



**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  
Arianna Camila Palancar Ochando

Carrera: Ciencia de la Computación

## 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

<b>1. Introducción</b>	<b>2</b>
<b>2. MetaFeatX</b>	<b>5</b>
2.1. AutoML y Meta-Features . . . . .	5
2.2. Transporte Óptimo . . . . .	6
2.3. Principio de MetaFeatX y Argumentación del Benchmark . . . . .	7
2.4. Algoritmo MetaFeatX . . . . .	8
2.5. Extracción de Vecinos y Ranking de Pipelines . . . . .	10
<b>3. Transfer-Learning de Hiperparámetros</b>	<b>11</b>
<b>4. Experimentación</b>	<b>11</b>
4.1. Configuración experimental . . . . .	11
4.2. Tareas experimentales . . . . .	12
4.2.1. Tarea 1: Captura de la topología objetivo . . . . .	13
4.2.2. Tarea 2: AutoML sin modelo de rendimiento . . . . .	13
4.2.3. Tarea 3: AutoML con modelo de rendimiento . . . . .	14
4.2.4. Algoritmos, hiperparámetros y protocolo . . . . .	15
4.2.5. Resultados y análisis . . . . .	16
4.2.6. Sensibilidad e interpretabilidad . . . . .	16
4.3. Experimentación adicional y análisis propio . . . . .	17
<b>5. Discusión</b>	<b>17</b>
<b>6. Conclusiones</b>	<b>17</b>

# 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 euclíadiana 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  $(\Omega_x, d_x)$  y  $(\Omega_y, d_y)$  espacios métricos compactos con distribuciones  $x$  y  $y$ . El conjunto de distribuciones con márgenes  $x$  y  $y$  se denota  $\Gamma(x, y)$ , y  $c : \Omega_x \times \Omega_y \rightarrow \mathbb{R}^+$  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]:

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

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]:

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

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

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

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  $\psi : \mathbb{R}^D \rightarrow \mathbb{R}^d$  que transforma la representación básica al espacio proyectado, de manera que la distancia euclídea 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 euclídea que refleja la proximidad de datasets en términos de desempeño de hiperparámetros, facilitando tareas de AutoML como inicialización de optimización o transferencia de conocimiento.

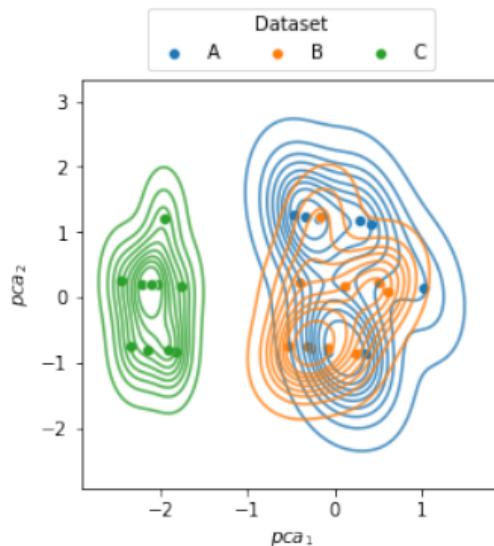


Figura 1: Configuraciones principales de los datasets A, B y C, donde B (en naranja) (respecto a C, en verde) es el vecino más cercano de A con respecto a la representación objetivo.

## 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.

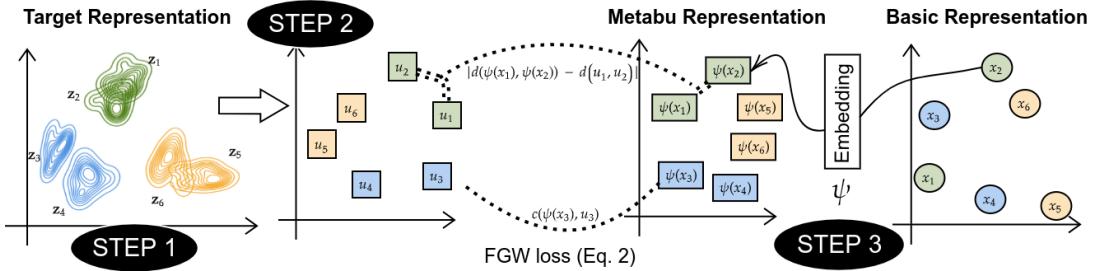


Figura 2: De las meta-features básicas a las MetaFeatX usando Fused Gromov-Wasserstein. Representaciones básicas (círculos), MetaFeatX (cuadrados) y representaciones objetivo (subgráfico izquierdo). Datasets vecinos en el espacio objetivo mantienen el mismo color en todos los subgráficos.

**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  $\Theta_i$ . La distancia entre datasets se mide mediante la “distancia 1-Wasserstein”:

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

donde  $c$  es el costo de transporte Euclíadiano 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 euclíadiana entre las proyecciones  $u_i$  y  $u_j$  aproxime la distancia 1-Wasserstein:

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

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:

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

y sobre las proyecciones objetivo:

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

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

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

donde  $\lambda$  es un parámetro de regularización y  $\|\psi\|$  es la norma de la función  $\psi$ . Se considera  $\psi$  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  $\gamma$  con el “algoritmo de Sinkhorn” [Cuturi, 2013].
- **Problema externo:** optimizar  $\psi$  considerando  $\gamma$  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

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

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

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

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 euclíadiana 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:

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

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:

$$p_1 \succ p_2 \succ \dots \succ p_m.$$

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

### 4. Experimentación

Esta sección presenta la evaluación experimental de los enfoques estudiados en este trabajo, organizada de manera progresiva y coherente con los objetivos del proyecto. Dado que los fundamentos teóricos y algorítmicos de los modelos han sido descritos en secciones anteriores, el énfasis aquí se pone exclusivamente en el análisis empírico de su comportamiento y en la validación de su utilidad práctica para AutoML.

En primer lugar, se reproducen y analizan los experimentos asociados al modelo **MetaFeatX**, siguiendo de cerca la metodología y los criterios de evaluación propuestos en el trabajo original que inspira este proyecto. El objetivo de esta etapa es verificar que las meta-features aprendidas capturan adecuadamente la estructura del espacio de datasets y que inducen una noción de similitud alineada con el rendimiento de los algoritmos y sus configuraciones de hiperparámetros.

A continuación, se presentan los **experimentos propios del equipo**, diseñados para estudiar el comportamiento de MetaFeatX en escenarios adicionales y para analizar empíricamente propiedades que no se exploran en profundidad en el trabajo original. Estos experimentos permiten evaluar la robustez del método, su sensibilidad a distintos parámetros y su impacto en tareas prácticas de muestreo e inicialización en AutoML.

Finalmente, se evalúa el modelo de **transfer learning para la predicción de hiperparámetros**, que utiliza las representaciones aprendidas por MetaFeatX como mecanismo de transferencia de conocimiento entre datasets. En esta etapa se analiza cómo la información estructural capturada por las meta-features puede ser explotada para inicializar de manera eficiente la búsqueda de configuraciones prometedoras, reduciendo el costo computacional asociado a la fase de *cold-start* en AutoML.

Esta organización permite separar claramente la validación del enfoque inspirado en la literatura, la contribución experimental propia del equipo y la evaluación del modelo de transferencia desarrollado, ofreciendo una visión completa y estructurada de los resultados obtenidos.

#### 4.1. Configuración experimental

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.

Dado que este número de conjuntos de datos es relativamente reducido para el aprendizaje de un mapeo entre meta-features básicas y representaciones objetivo, existe un riesgo significativo de **sobreajuste** y de una mala generalización del modelo aprendido. Para mitigar este problema, se amplía la colección de datasets mediante un procedimiento de *bootstrap*: para cada conjunto de datos original se generan 1000 versiones remuestreadas con reemplazo, recalculando las meta-características básicas pero heredando la misma representación objetivo.

Este procedimiento permite **densificar el espacio de meta-features** y estabilizar el aprendizaje del mapeo inducido por el Transporte Óptimo, sin introducir información artificial sobre el rendimiento de las configuraciones. De esta forma, se preserva la estructura topológica del espacio objetivo mientras se reduce la varianza del modelo y se mejora su capacidad de generalización a nuevos conjuntos de datos.

**Hiperparámetros del modelo** Para estos experimentos, el modelo fue configurado con los siguientes valores:

- $\alpha = 0,5$
- $\lambda_{\text{reg}} = 1 \times 10^{-3}$
- $\text{learning\_rate} = 1 \times 10^{-2}$
- $\text{seed} = 42$

## 4.2. Tareas experimentales

La experimentación en este trabajo se organiza en torno a tres objetivos principales: Validar si las meta-características aprendidas por MetaFeatX 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 MetaFeatX 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.

**Configuración de las tareas:** Sobre la base de la configuraciones experimentales anteriormente descrita, 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).

#### 4.2.1. 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 MetaFeatX coincide con su vecindario 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 MetaFeatX 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.

Dado que MetaFeatX aprende las meta-características mediante un procedimiento estocástico, cada experimento se repite varias veces variando *únicamente* la semilla aleatoria para capturar la variabilidad inherente al proceso de entrenamiento. Por el contrario, las meta-características de referencia, al ser deterministas, se evalúan una sola vez. Los resultados reportados en la Figura 3 representan el promedio de NDCG@k sobre todas las repeticiones y todos los conjuntos de datos de prueba, reflejando así tanto la calidad como la estabilidad de MetaFeatX frente a la aleatoriedad del aprendizaje.

#### 4.2.2. Tarea 2: AutoML sin modelo de rendimiento

En la segunda tarea se evalúa la capacidad de MetaFeatX para guiar el muestreo inicial de configuraciones de hiperparámetros sin recurrir a un modelo de rendimiento explícito, utilizando únicamente la información de vecindario. Para cada conjunto de datos de prueba y cada conjunto de meta-características (*MetaFeatX*, 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:

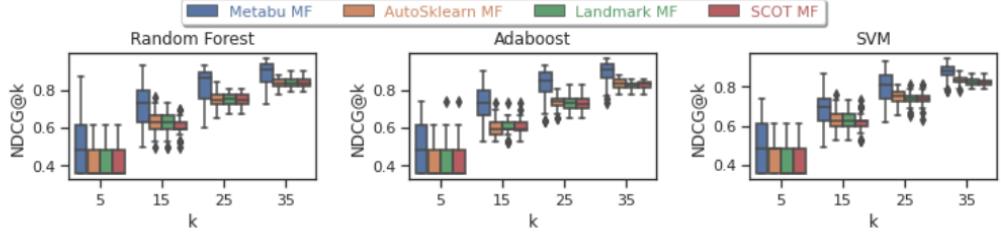


Figura 3: Captura de la topología objetivo. Similaridad NDCG@k entre el vecindario inducido por las meta-características y el vecindario real basado en distribuciones de configuraciones de alto rendimiento.

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

donde  $z^\ell$  es la representación objetivo del  $\ell$ -ésimo vecino más cercano según la distancia euclídea en el espacio de meta-características  $mf$ , y  $Z$  es una constante de normalización. Esta distribución se utiliza para “muestrear iterativamente configuraciones de hiperparámetros”, entrenar los modelos y registrar el rendimiento de la mejor configuración encontrada hasta la iteración  $t$ .

Para cada iteración  $t$ , se calcula el rango  $r(t, mf)$  de cada conjunto de meta-características, que indica la posición relativa de dicho método frente a los demás (MetaFeatX, AutoSklearn, Landmark, SCOT) y frente a un muestreador uniforme de referencia (*Random1x*). Este rango se asigna observando cuál método ha encontrado la mejor configuración de hiperparámetros hasta la iteración  $t$ : el método con mejor rendimiento recibe rango 1, el siguiente rango 2, y así sucesivamente hasta 5.

Un rango más bajo implica que la métrica es más efectiva para identificar regiones prometedoras del espacio de hiperparámetros sin entrenar previamente el modelo en el dataset de prueba. La inclusión del muestreador uniforme permite verificar que los métodos basados en meta-características aportan información útil más allá de la simple aleatoriedad.

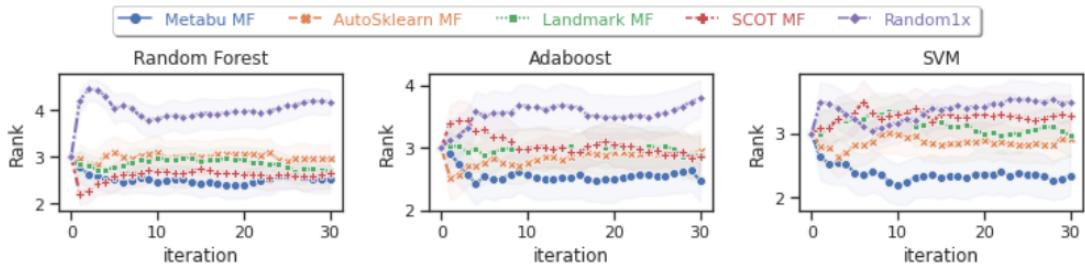


Figura 4: Task 2: muestreo de la distribución de configuraciones de hiperparámetros sin modelo de rendimiento. Las curvas muestran el rango promedio  $r(t, mf)$  sobre todos los conjuntos de datos de prueba.

#### 4.2.3. Tarea 3: AutoML con modelo de rendimiento

La tercera tarea evalúa el papel de las meta-características como mecanismo de inicialización de sistemas de AutoML basados en modelos de rendimiento, específicamente

AutoSklearn y PMF. En estos sistemas, la búsqueda de configuraciones es “adaptativa”: se seleccionan iterativamente configuraciones de hiperparámetros, se observa su rendimiento y se actualiza un modelo probabilístico que guía las decisiones de muestreo posteriores.

En esta tarea, las meta-características se emplean para seleccionar un conjunto inicial de configuraciones prometedoras. Para AutoSklearn, se consideran las mejores configuraciones de los diez vecinos más cercanos del dataset de prueba según la distancia en el espacio de meta-características, y se evalúan en el conjunto de datos objetivo para alimentar el modelo de rendimiento inicial, sobre el cual se ejecuta la optimización bayesiana. En PMF, se realiza un procedimiento análogo utilizando cinco vecinos, cuyos mejores resultados se usan para llenar parcialmente la fila correspondiente en la matriz de rendimientos y así inicializar su representación latente.

A partir de esta fase de inicialización, el sistema continúa su búsqueda estándar. El rendimiento se resume mediante “curvas de rango” en función del número de iteraciones, comparando las versiones originales de AutoSklearn y PMF con sus variantes híbridas MetaFeatX+AutoSklearn y MetaFeatX+PMF. También se incluyen muestreadores uniformes reforzados (Random2 $\times$  y Random4 $\times$ ), que seleccionan el mejor de varios candidatos aleatorios por iteración, como referencia de exploración aleatoria.

Esta tarea refleja la diferencia entre “Exploitation y Exploration”: mientras que en Task 2 se buscaba identificar rápidamente configuraciones prometedoras (Exploitation), en Task 3 se trata de inicializar la búsqueda en una “región diversa y de calidad” del espacio de hiperparámetros, de modo que la optimización adaptativa pueda descubrir configuraciones aún mejores a lo largo del tiempo.

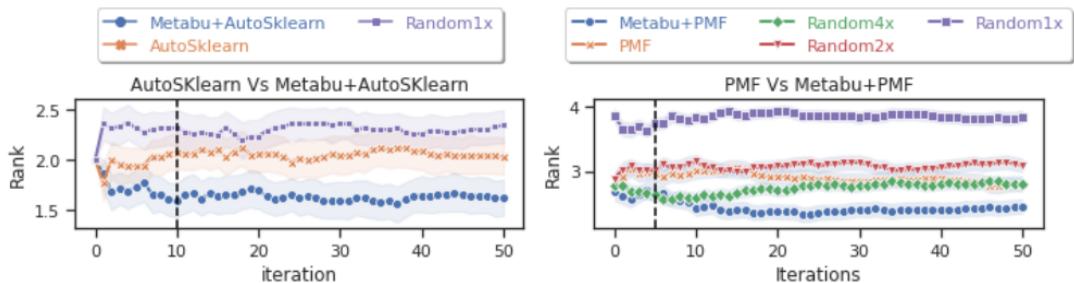


Figura 5: AutoML con modelo de rendimiento. Curvas de rango  $r(t, mf)$  que comparan las versiones originales de AutoSklearn y PMF con sus variantes híbridas inicializadas con MetaFeatX y con muestreadores uniformes. Un rango menor indica mejor desempeño relativo.

#### 4.2.4. 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  $\gamma$  en el caso de SVM, y un conjunto amplio de opciones de preprocessamiento 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 MetaFeatX 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.

#### 4.2.5. Resultados y análisis

En la Tarea 1, las curvas de NDCG@ $k$  muestran que la métrica inducida por MetaFeatX 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 MetaFeatX, 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 MetaFeatX 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 MetaFeatX se vuelve significativamente mejor a partir de ese punto; para SVM, MetaFeatX 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 MetaFeatX 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 MetaFeatX+AutoSklearn y MetaFeatX+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, MetaFeatX+AutoSklearn sobrepasa al pipeline AutoSklearn puro a partir de un número moderado de iteraciones, y MetaFeatX+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 MetaFeatX 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.

#### 4.2.6. Sensibilidad e interpretabilidad

El análisis de sensibilidad se centra en los dos hiperparámetros propios de MetaFeatX:  $\alpha$ , que pondera las componentes Wasserstein y Gromov-Wasserstein en la distancia FGW, y  $\lambda$ , que controla el peso de la regularización L1 sobre la transformación lineal  $\psi$ . La

sensibilidad se evalúa sobre Task 1, inspeccionando la diferencia

$$\text{NDCG@10}(\text{MetaFeatX}) - \text{NDCG@10}(\text{AutoSklearn})$$

para  $\alpha \in \{0,1,0,3,0,5,0,7,0,99\}$  y  $\lambda \in \{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}\}$ . Los resultados, mostrados en la Figura 6, indican que la diferencia es positiva en todo el dominio considerado y estadísticamente significativa según un test t de Student con  $p < 0,05$ .

Se observa además que MetaFeatX presenta baja sensibilidad frente a  $\lambda$  siempre que esta sea suficientemente pequeña ( $\lambda \leq 10^{-3}$ ). En este rango, también se aprecia baja sensibilidad respecto a  $\alpha$  en el intervalo  $[0,3,0,7]$ . Esto confirma la importancia de considerar ambas distancias: descartar la componente Wasserstein ( $\alpha \leq 0,1$ ) o la Gromov-Wasserstein ( $\alpha \geq 0,99$ ) degrada significativamente el rendimiento, mientras que los valores intermedios producen resultados estables y robustos.

En cuanto a la dimensión intrínseca del espacio de conjuntos de datos, estimada a partir de la distribución de distancias de Wasserstein de orden uno entre representaciones objetivo, se obtienen valores aproximados de 6 para AutoSklearn, 8 para AdaBoost, 9 para RandomForest y 14 para SVM. Un análisis adicional de estabilidad muestra cómo esta dimensión crece al incorporar más conjuntos de datos, ofreciendo 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 distintos conjuntos de datos, mayor es la dimensión intrínseca asociada.

Finalmente, la interpretabilidad de las meta-características se analiza proyectando las representaciones aprendidas mediante análisis de componentes principales (PCA) 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. Por ejemplo, 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 mayor peso en SVM que en RandomForest.

### 4.3. Experimentación adicional y análisis propio

## 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](#))

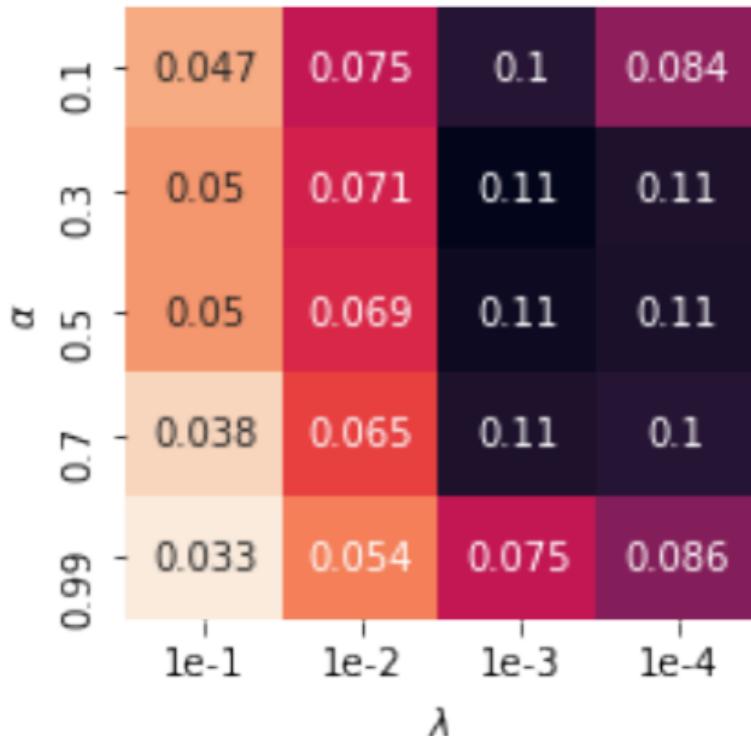


Figura 6: Sensibilidad de MetaFeatX frente a los hiperparámetros  $\alpha$  y  $\lambda$ , medida como la diferencia NDCG@10 respecto a AutoSklearn. Valores más oscuros indican mejor rendimiento.

- 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.

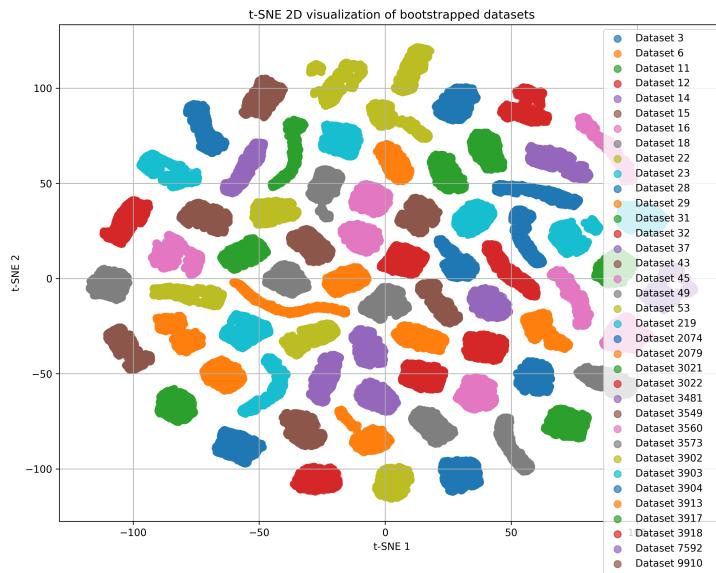


Figura 7: Visualización 2D mediante t-SNE de los conjuntos de datos de OpenML en el espacio de representación básica ,junto con sus ampliaciones generadas por bootstrapping. Solo se muestran algunos nombres de datasets para facilitar la legibilidad..

## 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  $\psi^*$

- 1: Construir la representación proyectada de los objetivos:

$C_{i,j} \leftarrow d_W^2(z_i, z_j)$ ,  $i, j = 1 \dots n$  (Distancia de Wasserstein par a par)  $\triangleright$  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  $C$ .  $\triangleright$  Esto ayuda a reducir la dimensionalidad manteniendo la estructura de los datos.
  - 3:  $u \leftarrow \text{MDS}(C, d)$   $\triangleright$  Multidimensional Scaling: proyecta los datos a un espacio de dimensión  $d$  preservando las distancias.
  - 4:  $\psi \leftarrow \text{Linear}(135, d)$   $\triangleright$  Capa lineal que mapea las 135 meta-características básicas a un espacio de dimensión  $d$ .
  - 5:  $x \leftarrow \frac{1}{n} \sum_{i=1}^n x_i$   $\triangleright$  Se obtiene un vector promedio de las representaciones básicas para normalizar el aprendizaje.
  - 6:  $L \leftarrow \text{FGW}$   $\triangleright$  FGW (Fused Gromov-Wasserstein) mide la discrepancia entre representaciones básicas y proyectadas.
  - 7:  $\psi^* \leftarrow \text{ADAM}(L, \psi \# \mathbf{x}, \mathbf{u})$   $\triangleright$  Se ajusta  $\psi$  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  $\Theta_i$ ,  $i = 1 \dots n$ . Dataset de prueba  $x$ .

**Resultado:** Distribución de configuraciones prometedoras  $z$ .

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

$z = \frac{1}{Z} \sum_{\ell=1}^{10} \exp(-\ell) \sum_{\theta \in \Theta_\ell} \theta$   $\triangleright$  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 muestran 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 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 n_estimators algorithm max_depth learning_rate	mean, median, most frequent [50, 500] SAMME, SAMME.R [1, 10] [0.01 , 2.0]
Random Forest (RF)	imputation criterion max_features min_samples_split min_samples_leaf bootstrap	mean, median, most frequent gini, entropy (0, 1] [2, 20] [1, 20] True, False
SVM	imputation C kernel degree gamma coef0 shrinking tol max_iter	mean, median, most frequent [0.03125, 32768] rbf, poly, sigmoid [1, 5] $[3,0517578125 \times 10^{-5}, 8]$ [-1, 1] True, False $[10^{-5}, 10^{-1}]$ -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.

# Listado 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_coeff	Coefficiente 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/extraclasé 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 dimensiones 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	Corroboration 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.
PCA KurtosisFirstPC	Curtosis de ejemplos en el primer componente principal.
PCA FracOfCompFor95Per	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.
NumberOffinstancesWithMissingValues	Número de instancias con valores faltantes.
NumberOffeatures	Número total de atributos.
NumberOfClasses	Número de clases.
NumberOffinstances	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.
PercentageOffinstancesWithMissingValues	Porcentaje de instancias con valores faltantes.
LogNumberOfFeatures	Logaritmo del número de atributos.
LogNumberOfInstances	Logaritmo del número de instancias.

## La estabilidad de la dimensión intrínseca

Intrinsic Dimension: mean (std)

	0.1	0.25	0.5	0.75	1
adaboost	5.70 (4.22)	7.04 (2.76)	7.53 (0.82)	8.29 (1.53)	8.07 (0.00)
random_forest	5.44 (2.91)	6.90 (2.51)	7.15 (1.51)	8.33 (0.81)	9.79 (0.00)
autosklearn	8.02 (13.38)	5.81 (2.17)	6.47 (1.22)	6.43 (0.83)	6.81 (0.00)
libsvm_svc	10.84 (7.75)	12.69 (3.95)	15.29 (2.72)	17.95 (2.78)	18.63 (0.00)

Figura 8: 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 8 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 muestran de manera suficientemente adecuada (las regiones buenas del) espacio de configuraciones.

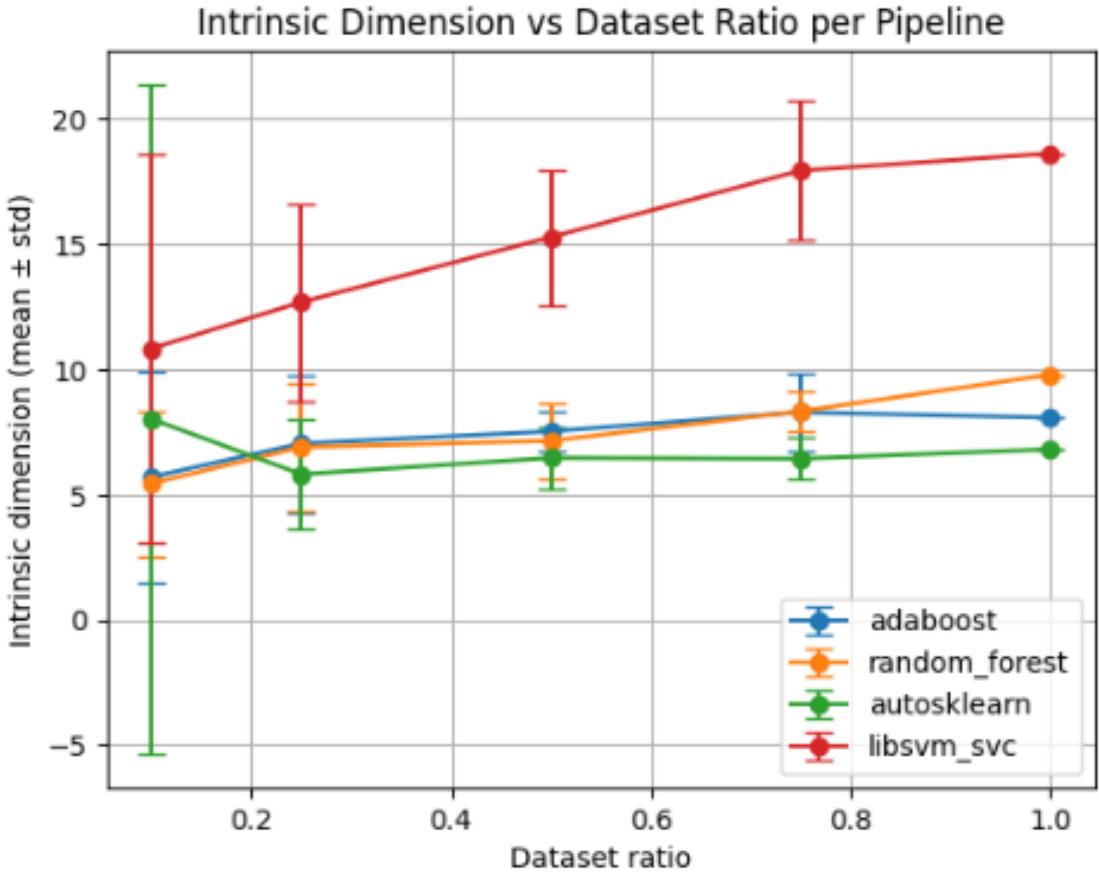


Figura 9: Dimensión intrínseca del espacio de datasets con respecto a los algoritmos de ML Adaboost, RandomForest, SVM y AutoSkLearn, en función de la fracción de datasets considerados en OpenML.

## 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  $\psi$ , 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  $\psi$  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 10 (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.

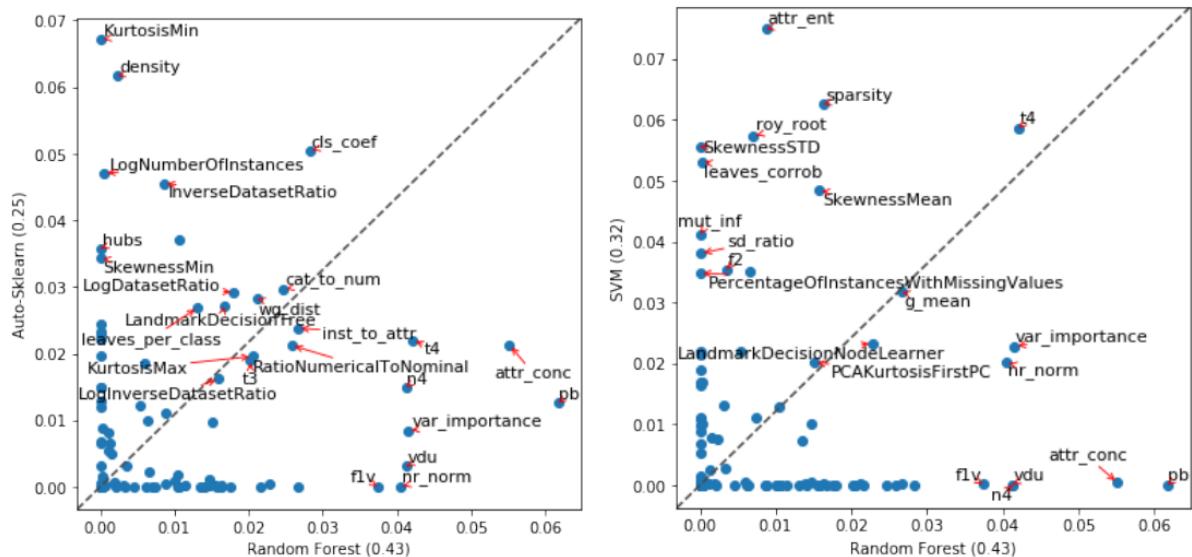


Figura 10: Importancia comparativa de las meta-features para Random Forest vs. AutoSkLearn (izquierda) y SVM (derecha). Las meta-features específicas de AutoSkLearn se indican porque su nombre comienza con una letra mayúscula.

# 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 imperdierunt 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 consectetur 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, consectetur 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 imperdierunt 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 consectetur tortor sapien facilisis magna. Mauris quis magna varius nulla scelerisque imperdierunt. 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, Michèle 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 Rasoaianaivo, 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.