ipsum dolor sed augue. Nulla nec lacus.

Comparación por pares con las meta-características de referencia

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

Análisis de sensibilidad de la dimensión d

Sed feugiat. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Ut pellentesque augue sed urna. Vestibulum diam eros, fringilla et, consectetuer eu, nonummy id, sapien. Nullam at lectus. In sagittis ultrices mauris. Curabitur malesuada erat sit amet massa. Fusce blandit. Aliquam erat volutpat. Aliquam euismod. Aenean vel lectus. Nunc imperdiet justo nec dolor.

Curvas de rendimiento de la Tarea 2

Etiam euismod. Fusce facilisis lacinia dui. Suspendisse potenti. In mi erat, cursus id, nonummy sed, ullamcorper eget, sapien. Praesent pretium, magna in eleifend egestas, pede pede pretium lorem, quis consectetuer tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdiet. Aliquam non quam. Aliquam porttitor quam a lacus. Praesent vel arcu ut tortor cursus volutpat. In vitae pede quis diam bibendum placerat. Fusce elementum convallis neque. Sed dolor orci, scelerisque ac, dapibus nec, ultricies ut, mi. Duis nec dui quis leo sagittis commodo.

Referencias

L. F. Alcobaça et al. Pymfe: A python library for meta-feature extraction. *Journal of Machine Learning Research*, 21(116):1–7, 2020.

David Alvarez-Melis and Nicolo Fusi. Optimal transport for structured data with application on graphs. In *Proceedings of the International Conference on Machine Learning*, volume 97, pages 6275–6284. PMLR, 2020.

Nicolas Courty, Rémi Flamary, Alain Habrard, and Alain Rakotomamonjy. Joint distribution optimal transportation for domain adaptation. In *Advances in Neural Information Processing Systems*, pages 3733–3742, 2017.

Trevor F. Cox and Michael A.A. Cox. *Multidimensional Scaling*. Chapman and Hall/CRC, Boca Raton, FL, USA, 2nd edition, 2001.

Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. 26, 2013.

Bradley Efron. *Bootstrap Methods: Another Look at the Jackknife*, volume 7. The Annals of Statistics, 1979.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2962–2970, 2015a.

Matthias Feurer et al. Efficient and robust automated machine learning. *Advances in Neural Information Processing Systems*, 2015b.

Nicolo Fusi, Rishit Sheth, and Huseyn Melih Elibol. Probabilistic matrix factorization for automated machine learning. In *Advances in Neural Information Processing Systems*, 2018.

Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michéle Sebag, Alexander Statnikov, WeiWei Tu, and Evelyne Viegas. Analysis of the automl challenge series 2015-2018. In *AutoML, Challenges in Machine Learning series*, pages 1–24. Springer, 2019.

Elad Hazan, Adam Klivans, and Yang Yuan. Hyperparameter optimization: A spectral approach. In *International Conference on Learning Representations (ICLR)*, 2018.

Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning: Methods, Systems, Challenges*. Springer, The Springer Series on Challenges in Machine Learning, 2019.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2015.

Marius Lindauer, Katharina Eggensperger, Matthias Feurer, André Biedenkapp, Johannes Marben, Paul Müller, and Frank Hutter. Boah: A tool suite for multi-fidelity bayesian optimization and analysis of hyperparameters. *arXiv preprint arXiv:1908.06756*, 2019.

Mustafa Misir and Michèle Sebag. Alors: An algorithm recommender system. *Artificial Intelligence*, 244:291–314, 2017.

Facundo Mémoli. Gromov–wasserstein distances and the metric approach to object matching. *Foundations of Computational Mathematics*, 11:417–487, 2011.

Bao Nguyen, Binh Nguyen, Hieu Trung Nguyen, and Viet Anh Nguyen. Generative conditional distributions by neural (entropic) optimal transport. In *Proceedings of the 41st International Conference on Machine Learning*, pages 37761–37775, 2024.

Gabriel Peyré and Marco Cuturi. *Computational optimal transport: With applications to data science*, volume 11. Foundations and Trends in Machine Learning, 2019.

Herilalaina Rakotoarison, Louisot Milijaona, Michèle Sebag, Andry Rasoanaivo, and Marc Schoenauer. Learning meta-features for automl. In *International Conference on Learning Representations (ICLR)*, 2022. Published as a conference paper at ICLR 2022.

John R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

Vayer Titouan, Nicolas Courty, Romain Tavenard, Laetitia Chapel, and Rémi Flamary. Optimal transport for structured data with application on graphs. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6275–6284. PMLR, 2019.

Renjun Xu, Pelen Liu, Liyan Wang, Chao Chen, and Jindong Wang. Reliable weighted optimal transport for unsupervised domain adaptation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4394–4403, 2020.

X. Xu, Z. Luo, M. Caron, and M. Sebag. Scaling up optimal transport for learning with large datasets: A proximal gradient approach. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

Chengrun Yang, Yuji Akimoto, Dae Won Kim, and Madeleine Udell. Oboe: collaborative filtering for automl model selection. In *ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1173–1183, 2019.