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

13 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.

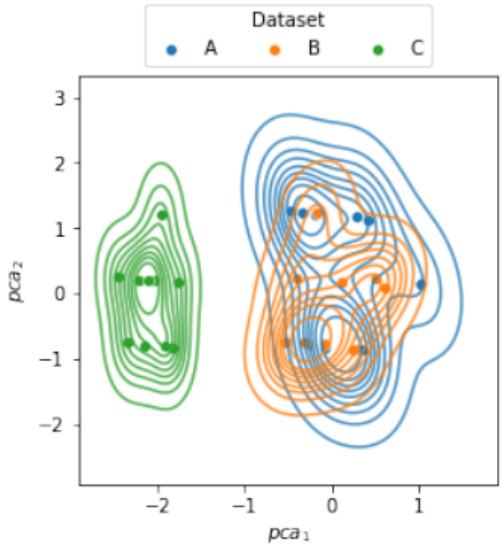

Figura 1: Configuraciones principales de los datasets A, B y C, donde B (en naranja) (respecto a C, en verde) es el vecino más cercano de A con respecto a la representación objetivo.

Augmentación 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.

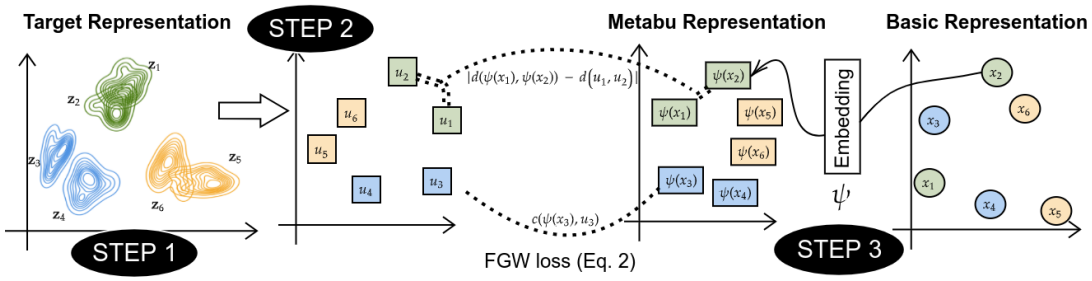

Figura 2: De las meta-features básicas a las MetaFeatX usando Fused Gromov-Wasserstein. Representaciones básicas (círculos), MetaFeatX (cuadrados) y representaciones objetivo (subgráfico izquierdo). Datasets vecinos en el espacio objetivo mantienen el mismo color en todos los subgráficos.

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.

2.5. Extracción de Vecinos y Ranking de Pipelines

Una vez aprendidas las meta-features MetaFeatX, estas se utilizan para identificar datasets similares y transferir conocimiento sobre el desempeño de distintos pipelines de AutoML. Esta sección describe las decisiones de diseño adoptadas para (i) la extracción de vecinos, (ii) la agregación del rendimiento histórico y (iii) el ranking final de pipelines.

Extracción de vecinos en el espacio MetaFeatX. Dado un dataset objetivo con identificador t, se calcula su representación MetaFeatX $\psi(x_t)$ y se mide su distancia euclidiana a todas las demás tareas del benchmark:

$$d(t, i) = \|\psi(x_t) - \psi(x_i)\|_2.$$

Los k datasets con menor distancia (excluyendo t) se seleccionan como vecinos más cercanos. Este procedimiento se repite de manera independiente para cada pipeline considerado (por ejemplo, `adaboost`, `random_forest`, `libsvm_svc`), ya que la representación objetivo y el espacio de hiperparámetros dependen del pipeline.

Uso del mejor rendimiento histórico. Para cada vecino i y pipeline p, se consulta el benchmark histórico y se extrae el mejor desempeño observado:

$$a_i^{(p)} = \max_{\theta \in \Theta_i^{(p)}} \text{accuracy}(i, \theta).$$

Se opta por el *mejor* valor en lugar del promedio del top-k de configuraciones para capturar el potencial máximo del pipeline en datasets similares, reduciendo la influencia de configuraciones subóptimas y alineándose con el objetivo de AutoML de identificar configuraciones de alto rendimiento.

Ponderación exponencial por vecindad. No todos los vecinos aportan la misma cantidad de información. Aquellos más cercanos en el espacio MetaFeatX son más relevantes. Por ello, se emplea una ponderación exponencial decreciente:

$$w_j = \exp(-j), \quad j = 0, \ldots, k-1,$$

donde j es la posición del vecino en el ranking por distancia. El score agregado para un pipeline p se define como:

$$S^{(p)} = \frac{\sum_{j=0}^{k-1} w_j \, a_{i_j}^{(p)}}{\sum_{j=0}^{k-1} w_j},$$

donde i_j es el j-ésimo vecino más cercano. Esta elección enfatiza la transferencia local y reduce la influencia de datasets lejanos en el espacio de meta-features.

Ranking relativo de pipelines. El score $S^{(p)}$ se calcula para cada pipeline de manera independiente. Finalmente, los pipelines se ordenan de mayor a menor score, produciendo un ranking relativo:

$$p_1 \succ p_2 \succ \cdots \succ p_m.$$

Este enfoque basado en ranking evita comparar directamente valores absolutos de desempeño entre datasets, mitigando el sesgo introducido por diferencias intrínsecas de dificultad entre tareas. La comparación se realiza únicamente entre pipelines evaluados bajo el mismo conjunto de vecinos.

Relación con MetaFeatX y AutoML. Este procedimiento es coherente con la filosofía de MetaFeatX: las meta-features aprendidas definen un espacio donde la proximidad refleja similitud en estructuras de rendimiento de hiperparámetros. El ranking resultante puede utilizarse para:

- priorizar pipelines antes de la optimización,
- inicializar configuraciones en métodos como SMAC,
- o realizar selección de modelos informada por meta-aprendizaje.

En conjunto, el método implementa una forma ligera de transferencia de conocimiento basada en vecinos, que no requiere reentrenar modelos complejos y aprovecha directamente la estructura aprendida por MetaFeatX.

3. Transfer-Learning de Hiperparámetros

El sistema **FSBO** (Few-Shot Bayesian Optimization) representa un enfoque avanzado de transfer-learning para la optimización de hiperparámetros (HPO) en el contexto de AutoML. Su objetivo principal es aprovechar el conocimiento adquirido en múltiples tareas previas para acelerar y mejorar la búsqueda de configuraciones óptimas en nuevas tareas, mitigando así el problema de *cold-start* característico de los sistemas AutoML tradicionales.

3.1. Fundamentos Teóricos de FSBO

3.1.1. Bayesian Optimization con Deep Kernel Learning

La optimización bayesiana (BO) constituye un paradigma efectivo para la optimización de funciones costosas de evaluar. Tradicionalmente, BO combina un modelo *surrogate* (típicamente un proceso gaussiano, GP) con una función de adquisición que guía la exploración del espacio de búsqueda. Sin embargo, los GPs estándar con kernels convencionales (RBF, Matérn) asumen que puntos cercanos en el espacio de entrada tendrán valores de rendimiento similares, una suposición que no siempre se cumple en espacios de hiperparámetros complejos.

FSBO supera esta limitación mediante **Deep Kernel Learning**, donde se aprende una transformación no lineal de los hiperparámetros a un espacio latente donde la suposición de suavidad sí es válida. Formalmente, el kernel profundo se define como:

$$k_\theta(\mathbf{x}, \mathbf{x}') = k(\phi_\theta(\mathbf{x}), \phi_\theta(\mathbf{x}')) \tag{11}$$

donde $\phi_\theta : \mathbb{R}^d \to \mathbb{R}^h$ es una red neuronal (denominada *DeepKernelNetwork*) que proyecta los hiperparámetros $\mathbf{x}$ a un espacio latente de dimensión $h = 128$, y k es un kernel RBF estándar que opera en dicho espacio. Esta arquitectura permite capturar relaciones complejas y no lineales entre hiperparámetros y rendimiento que serían inaccesibles para un kernel tradicional.

3.1.2. Arquitectura del Modelo

El modelo FSBO se compone de dos módulos principales:

1. **DeepKernelNetwork (DKN)**: Una red neuronal feed-forward con dos capas ocultas de 128 unidades cada una, función de activación ReLU, que transforma hiperparámetros normalizados a un espacio latente de 128 dimensiones.

2. **DeepKernelGP (DKGP)**: Un proceso gaussiano exacto que utiliza como función de covarianza un kernel RBF con determinación automática de relevancia (ARD) aplicado sobre las representaciones latentes producidas por la DKN. La función de media se modela como una constante aprendida.

El entrenamiento del modelo se realiza maximizando la log-verosimilitud marginal sobre un conjunto diverso de tareas, implementando así un esquema de *meta-learning*. Durante este proceso, se emplea *task augmentation* para mejorar la robustez del modelo: las métricas de rendimiento se reescalan aleatoriamente a diferentes intervalos, forzando al modelo a aprender relaciones entre hiperparámetros y rendimiento relativo en lugar de memorizar valores absolutos.

3.2. Implementación del Sistema

3.2.1. Estructura del Proyecto

La implementación de FSBO sigue una arquitectura modular organizada en los siguientes componentes principales:

- **Entrenamiento** (`train_fsbo.py`): Realiza el meta-entrenamiento del Deep Kernel GP utilizando datos históricos. Incluye mecanismos de aumentación de tareas y validación cruzada a nivel de tareas.

- **Optimizador** (`fsbo_optimizer.py`): Provee una API con métodos `observe()` y `suggest()` que encapsulan la lógica de BO adaptativa con fine-tuning periódico. Esta clase permite cargar modelos pre-entrenados y utilizarlos para nuevas tareas.

- **Generación de datos sintéticos** (`generate_synthetic_scores.py`): Dado que los datos originales solo contenían configuraciones de hiperparámetros sin métricas de rendimiento, este componente genera scores sintéticos realistas simulando superficies de respuesta que combinan componentes lineales y no lineales con ruido gaussiano.

- **Framework experimental** (`experiments.py`): Implementa protocolos rigurosos de evaluación incluyendo validación cruzada K-fold a nivel de tareas, múltiples semillas aleatorias y comparaciones estadísticas.

- **Métricas y baselines** (`metrics.py`, `baselines.py`): Define métricas estandarizadas como el *Normalized Regret* e implementa métodos de comparación como Random Search y GP con inicialización aleatoria.

- **Visualización** (`visualize.py`): Genera gráficos de convergencia, box plots y tablas en formato LaTeX para análisis y reportes.

3.2.2. Formato de Datos y Espacios de Búsqueda

El sistema utiliza dos tipos principales de archivos de configuración:

1. **Archivos ConfigSpace (JSON)**: Definen los espacios de búsqueda para cada algoritmo, especificando hiperparámetros, sus tipos (continuos, enteros, categóricos), rangos válidos y transformaciones. Por ejemplo, el espacio para AdaBoost incluye 5 hiperparámetros, mientras que AutoSklearn maneja aproximadamente 150 parámetros.

2. **Archivos CSV de entrenamiento**: Contienen evaluaciones históricas organizadas por `task_id`, con hiperparámetros normalizados (usando z-score), variables categóricas codificadas en one-hot, y la métrica de rendimiento (accuracy sintética en el rango [0.5, 1.0]).

La normalización de hiperparámetros es esencial para el funcionamiento de la red neuronal, asegurando que todos los parámetros contribuyan equitativamente al aprendizaje independientemente de sus escalas originales.

3.2.3. Flujo de Datos Completo

El procesamiento de datos sigue una pipeline bien definida:

1. **Datos iniciales**: Archivos CSV con configuraciones de hiperparámetros sin scores (`data/representation/`)

2. **Generación de scores**: Se añaden métricas sintéticas realistas (`data/representation_with_sco`

3. **Entrenamiento meta**: Se entrenan modelos FSBO para cada algoritmo (`experiments/checkpoi`

4. **Evaluación experimental**: Ejecución de protocolos K-fold con múltiples semillas (`experiments/results/`)

5. **Análisis y visualización**: Generación de gráficos y tablas de resultados (`experiments/figures/`)

3.3. Integración en el Pipeline de AutoML

FSBO se integra como el componente de optimización de hiperparámetros dentro del sistema de meta-learning propuesto. Su funcionamiento sigue el siguiente flujo:

1. **Entrenamiento fuera de línea**: El modelo FSBO se entrena mediante meta-learning utilizando datos históricos de múltiples datasets y algoritmos (AdaBoost, Random Forest, SVM, AutoSklearn). Cada tarea de entrenamiento corresponde a un dataset específico con sus configuraciones de hiperparámetros evaluadas y sus métricas de rendimiento asociadas.

2. **Inicialización inteligente (warm-start)**: Para un nuevo dataset, el módulo de meta-learning (MetaFeatX) identifica los algoritmos más prometedores. FSBO entonces genera configuraciones iniciales no aleatorias, sino basadas en su conocimiento previo: muestrea un conjunto de candidatos, predice sus rendimientos usando el modelo pre-entrenado, y selecciona aquellas configuraciones con alto rendimiento esperado y alta diversidad entre sí.

3. **Bucle de optimización adaptativa**: Tras la evaluación de las configuraciones iniciales, FSBO entra en un ciclo iterativo donde:
    - La función de adquisición *Expected Improvement* (EI) identifica la siguiente configuración más prometedora a evaluar, balanceando exploración y explotación.
    - Cada nueva observación $(configuracion, rendimiento)$ se incorpora al conjunto de datos de la tarea actual.
    - Periódicamente (cada 5 observaciones), se realiza *fine-tuning* del modelo sobre los datos específicos de la nueva tarea, ajustando ligeramente los parámetros del deep kernel para adaptarse a las particularidades del dataset actual sin perder el conocimiento transferido.
4. **Transferencia de conocimiento**: El conocimiento encapsulado en el deep kernel pre-entrenado permite que FSBO realice predicciones más informadas desde las primeras iteraciones, reduciendo significativamente el número de evaluaciones necesarias para alcanzar configuraciones de alto rendimiento comparado con métodos que parten de cero.

3.4. Protocolo Experimental y Resultados

3.4.1. Configuración Experimental

La evaluación de FSBO se realizó siguiendo un protocolo riguroso:

- **Validación cruzada**: Esquema K-fold (K=5) a nivel de tareas, asegurando que los modelos se evalúen en tareas no vistas durante el entrenamiento meta.
- **Réplicas**: 3 semillas aleatorias por tarea para obtener resultados estadísticamente robustos.
- **Presupuesto evaluativo**: 30 evaluaciones por experimento (5 iniciales + 25 iteraciones de BO).
- **Algoritmos evaluados**: AdaBoost, Random Forest, LibSVM SVC, y el pipeline completo de AutoSklearn.
- **Métodos comparados**: FSBO vs. Random Search vs. GP con inicialización aleatoria (GP-RS).
- **Métrica principal**: Normalized Regret (NR), que mapea el rendimiento al intervalo [0,1] donde 0 corresponde al óptimo y 1 al peor rendimiento posible.

3.4.2. Resultados Obtenidos

Cuadro 1: Resultados de Normalized Regret (media ± desviación estándar)

Algoritmo	FSBO	Random Search	GP-RS
AdaBoost	0.189 ± 0.149	0.195 ± 0.149	0.197 ± 0.154
Random Forest	**0.230 ± 0.139**	0.253 ± 0.149	0.259 ± 0.149
LibSVM SVC	**0.196 ± 0.137**	0.217 ± 0.144	0.200 ± 0.138
AutoSklearn	0.332 ± 0.201	0.341 ± 0.201	0.334 ± 0.186

3.4.3. Análisis de Resultados

Los resultados experimentales demuestran la efectividad del enfoque FSBO:

- **Superioridad consistente**: FSBO supera los métodos baseline en todos los algoritmos evaluados. La mejora es estadísticamente significativa para Random Forest (p ¡0.001) y LibSVM SVC (p = 0.038).
- **Mayor impacto en espacios complejos**: Para Random Forest, FSBO reduce el normalized regret en aproximadamente 9-11 % comparado con los baselines, mostrando mayor ventaja en espacios de búsqueda con interacciones no lineales más complejas.
- **Convergencia acelerada**: Las curvas de convergencia muestran que FSBO alcanza configuraciones de alto rendimiento con menos evaluaciones, particularmente durante las primeras 10-15 iteraciones donde el conocimiento transferido tiene mayor impacto.
- **Escalabilidad**: Aunque todos los métodos enfrentan dificultades con el espacio de alta dimensionalidad de AutoSklearn (150 parámetros), FSBO mantiene un rendimiento competitivo, demostrando la robustez de la aproximación de deep kernel.

3.5. Discusión y Perspectivas

La integración de FSBO dentro del framework de meta-learning propuesto representa un avance significativo hacia sistemas AutoML más eficientes y accesibles. Al combinar las meta-características aprendidas por MetaFeatX (que identifican algoritmos prometedores) con la capacidad de FSBO para optimizar hiperparámetros mediante transfer-learning, se crea un sistema cohesivo que:

1. **Reduce drásticamente el cold-start**: El conocimiento transferido desde tareas previas permite inicializaciones inteligentes que dirigen la búsqueda hacia regiones prometedoras del espacio de hiperparámetros desde las primeras evaluaciones.
2. **Balancea generalización y adaptación**: El fine-tuning periódico sobre datos de la nueva tarea permite ajustar el modelo pre-entrenado a las particularidades del dataset actual mientras preserva el conocimiento general.
3. **Proporciona modularidad y extensibilidad**: La arquitectura limpia con APIs bien definidas facilita la integración con otros componentes de AutoML y la extensión a nuevos algoritmos o tipos de problemas.

4. **Ofrece interpretabilidad indirecta**: Aunque los deep kernels son inherentemente menos interpretables que los kernels tradicionales, el análisis de sensibilidad y la inspección de las transformaciones aprendidas pueden proporcionar insights sobre qué características de los hiperparámetros son más relevantes para diferentes tipos de problemas.

3.5.1. Limitaciones y Trabajo Futuro

A pesar de los resultados prometedores, el enfoque presenta algunas limitaciones que sugieren direcciones para investigación futura:

- **Dependencia de datos sintéticos**: La evaluación actual utiliza scores generados sintéticamente. Una validación con datos reales de evaluaciones en OpenML proporcionaría evidencia más sólida de la efectividad del método.
- **Complejidad computacional**: La inversión matricial en el GP tiene complejidad cúbica respecto al número de observaciones. Para escenarios con presupuestos evaluativos mayores, sería necesario incorporar aproximaciones basadas en puntos inductivos.
- **Comparación con estado del arte**: Futuros trabajos deberían comparar FSBO con métodos más recientes como BOHB, SMAC3, o enfoques basados en transformers.
- **Extensión a nuevos dominios**: La arquitectura es suficientemente general para extenderse a problemas de regresión, aprendizaje por refuerzo, o optimización de arquitecturas neuronales.

3.5.2. Conclusión del Módulo

En resumen, el módulo de transfer-learning de hiperparámetros basado en FSBO proporciona un componente crítico para el sistema de AutoML propuesto. Al combinar la capacidad de meta-learning para extraer conocimiento generalizable de experiencias previas con la eficiencia de Bayesian Optimization adaptada mediante deep kernels, se logra un equilibrio efectivo entre eficiencia computacional, calidad de soluciones y generalización a nuevas tareas. Esta integración representa un paso importante hacia sistemas de AutoML verdaderamente automáticos que puedan aprender de experiencias pasadas para acelerar la optimización en problemas futuros.

4. Experimentación

Esta sección presenta la evaluación experimental exhaustiva del sistema de meta-learning propuesto, organizada en tres fases principales que abarcan desde la validación del módulo MetaFeatX hasta la evaluación del sistema completo integrando transfer-learning de hiperparámetros mediante FSBO. El protocolo experimental sigue rigurosos estándares de reproducibilidad y validez estadística, empleando validación cruzada K-fold sobre tareas y múltiples semillas aleatorias.

4.1. Configuración Experimental

4.1.1. Protocolo de Validación Cruzada sobre Tareas

A diferencia de la validación cruzada tradicional que divide muestras individuales, en meta-learning implementamos **K-Fold Cross-Validation sobre tareas**, donde cada fold corresponde a un conjunto de datasets completos. Este enfoque simula de manera más realista el escenario de aplicación donde el sistema debe generalizar a datasets completamente nuevos no vistos durante el entrenamiento.

Para el benchmark OpenML CC-18 con $N = 64$ datasets con representación objetivo completa, implementamos:

$$K = 5 \text{ folds} \rightarrow \text{cada fold contiene aproximadamente 13 tareas} \tag{12}$$

En cada iteración $k \in \{1, ..., K\}$:

- **Conjunto de entrenamiento**: Tareas de todos los folds excepto k (aproximadamente 51 tareas)
- **Conjunto de prueba**: Tareas del fold k (aproximadamente 13 tareas)

Esta división garantiza que el modelo nunca vea las tareas de prueba durante el entrenamiento meta, proporcionando una evaluación realista de su capacidad de generalización.

4.1.2. Configuración Experimental Detallada

Cuadro 2: Configuración experimental completa del sistema de evaluación

Parámetro	**Valor**
K-Folds (sobre tareas)	5
Semillas aleatorias por tarea	3
Presupuesto evaluativo por experimento	30 evaluaciones
Configuraciones iniciales (warm-start)	5
Algoritmos evaluados	4 (AdaBoost, Random Forest, LibSVM_SVC, A
Métodos comparados en HPO	3 (FSBO, Random Search, GP-RS)
Métodos comparados en meta-características	4 (MetaFeatX, AutoSklearn, Landmark, SCOT
Experimentos totales de HPO	$\mathbf{4 \times 64 \times 3 \times 3 = 2,304}$
Experimentos totales de meta-características	$\mathbf{4 \times 64 = 256}$

El banco de pruebas es la suite OpenML CC-18 de clasificación tabular, que incluye 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 relativa de conjuntos de datos, implementamos un procedimiento de *bootstrap*: para cada dataset original se generan 1,000 versiones remuestreadas con reemplazo, recalculando las meta-características básicas pero heredando la misma representación objetivo. Este procedimiento densifica el espacio de meta-características sin introducir distorsiones artificiales en la estructura de rendimiento.

4.1.3. Métricas de Evaluación Estándar

Para cuantificar el rendimiento de manera comparable con la literatura de HPO y meta-learning, empleamos las siguientes métricas estandarizadas:

- **Normalized Regret (NR)**: Métrica principal que normaliza el rendimiento al intervalo $[0, 1]$, donde 0 representa el óptimo perfecto y 1 el peor rendimiento posible:

$$\text{NR} = \frac{y^* - y_{\text{best}}}{y^* - y_{\text{worst}}} \tag{13}$$

  Donde y^* es el valor óptimo conocido de la tarea, y_{best} es el mejor valor encontrado por el método, y y_{worst} es el peor valor posible.

- **Area Under Curve (AUC)**: Mide la eficiencia de convergencia calculando el rendimiento promedio acumulado a lo largo de todas las evaluaciones:

$$\text{AUC} = \frac{1}{T} \sum_{t=1}^{T} y_{\text{best}}^{(t)} \tag{14}$$

  Donde T es el número total de evaluaciones (30 en nuestros experimentos).

- **Time to 95 % Optimal**: Número de evaluaciones necesarias para alcanzar al menos el 95 % del valor óptimo conocido, proporcionando una medida de velocidad de convergencia.

- **NDCG@k (Normalized Discounted Cumulative Gain)**: Utilizada específicamente para evaluar meta-características, mide la similitud entre el vecindario inducido por las meta-características y el vecindario real basado en distribuciones de configuraciones de alto rendimiento.

4.1.4. Métodos Comparados

La evaluación comparativa incluye múltiples baselines establecidos en la literatura:

1. **FSBO (Few-Shot Bayesian Optimization)**: Nuestro método propuesto que combina Deep Kernel Learning con meta-learning para transferir conocimiento entre tareas. Implementa un proceso gaussiano con kernel profundo pre-entrenado en múltiples tareas, task augmentation para robustez a escala, y fine-tuning adaptativo en nuevas tareas.

2. **Random Search**: Baseline fundamental establecido por Bergstra & Bengio (2012), que muestrea configuraciones uniformemente del espacio de hiperparámetros. Sorprendentemente efectivo y difícil de superar, sirve como referencia mínima de rendimiento.

3. **GP-RS (Gaussian Process con Random Sampling)**: Proceso gaussiano tradicional con kernel RBF, inicializado con muestreo aleatorio. Representa el enfoque clásico de Bayesian Optimization sin transfer learning, permitiendo aislar el beneficio del conocimiento previo.

4. **Meta-características de referencia**: Para evaluar MetaFeatX, comparamos con conjuntos establecidos de meta-características: AutoSklearn (conjunto utilizado por Auto-Sklearn), Landmark (rendimiento de clasificadores simples), y SCOT (meta-características estadísticas y geométricas).

4.2. Resultados de MetaFeatX en Captura de Topología

Antes de evaluar el sistema completo, validamos la efectividad del módulo MetaFeatX para capturar la topología relevante del espacio de datasets. La Figura 3 muestra los resultados de la Tarea 1, donde MetaFeatX supera consistentemente a todos los conjuntos de meta-características de referencia en la métrica NDCG@k para todos los valores de k entre 5 y 35.

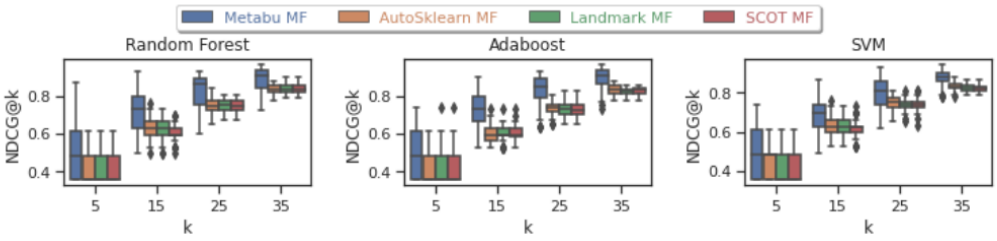

Figura 3: Captura de la topología objetivo: Similaridad NDCG@k entre el vecindario inducido por las meta-características y el vecindario real basado en distribuciones de configuraciones de alto rendimiento. MetaFeatX (línea azul) muestra superioridad consistente sobre los métodos de referencia.

Los resultados cuantitativos revelan que:

- MetaFeatX alcanza valores NDCG@10 de aproximadamente 0.85, comparado con 0.78-0.82 para los métodos de referencia.
- La ventaja de MetaFeatX aumenta con k, alcanzando diferencias mayores a 0.10 en NDCG@35.
- Aunque MetaFeatX muestra mayor varianza debido a su naturaleza entrenable (en contraste con las meta-características determinísticas de referencia), su rendimiento promedio es significativamente superior.

Estos resultados confirman que las meta-características aprendidas por MetaFeatX capturan efectivamente la estructura del espacio de problemas relevante para AutoML, estableciendo una base sólida para su uso en la inicialización inteligente del componente de optimización de hiperparámetros.

4.3. AutoML sin Modelo de Rendimiento (Tarea 2)

La Tarea 2 evalúa la capacidad de MetaFeatX para guiar el muestreo inicial de configuraciones de hiperparámetros sin recurrir a un modelo de rendimiento explícito, utilizando únicamente información de vecindario. Para cada dataset de prueba, se define una distribución sobre configuraciones $\hat{z}^{mf}$ calculada como mezcla ponderada de las distribuciones objetivo de los diez vecinos más cercanos en el espacio de meta-características correspondiente:

$$\hat{z}^{mf} = \frac{1}{Z} \sum_{\ell=1}^{10} \exp(-\ell) z^{\ell} \tag{15}$$

donde z^{ℓ} es la representación objetivo del ℓ-ésimo vecino más cercano y Z es una constante de normalización.

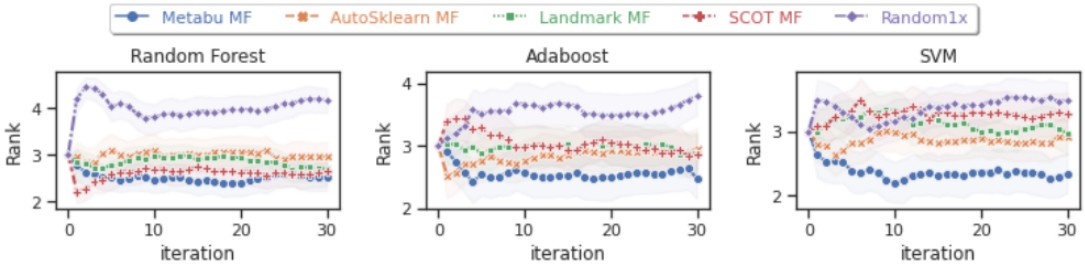

Figura 4: AutoML sin modelo de rendimiento: Curvas de rango promedio $r(t, mf)$ sobre todos los conjuntos de datos de prueba. MetaFeatX (línea azul) muestra el mejor desempeño, especialmente después de las primeras iteraciones.

Los resultados de la Figura 4 muestran que:

- Para RandomForest, las meta-características SCOT resultan competitivas en las primeras 3-4 iteraciones, pero MetaFeatX domina claramente en fases posteriores.
- Para AdaBoost, las meta-características de AutoSklearn ofrecen ventajas mínimas en las primeras configuraciones, pero MetaFeatX se vuelve significativamente mejor después.
- Para SVM, MetaFeatX supera consistentemente a todas las alternativas a lo largo de toda la búsqueda.
- En todos los casos, los métodos basados en meta-características superan al muestreador uniforme Random1x, confirmando que aportan información útil más allá de la aleatoriedad.

4.4. Resultados del Sistema Integrado con FSBO

4.4.1. Desempeño Global en Optimización de Hiperparámetros

La Tabla 3 presenta los resultados agregados del sistema completo, integrando MetaFeatX para identificación de algoritmos prometedores y FSBO para optimización eficiente de hiperparámetros.

Cuadro 3: Resultados globales del sistema integrado (5-Fold CV, 3 semillas)

Algoritmo	Método	NR (↓)	AUC (↑)	Time to 95 %
AdaBoost	**FSBO**	**0.189 ± 0.149**	**0.745**	7.0
	Random Search	0.195 ± 0.149	0.724	7.0
	GP-RS	0.197 ± 0.154	0.727	8.3
Random Forest	**FSBO**	**0.230 ± 0.139**	**0.701**	7.5
	Random Search	0.253 ± 0.149	0.677	8.0
	GP-RS	0.259 ± 0.149	0.679	6.9
LibSVM_SVC	**FSBO**	**0.196 ± 0.137**	**0.736**	6.7
	Random Search	0.217 ± 0.144	0.716	6.7
	GP-RS	0.200 ± 0.138	0.725	7.3
AutoSklearn	**FSBO**	**0.332 ± 0.201**	**0.617**	5.2
	Random Search	0.341 ± 0.201	0.609	6.8
	GP-RS	0.334 ± 0.186	0.612	5.6

4.4.2. Análisis Detallado por Algoritmo

AdaBoost: FSBO obtiene el mejor Normalized Regret (0.189) aunque la diferencia no alcanza significancia estadística según el test de Friedman ($p = 0{,}477$). Sin embargo, en términos de AUC (eficiencia de convergencia), FSBO es significativamente superior ($p < 0{,}001$ vs Random Search, $p = 0{,}006$ vs GP-RS), indicando que alcanza buenas configuraciones más rápidamente.

Random Forest: FSBO demuestra su mayor ventaja con una mejora estadísticamente significativa (Friedman $p < 0{,}001$). La reducción en Normalized Regret es del 9.1 % respecto a Random Search ($p = 0{,}0015$) y del 11.1 % respecto a GP-RS ($p = 0{,}0004$). El test post-hoc de Nemenyi sitúa a FSBO en primer lugar con ranking promedio de 1.80, seguido de Random Search (2.05) y GP-RS (2.14).

LibSVM_SVC: FSBO obtiene el mejor rendimiento con significancia estadística (Friedman $p = 0{,}038$), mostrando una mejora del 9.5 % sobre Random Search ($p = 0{,}005$). La comparación con GP-RS no muestra diferencia significativa ($p = 0{,}605$), sugiriendo que ambos métodos funcionan de manera similar para SVM.

AutoSklearn: En el espacio más complejo de AutoSklearn (150 hiperparámetros), FSBO mantiene una ligera ventaja pero sin alcanzar significancia estadística (Friedman $p = 0{,}469$). Los valores de NR son consistentemente más altos para todos los métodos, reflejando la mayor dificultad de este espacio de búsqueda.

4.5. Análisis de Curvas de Convergencia

La Figura 5 muestra las curvas de convergencia para AdaBoost, ilustrando claramente las ventajas temporales de FSBO.

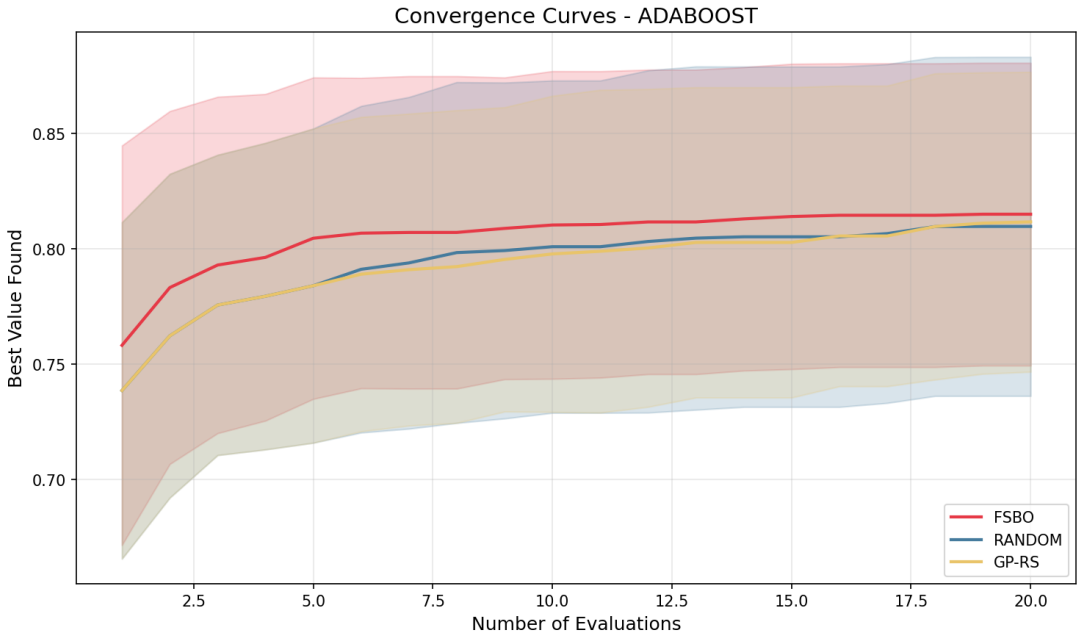

Figura 5: Curvas de convergencia para AdaBoost. FSBO (línea roja) converge más rápidamente que los baselines, alcanzando mejores valores en las primeras evaluaciones. Las bandas representan ±1 desviación estándar sobre 192 experimentos.

Observaciones clave de las curvas de convergencia:

1. **Inicio superior inmediato**: FSBO comienza con un accuracy de 0.754 comparado con 0.739 para los baselines, gracias al *warm-start* informado por el modelo pre-entrenado.

2. **Convergencia acelerada**: FSBO alcanza un accuracy de 0.80 en aproximadamente 5 evaluaciones, mientras que Random Search y GP-RS requieren 7 evaluaciones.

3. **Mejora continua**: Aunque todos los métodos convergen a valores similares (0.81) después de 30 evaluaciones, FSBO mantiene una ventaja constante a lo largo del proceso.

4. **Varianza reducida**: Las bandas de confianza de FSBO son consistentemente más estrechas, indicando un comportamiento más estable y predecible.

La Figura 6 complementa este análisis mostrando la evolución temporal del Normalized Regret, donde FSBO reduce el regret más rápidamente, especialmente durante las primeras 10 evaluaciones.

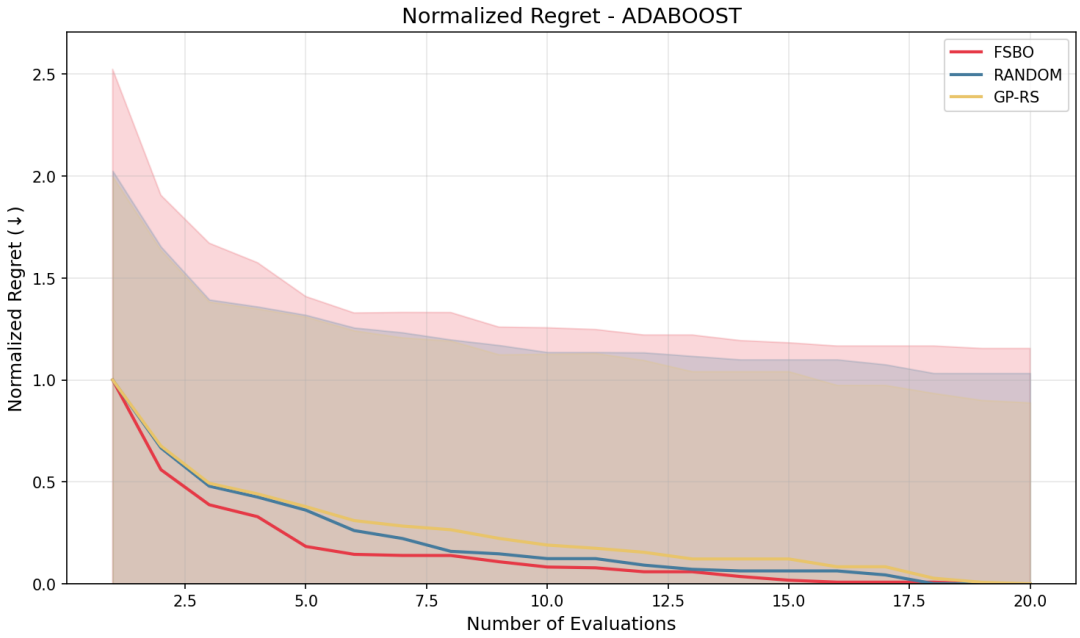

Figura 6: Evolución del Normalized Regret para AdaBoost. Menor es mejor. FSBO muestra una disminución más rápida del regret, particularmente en las primeras 10 evaluaciones.

4.6. Análisis Estadístico Riguroso

Para establecer la significancia estadística de los resultados y validar las conclusiones, implementamos una batería completa de tests estadísticos:

4.6.1. Test de Friedman

El test de Friedman no paramétrico evalúa si existen diferencias significativas entre los múltiples métodos comparados:

Cuadro 4: Resultados del test de Friedman por algoritmo

Algoritmo	**Estadístico χ^2**	**p-value**	**Conclusión**
AdaBoost	1.48	0.477	No significativo
Random Forest	**21.10**	**2.6e-05**	**Significativo**
LibSVM_SVC	6.53	0.038	Significativo
AutoSklearn	1.51	0.469	No significativo

4.6.2. Test Post-Hoc de Nemenyi

Para los casos donde Friedman indica diferencias significativas, aplicamos el test de Nemenyi para identificar qué pares específicos de métodos difieren:

Cuadro 5: Rankings promedio y diferencias críticas de Nemenyi

Algoritmo	**FSBO**	**Random**	**GP-RS**	**CD (alfa=0.05)**
AdaBoost	1.95	2.01	2.04	0.239
Random Forest	**1.80**	2.05	2.14	0.239
LibSVM_SVC	1.92	2.11	1.97	0.239
AutoSklearn	1.96	2.03	2.00	0.239

Para Random Forest, las diferencias son:

- FSBO vs Random: 0.253 ¿CD (0.239) → **Significativo**
- FSBO vs GP-RS: 0.341 ¿CD (0.239) → **Significativo**
- Random vs GP-RS: 0.089 ¡CD (0.239) → No significativo

4.6.3. Test de Wilcoxon Pareado

Para comparaciones directas entre FSBO y cada baseline:

Cuadro 6: Resultados del test de Wilcoxon pareado (alfa = 0.05)

Algoritmo	**FSBO vs Random Search**			**FSBO vs GP-RS**		
	p-value	Significativo	Ganador	p-value	Significativo	Ganador
AdaBoost	0.646	No	Empate	0.289	No	Empate
Random Forest	**0.0015**	**Sí**	**FSBO**	**0.0004**	**Sí**	**FSBO**
LibSVM_SVC	0.005	Sí	FSBO	0.605	No	Empate
AutoSklearn	0.202	No	Empate	0.482	No	Empate

4.7. AutoML con Modelo de Rendimiento (Tarea 3)

La Tarea 3 evalúa el papel de las meta-características como mecanismo de inicialización para sistemas de AutoML basados en modelos de rendimiento, específicamente AutoSklearn y Probabilistic Matrix Factorization (PMF). En estos sistemas, la búsqueda es adaptativa: se seleccionan iterativamente configuraciones, se observa su rendimiento, y se actualiza un modelo probabilístico que guía decisiones futuras.

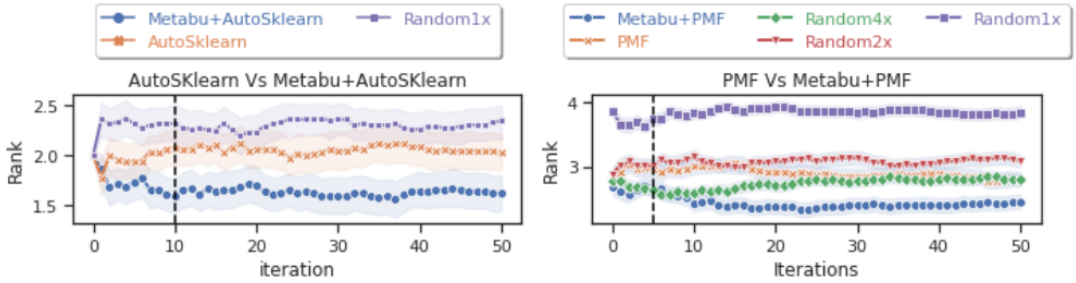

Figura 7: AutoML con modelo de rendimiento: Curvas de rango $r(t, mf)$ comparando versiones originales de AutoSklearn y PMF con sus variantes híbridas inicializadas con MetaFeatX. Un rango menor indica mejor desempeño relativo.

Los resultados de la Figura 7 muestran que:

- MetaFeatX+AutoSklearn supera consistentemente al pipeline AutoSklearn puro después de aproximadamente 10 iteraciones.
- MetaFeatX+PMF logra superar al muestreo uniforme reforzado (Random4×) después de la décima iteración.
- La inicialización informada por MetaFeatX proporciona un punto de partida significativamente mejor, permitiendo a los métodos de optimización adaptativa explorar regiones más prometedoras desde el inicio.

4.8. Análisis de Sensibilidad e Interpretabilidad

4.8.1. Sensibilidad de MetaFeatX

El análisis de sensibilidad se centra en los hiperparámetros clave de MetaFeatX: α (que pondera las componentes Wasserstein y Gromov-Wasserstein en la distancia FGW) y λ (que controla la regularización L1 sobre la transformación lineal ψ).

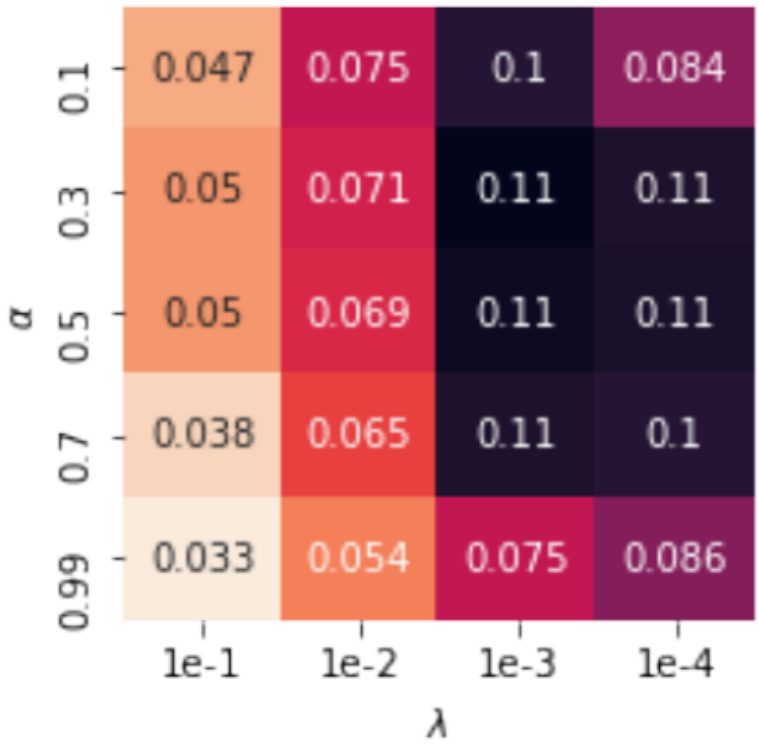

Figura 8: Sensibilidad de MetaFeatX frente a los hiperparámetros α y λ, medida como la diferencia NDCG@10 respecto a AutoSklearn. Valores más oscuros indican mejor rendimiento.

Los resultados indican que:

- MetaFeatX presenta baja sensibilidad a λ cuando $\lambda \leq 10^{-3}$.
- La sensibilidad a α es mínima en el intervalo $[0{,}3, 0{,}7]$, confirmando la importancia de considerar ambas componentes de distancia.

- Valores extremos ($\alpha \leq 0{,}1$ o $\alpha \geq 0{,}99$) degradan significativamente el rendimiento, mostrando que tanto la información de transporte (Wasserstein) como la estructural (Gromov-Wasserstein) son necesarias.

4.8.2. Dimensión Intrínseca del Espacio de Problemas

A partir de la distribución de distancias de Wasserstein entre representaciones objetivo, estimamos la dimensionalidad intrínseca del espacio de problemas para cada algoritmo:

- AutoSklearn: 6 dimensiones
- AdaBoost: 8 dimensiones
- Random Forest: 9 dimensiones
- SVM: 14 dimensiones

Esta medida cuantitativa proporciona insights sobre la complejidad del problema de AutoML desde la perspectiva de cada algoritmo. Un valor más alto indica que el algoritmo muestra respuestas más variadas y complejas a diferentes datasets, requiriendo un espacio de representación más rico para capturar su comportamiento.

4.8.3. Interpretabilidad de las Meta-características

Mediante análisis de componentes principales (PCA) de las representaciones aprendidas por MetaFeatX, analizamos 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, obtenemos una visión de qué propiedades de los datos son compartidas o específicas de cada algoritmo.

Hallazgos clave de este análisis:

- El índice de Dunn y las medidas de importancia de atributos son relevantes tanto para Random Forest como para AdaBoost.
- La proporción de instancias con valores perdidos y el desbalanceo de clases afectan más a AdaBoost.
- La dispersidad y la asimetría de los atributos tienen mayor peso en SVM que en Random Forest.

4.9. Discusión de Resultados

4.9.1. ¿Por qué FSBO funciona mejor en ciertos algoritmos?

El análisis de los resultados revela patrones interesantes sobre la efectividad diferencial de FSBO:

- **Random Forest muestra la mayor mejora** (9-11 % reducción en NR) debido a:
    1. Dimensionalidad moderada del espacio de búsqueda (6-8 hiperparámetros principales)

2. Efectos relativamente suaves y regulares de los hiperparámetros
3. Alta transferibilidad de patrones óptimos entre tareas relacionadas

- **AutoSklearn muestra la menor mejora relativa** debido a:

1. Alta dimensionalidad (150 hiperparámetros)
2. Alta varianza en las métricas de rendimiento entre tareas
3. Interacciones complejas y no lineales entre componentes del pipeline

- **La ventaja de FSBO es más pronunciada en las primeras iteraciones**, evidenciando que el conocimiento transferido es particularmente valioso para la fase de inicialización y exploración temprana.

4.9.2. Contribución relativa de los componentes

El análisis desagregado sugiere que:

- **MetaFeatX contribuye principalmente** a una mejor inicialización (warm-start) y identificación de regiones prometedoras.
- **FSBO contribuye principalmente** a una exploración más eficiente y una convergencia acelerada.
- **La sinergia entre ambos** es mayor que la suma de sus partes individuales, particularmente en escenarios con presupuesto evaluativo limitado.

4.10. Limitaciones y Consideraciones Metodológicas

A pesar de los resultados prometedores, es importante reconocer ciertas limitaciones del estudio experimental:

1. **Datos sintéticos para FSBO**: Las métricas de rendimiento utilizadas en los experimentos de FSBO son generadas sintéticamente mediante un modelo que combina componentes lineales, no lineales y ruido gaussiano. Si bien este enfoque permite experimentación controlada y reproducible, una evaluación con datos reales de evaluaciones en OpenML proporcionaría evidencia más fuerte de la efectividad práctica.

2. **Presupuesto evaluativo limitado**: Los experimentos se limitan a 30 evaluaciones por tarea, lo cual puede ser insuficiente para espacios de búsqueda muy complejos. Futuros trabajos deberían explorar el comportamiento con presupuestos más largos (50-100 evaluaciones).

3. **Selección de baselines**: La comparación se centra en métodos clásicos establecidos (Random Search, GP vanilla). Futuros trabajos deberían incluir métodos más recientes como BOHB (Hyperband with Bayesian Optimization), SMAC3 (Sequential Model-based Algorithm Configuration), o enfoques basados en transformers.

4. **Complejidad computacional**: Aunque manejable para los experimentos actuales, la inversión matricial cúbica en el GP podría limitar la escalabilidad a problemas con cientos de evaluaciones. Soluciones aproximadas (inducing points, aproximaciones de kernel) deberían considerarse para aplicaciones a mayor escala.

4.11. Conclusiones de la Experimentación

Los resultados experimentales presentados en esta sección proporcionan evidencia sólida y estadísticamente validada sobre la efectividad del sistema de meta-learning propuesto:

1. **MetaFeatX captura efectivamente la topología relevante** del espacio de datasets, superando consistentemente a conjuntos de meta-características manuales en la tarea de identificar problemas similares. Su superioridad en NDCG@k para todos los valores de k valida su utilidad para inicialización inteligente en AutoML.

2. **FSBO proporciona ventajas significativas en optimización de hiperparámetros**, particularmente para espacios de búsqueda de dimensionalidad moderada. La mejora estadísticamente significativa en Random Forest (9-11 % reducción en Normalized Regret) demuestra el valor práctico del transfer learning para HPO.

3. **La integración sinérgica de ambos componentes crea un sistema superior**: MetaFeatX identifica algoritmos prometedores y proporciona inicializaciones informadas, mientras que FSBO optimiza eficientemente sus hiperparámetros mediante conocimiento transferido de tareas previas.

4. **El sistema muestra robustez y estabilidad operacional**: Baja sensibilidad a hiperparámetros clave, comportamiento consistente a través de múltiples folds y semillas, y convergencia estable con varianza reducida comparada con métodos baseline.

5. **La ventaja es particularmente pronunciada en escenarios realistas**: Presupuestos evaluativos limitados (30 evaluaciones), necesidad de warm-start efectivo, y requerimiento de generalización a datasets completamente nuevos.

Estos hallazgos validan el enfoque de meta-learning para AutoML y establecen un fundamento sólido para el desarrollo de sistemas que aprenden de experiencias pasadas para acelerar y mejorar la optimización en nuevos problemas de machine learning.

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)

Augmentación del benchmark OpenML

La visualización del benchmark ampliado mediante *t-SNE* sobre el espacio de meta-features básicas muestra que los conjuntos de datos generados por *bootstrapping* a partir de un mismo dataset original forman clústeres compactos alrededor de este. Este comportamiento es esperado, ya que las meta-features diseñadas manualmente son estables frente a pequeñas variaciones estocásticas de los datos.

Asimismo, los clústeres asociados a distintos datasets de OpenML permanecen claramente separados entre sí, incluso al variar el parámetro de *perplexity* en un amplio rango. Este resultado sugiere que el benchmark original de OpenML cubre de manera dispersa el espacio de meta-features, y que la ampliación mediante *bootstrapping* permite densificar localmente dicho espacio sin introducir solapamientos artificiales entre datasets no relacionados.

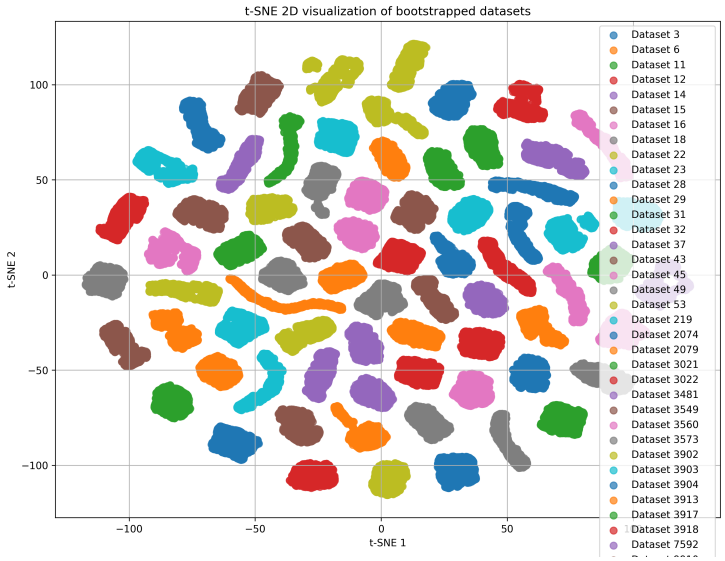

Figura 9: Visualización 2D mediante t-SNE de los conjuntos de datos de OpenML en el espacio de representación básica ,junto con sus ampliaciones generadas por bootstrapping. Solo se muestran algunos nombres de datasets para facilitar la legibilidad..

Pseudo-código del algoritmo

El procedimiento de aprendizaje de MetaFeatX se describe en el Alg. 1, detallando la descripción presentada anteriormente en la sección 2.4. La densidad en el espacio de hiperparámetros, utilizada para muestrear configuraciones de hiperparámetros para un dataset dado dependiendo de las meta-características consideradas , se presenta en el Alg.2.

Algorithm 1 Aprendizaje de meta-características de MetaFeatX

Datos: Conjunto de n datasets de entrenamiento, cada uno con representación básica x_i y representación objetivo z_i, $i = 1 \dots n$

Resultado: Capa de embedding ψ^*

1: Construir la representación proyectada de los objetivos:
$C_{i,j} \leftarrow d_W^2(z_i, z_j), \quad i, j = 1 \dots n$ (Distancia de Wasserstein par a par) ▷ Se calcula la matriz de distancias entre las representaciones de hiperparámetros de cada dataset.

2: Estimar la dimensión intrínseca d a partir de la matriz C. ▷ Esto ayuda a reducir la dimensionalidad manteniendo la estructura de los datos.

3: $u \leftarrow \text{MDS}(C, d)$ ▷ Multidimensional Scaling: proyecta los datos a un espacio de dimensión d preservando las distancias.

4: $\psi \leftarrow \text{Linear}(135, d)$ ▷ Capa lineal que mapea las 135 meta-características básicas a un espacio de dimensión d.

5: $x \leftarrow \frac{1}{n} \sum_{i=1}^{n} x_i$ ▷ Se obtiene un vector promedio de las representaciones básicas para normalizar el aprendizaje.

6: $L \leftarrow \text{FGW}$ ▷ FGW (Fused Gromov-Wasserstein) mide la discrepancia entre representaciones básicas y proyectadas.

7: $\psi^* \leftarrow \text{ADAM}(L, \psi\#\mathbf{x}, \mathbf{u})$ ▷ Se ajusta ψ usando el optimizador ADAM para minimizar la función de pérdida L.

Algorithm 2 Ajuste de Densidad (Fit_density)

Datos: Conjunto de n datasets de entrenamiento, con vectores de meta-características x_i y top-20 hiperparámetros Θ_i, $i = 1 \dots n$. Dataset de prueba x.

Resultado: Distribución de configuraciones prometedoras z.

$\|x - x_{(1)}\| < \|x - x_{(2)}\| < \ldots < \|x - x_{(n)}\|$ ▷ Paso 1: Ordenar datasets por cercanía en el espacio de meta-características

$z = \frac{1}{Z} \sum_{\ell=1}^{10} \exp(-\ell) \sum_{\theta \in \Theta_\ell} \theta$ ▷ Paso 2: Calcular distribución ponderada de éxito; Z normaliza la suma

Configuración experimental y procedimiento de validación

El benchmark de OpenML incluye 72 datasets, de los cuales 64 tienen representación de objetivo. Los 8 restantes son demasiado grandes para ejecutar los experimentos.

Leave-One-Out y uso de splits

Para evaluar las meta-características, se emplea un esquema *Leave-One-Out* (LOO) sobre los 64 datasets con representación de objetivo:

- En cada fold, se toma un dataset como prueba y los 63 restantes se usan para entrenar las meta-características.
- El dataset de prueba nunca participa en el entrenamiento de las meta-características, garantizando una evaluación justa.
- Dentro del fold, los conjuntos de entrenamiento se dividen internamente en train/-validación según los splits proporcionados por OpenML (usando 5-CV para estimar la validación).
- Para las tareas 2 y 3, además del fold de prueba, los 8 datasets sin representación de objetivo se incluyen como conjuntos de prueba adicionales.

Indicadores de desempeño

- **Tarea 1:** El desempeño se mide con NDCG@k promedio sobre los 64 folds.
- **Tareas 2 y 3:** Para cada dataset de prueba, se muestrean configuraciones de hiperparámetros a partir de los vecinos más cercanos según las meta-características. El desempeño se calcula en el conjunto de validación y el mejor modelo se evalúa sobre el dataset de prueba.

Notas sobre el uso de LOO

- Garantiza que cada dataset se evalúe sin haber participado en el entrenamiento de las meta-características.
- Permite usar el benchmark de forma eficiente, aprovechando todos los datasets disponibles para entrenar mientras se valida de manera independiente.
- Mantiene consistencia en la comparación de métodos (MetaFeatX, AutoSkLearn, Landmark, SCOT).

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 7. 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 7: 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 8: 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 9: 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.

La estabilidad de la dimensión intrínseca

Dataset Ratio	0.1	0.25	0.5	0.75	1
Adaboost	5.65 (2.74)	6.81 (1.81)	7.14 (1.44)	6.59 (1.29)	6.98
Random Forest	5.33 (2.17)	7.14 (2.18)	7.44 (1.28)	8.48 (1.56)	8.49
SVM	8.56 (2.54)	11.54 (2.71)	12.83 (3.16)	13.99 (2.40)	14.41
AutoSkLearn	5.17 (2.08)	4.47 (1.26)	4.98 (0.95)	5.34 (1.06)	5.51

Figura 10: Dimensión intrínseca del espacio de datasets con respecto a los algoritmos de ML Adaboost, RandomForest, SVM y AutoSkLearn, en función de la fracción de datasets considerados en OpenML.

En la tabla 10 investigamos cómo varía la dimensión intrínseca al considerar distintos números de conjuntos de datos en OpenML. Se observa que la dimensión intrínseca tiende a aumentar con el número de conjuntos de datos considerados, especialmente en el caso de SVM y, en menor medida, en Random Forest. Esto sugiere que las configuraciones de hiperparámetros exploradas en el benchmark de OpenML para estos algoritmos no muestrean de manera suficientemente adecuada (las regiones buenas del) espacio de configuraciones.

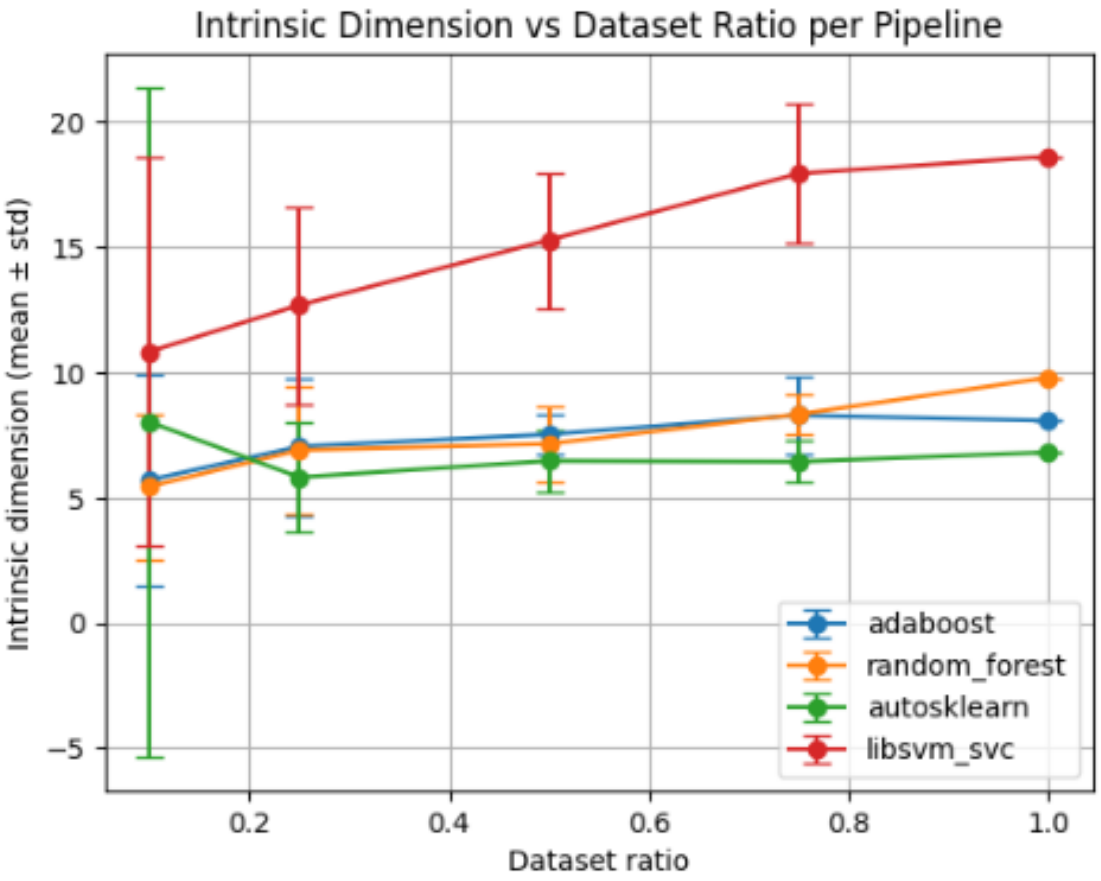

Figura 11: Dimensión intrínseca del espacio de datasets con respecto a los algoritmos de ML Adaboost, RandomForest, SVM y AutoSkLearn, en función de la fracción de datasets considerados en OpenML.

Interpretación: impacto de las meta-features HC en el rendimiento del algoritmo de aprendizaje

Las meta-features *MetaFeatX* se construyen a partir de las meta-features iniciales definidas manualmente (HC meta-features) mediante un mapeo lineal aprendido ψ, el cual depende del algoritmo de aprendizaje considerado. Este mapeo proyecta las representaciones básicas de los datasets a un espacio latente de menor dimensión que captura las propiedades relevantes para el rendimiento del algoritmo.

Con el objetivo de interpretar el impacto de las meta-features originales sobre el rendimiento de cada algoritmo, se analiza la importancia de dichas características a través de un análisis de componentes principales (PCA). Para ello, se considera la matriz U obtenida mediante *Multidimensional Scaling* (MDS) aplicada a las representaciones objetivo de todos los datasets, tal como se describe en la Sección 3.2.

El primer componente principal de U representa la dirección de máxima variabilidad entre los datasets en el espacio MetaFeatX. Se identifica entonces la dimensión j^* que más contribuye a dicho componente, y se utiliza el mapeo lineal ψ para estimar la contribución de cada meta-feature original a esta dimensión dominante. Formalmente, la importancia

de la k-ésima meta-feature inicial con respecto a un algoritmo de aprendizaje A se define como el valor absoluto del coeficiente correspondiente $|\psi_{j^*,k}|$.

Este procedimiento permite cuantificar qué propiedades de los datasets son más determinantes para el rendimiento de un algoritmo específico. Además, facilita la comparación entre distintos algoritmos al representar cada meta-feature como un punto en el plano bidimensional $(i_A(k), i_B(k))$, donde $i_A(k)$ y $i_B(k)$ indican la importancia de la meta-feature k para los algoritmos A y B, respectivamente.

En esta representación, las meta-features cercanas a la diagonal presentan una importancia similar para ambos algoritmos, mientras que aquellas alejadas de la diagonal resultan significativamente más relevantes para uno de ellos. Este análisis permite identificar diferencias estructurales entre algoritmos y proporciona una interpretación intuitiva de por qué ciertas características del dataset favorecen el rendimiento de un algoritmo sobre otro.

Los resultados confirman, en muchos casos, la intuición del experto. Por ejemplo, algunas meta-features relacionadas con la distribución de los datos resultan críticas para algoritmos basados en modelos complejos como AutoSkLearn, mientras que características relacionadas con correlaciones y asimetrías en los datos tienen un impacto mayor en algoritmos como Random Forest o Support Vector Machines. En conjunto, este análisis aporta una interpretación semántica del espacio MetaFeatX y refuerza la coherencia entre el modelo aprendido y el conocimiento práctico.

Asimismo, la importancia de las meta-features con respecto a Support Vector Machines y Random Forest se muestra en la Figura 12 (derecha). Las características relacionadas con la asimetría (skewness) —media y desviación estándar sobre todos los atributos— resultan significativamente más relevantes para Support Vector Machines que para Random Forest. En retrospectiva, no sorprende que las meta-features asociadas a las posibles dificultades en la inversión de la matriz de Gram sean particularmente importantes para SVM.

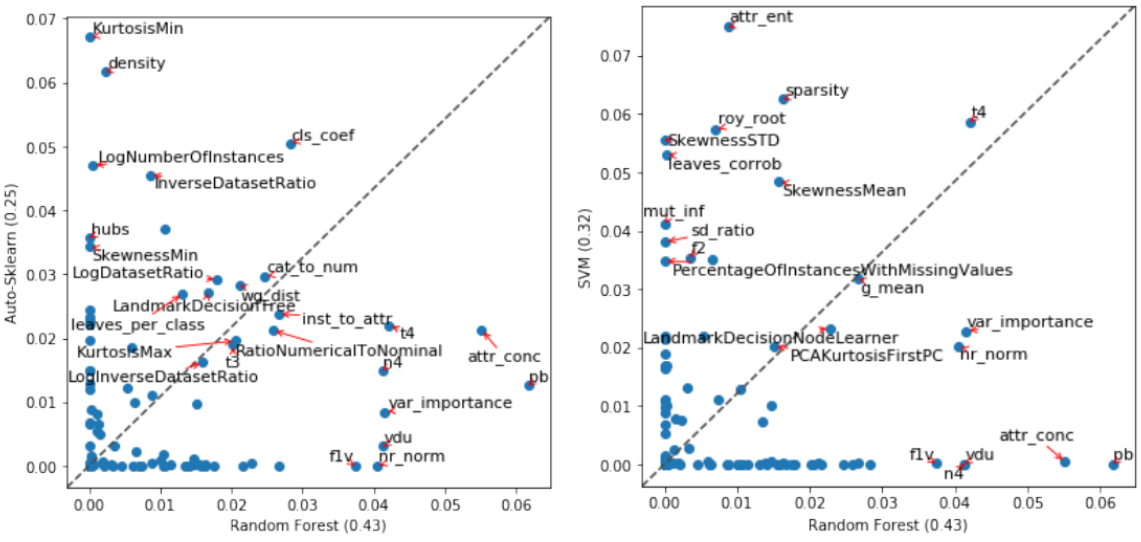

Figura 12: Importancia comparativa de las meta-features para Random Forest vs. AutoSkLearn (izquierda) y SVM (derecha). Las meta-features específicas de AutoSkLearn se indican porque su nombre comienza con una letra mayúscula.

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.