Laboratorio 10: Git Wars Superheroes

Duración: 2 horas Formato: Competencia por Pull Requests

Objetivo: Que los estudiantes integren teoría y práctica sobre el consumo de APIs, limpieza de datos, modelos de regresión y la base teórica de la **Optimización Bayesiana**, mediante una dinámica tipo **Git Wars**.

Estructura detallada de la sesión

Tiempo	Actividad
0:00–0:10	Introducción a la dinámica y breve teoría de Optimización Bayesiana.
0:10–0:15	**Kahoot 1**
0:15–0:45	**Elemento 0 — Consumo de API y generación del dataset base.** **(en paralelo)** **Elemento 1 — Orquestador simple de evaluación de modelos.**
0:45–0:50	**Kahoot 2**
0:50–1:15	**Elemento 2 — Implementación de Optimización Bayesiana (BO).**
1:15–1:20	**Kahoot 3**
1:20–1:35	**Elemento 3 — Análisis comparativo de resultados.**
1:35–1:40	**Kahoot 4**
1:40–1:55	**Elemento 4 — API + Deploy.**
1:55–2:00	**Kahoot 5**

Requisitos técnicos

- **Stack sugerido:** `python 3.10+`, `numpy`, `pandas`, `scikit-learn`, `FastAPI` o `Flask`, `uvicorn`.
- **Entorno de contenedores:** `Docker` instalado (Docker Desktop, Docker Engine o equivalente).
- **Automatización:** uso de `Makefile` para construir, ejecutar y detener el contenedor.
- **Repositorio local / GitHub:** estructura clara del proyecto:

```
/gitwars-superheroes/
 data/
    data.csv                    # Dataset generado en la Elemento 0

 src/
    orchestrator.py             # Elemento 1: evaluadores SVM/RF/MLP
    optimizer.py                # Elemento 2: algoritmo BO (GP + UCB)
    random_search.py            # Comparativo BO vs. Random Search
    utils.py                    # Funciones auxiliares (carga y métricas)

 api/
    main.py                     # Elemento 4: API mínima de inferencia
    requirements.txt            # Dependencias del servicio web

 deployments/
    Dockerfile                  # Imagen final del contenedor
    render.yaml                 # Config obligatoria para Render

 notebooks/
    nb_<equipo>.ipynb           # Notebook con toda la práctica

 README.md                       # Instrucciones generales del laboratorio
 Makefile                        # Comandos make (build/run/test)
```

- **Entregable final para evaluación:** un archivo `.tar.gz` (tarball) que contenga todo el repositorio:

  `equipo_<nombre>.tar.gz`

Dinámica *Git Wars*

1. Se entregará a cada equipo un archivo `.tar.gz`.
2. Cada equipo hace **fork** del repositorio base indicado dentro del material incluido y crea una **rama por Elemento** (p. ej., `Elemento0/equipo`).
3. Se liberará un README por cada Elemento.
4. Tras implementar los cambios de cada Elemento, el equipo abre un **Pull Request** (PR) hacia el repositorio principal.
5. El PR se **revisa y mergea** únicamente si:
   (i) pasa los criterios de aceptación,
   (ii) el código es claro y ejecuta sin errores,
6. El **orden de aceptación (merge)** se refleja en el ranking en vivo del laboratorio.
7. Al finalizar, cada equipo deberá sincronizar su fork mediante `git pull` desde el `main` del repositorio base, obteniendo así la versión consolidada.

Elemento 0 — Consumo de API y generación del dataset

Objetivo: Consumir la **SuperHero API**, extraer y limpiar las variables necesarias, y generar un archivo `data/data.csv` adecuado para las Elementos posteriores.

Descripción general:

Cada equipo deberá consumir la API:

`https://akabab.github.io/superhero-api`

A partir de la estructura JSON devuelta, deberán extraer únicamente:

- **powerstats:** `intelligence`, `strength`, `speed`, `durability`, `combat`
- **appearance:** `height` (convertido a cm), `weight` (convertido a kg)
- **power** (variable objetivo)

Notas importantes:

- `gender` y `race` deben excluirse del dataset final.
- Las columnas de `height` y `weight` pueden venir en distintos formatos; deben normalizarse correctamente.
- Los procesamientos deberán considerarse para el momento en donde se pidan predicciones a su API monetada.

Procesamiento requerido (mínimo):

- Convertir alturas a centímetros → `height_cm`.
- Convertir pesos a kilogramos → `weight_kg`.
- Eliminar registros con valores faltantes en cualquier columna seleccionada.
- Asegurar que todas las columnas resultantes sean numéricas.

Script base recomendado (`src/Elemento0/get_data.py`):

```
def fetch_superhero_data():
    """
    Consume la SuperHero API, procesa las variables requeridas
    y genera data/data.csv con el dataset final.
    """
```

Cada equipo debe completar esta función respetando la estructura del repositorio.

Criterios de aceptación del PR (mínimos):

- El script corre sin errores con:

```
python3 src/Elemento0/get_data.py
```

- Se genera el archivo:

`data/data.csv`

- El archivo contiene exclusivamente las columnas:

  `intelligence, strength, speed, durability, combat, height_cm, weight_kg, power`

- No hay valores faltantes.
- Todas las columnas son numéricas.
- El archivo de data debe contener exactamente 600 registros.

Entrega por equipo:

- Archivo `src/Elemento0/get_data.py` funcional.
- Archivo `data/data.csv`.
- Pull Request limpio y bien documentado.

Elemento 1 — Orquestador simple de evaluación de modelos

Objetivo: Completar las funciones base del archivo `orchestrator.py` para evaluar tres modelos de ML (SVM, Random Forest y MLP) mediante un conjunto fijo de hiperparámetros. Cada función deberá:
(i) entrenar el modelo con los valores proporcionados,
(ii) medir su desempeño sobre los datos preparados en la Elemento 0 y
(iii) retornar un *float* con la métrica seleccionada (accuracy o F1 macro).

En el repositorio base (dentro del archivo `src/orchestrator.py`) se proporciona el siguiente esqueleto:

```
# TODO: implementar SVM
def evaluate_svm(C, gamma):
    """
    Debe entrenar un SVM con los hiperparámetros dados
    y devolver un float (accuracy o F1 macro).
    """
    pass

# TODO: implementar Random Forest
def evaluate_rf(n_estimators, max_depth):
    """
    Entrenar un Random Forest y retornar un float
    con la métrica de desempeño.
    """
    pass
```

```
# TODO: implementar MLP
def evaluate_mlp(hidden_layer_sizes, alpha):
    """
    Entrenar un MLPClassifier y devolver la métrica final.
    """
    pass
```

Criterios de aceptación (mínimos):

- Cada función debe ejecutar sin errores y devolver un **float** realista en el rango $[0, 1]$.
- El entrenamiento debe realizarse sólo con las columnas del dataset preparado en la Elemento 0.
- Las funciones no deben imprimir nada ni generar figuras; únicamente retornar la métrica.
- El archivo `orchestrator.py` debe ser importable sin fallos y las funciones deben ejecutarse con parámetros válidos.

Validación automática para PR:

- El pipeline ejecutará:

  ```
  from orchestrator import evaluate_svm, evaluate_rf, evaluate_mlp
  ```

  y verificará que cada función retorne un **float** sin lanzar excepciones.
- Si alguna función falla, el PR no será mergeado.

Notas operativas:

- Se recomienda usar `scikit-learn` para entrenar los tres modelos.
- La métrica sugerida es `mean_squared_error` (MSE), `root mean squared error` (RMSE) o `r2_score` (R^2), siendo RMSE la opción preferida por su interpretabilidad.
- Mantener los nombres de funciones y argumentos exactamente como aparecen en el esqueleto.

Elemento 2 — Implementación manual de Optimización Bayesiana (BO)

Objetivo: Implementar desde cero una versión simplificada de la **Optimización Bayesiana** para seleccionar hiperparámetros de los modelos definidos en la Elemento 1. La implementación deberá construirse sin librerías externas de BO, utilizando un **Gaussian Process (GP) simplificado** como modelo surrogate y una función de adquisición del tipo **Upper Confidence Bound (UCB)**.

Descripción general: Cada equipo completará el archivo `src/optimizer.py` implementando:

1. Un kernel RBF simplificado.
2. Ajuste del GP mediante la solución del sistema lineal $(K + \sigma_n^2 I)^{-1} y$.
3. Predicción del GP: media y varianza.
4. Función de adquisición UCB: $\text{UCB}(x) = \mu(x) + \kappa\sigma(x)$.
5. Búsqueda en una rejilla discreta sobre el espacio de hiperparámetros.
6. Un ciclo iterativo de BO que seleccione el siguiente punto evaluando la adquisición.

Se recomienda un esqueleto base como el siguiente:

```
from orchestrator import evaluate_svm, evaluate_rf, evaluate_mlp

# ------------------------------------------------------------
# Kernel RBF
# ------------------------------------------------------------
def rbf_kernel(x1, x2, length_scale=1.0):
    """
    TODO: implementar kernel RBF
    """
    pass

# ------------------------------------------------------------
# Ajuste del GP
# ------------------------------------------------------------
def fit_gp(X, y, length_scale=1.0, noise=1e-6):
    """
    TODO:
    - Construir matriz K usando el kernel RBF
    - Resolver (K + noise*I)^(-1) y
    - Regresar los parámetros necesarios para predecir
    """
    pass

# ------------------------------------------------------------
# Predicción del GP
# ------------------------------------------------------------
def gp_predict(X_train, y_train, X_test, length_scale=1.0, noise=1e-6):
    """
    TODO:
    - Calcular media mu(x*) y varianza sigma^2(x*)
    - Para cada punto en X_test
    """
    pass

# ------------------------------------------------------------
# Función de adquisición UCB
# ------------------------------------------------------------
def acquisition_ucb(mu, sigma, kappa=2.0):
    """
```

```
    TODO: UCB = mu + kappa * sigma
    """
    pass

# ------------------------------------------------------------
# BO principal
# ------------------------------------------------------------
def optimize_model(model_name, n_init=3, n_iter=10):
    """
    TODO:
    - Elegir dominios discretos para cada modelo
    - Samplear n_init puntos al azar
    - Evaluar el modelo usando orchestrator.py
    - Ajustar el GP con los puntos observados
    - Iterar:
        * Construir rejilla de hiperparámetros
        * Predecir mu, sigma
        * Evaluar UCB
        * Seleccionar el X que maximiza UCB
        * Evaluar el modelo real en ese X
        * Agregar punto a la historia
    - Regresar el mejor hiperparámetro encontrado
      y la mejor métrica (float)
    """
    pass
```

Fundamentos teóricos para la implementación manual de BO

La Optimización Bayesiana (BO) es un marco que permite optimizar funciones cuyo costo de evaluación es alto, ruidoso o no se conoce de forma explícita. En este laboratorio, BO se utilizará para seleccionar hiperparámetros óptimos de modelos de aprendizaje supervisado (SVM, Random Forest y MLP). Con el fin de que los estudiantes implementen el método desde cero, a continuación se presenta una guía teórica detallada para construir el modelo surrogate y la función de adquisición.

1. Modelo surrogate basado en Gaussian Processes (GP)

En el contexto de BO, el **modelo surrogate** aproxima la función real que se desea optimizar. Aquí, dicha función corresponde al desempeño de un modelo (accuracy o F1 macro), medido en función de sus hiperparámetros.

Un **Gaussian Process (GP)** define una distribución sobre funciones. Intuitivamente, permite predecir no solo un valor esperado en un punto, sino también la **incertidumbre** asociada a dicha predicción.

1.1 Elección del kernel

Se utilizará el **kernel RBF (Radial Basis Function)** debido a que:

- Es suave y universalmente aproximador.
- Permite construir un GP estable sin optimizar hiperparámetros complejos.
- Su forma cerrada es sencilla y computacionalmente eficiente.

La expresión del kernel RBF entre dos vectores de hiperparámetros x_1 y x_2 es:

$$k(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\,\ell^2}\right)$$

donde ℓ es la **longitud de escala**. En este laboratorio se fija $\ell = 1.0$ para evitar un ajuste adicional.

1.2 Construcción de la matriz K

Sea $X = \{x_1, x_2, \ldots, x_n\}$ el conjunto de hiperparámetros ya evaluados y $y = \{y_1, \ldots, y_n\}$ los desempeños asociados.

La matriz de covarianza se define como:

$$K_{ij} = k(x_i, x_j).$$

Al construir la matriz de covarianza del Gaussian Process, cada punto x_i representa una configuración de hiperparámetros que ya ha sido evaluada durante el proceso de optimización. Si se han evaluado n configuraciones diferentes, entonces se cuenta con los puntos:

$$X = \{x_1, x_2, \ldots, x_n\}.$$

Los índices i y j recorren este conjunto, de modo que cada entrada de la matriz de covarianza se define como:

$$K_{ij} = k(x_i, x_j),$$

donde $k(\cdot, \cdot)$ es el kernel RBF. Esto implica que:

- La **fila** i corresponde al punto x_i comparado con todos los demás puntos evaluados.
- La **columna** j corresponde a las similitudes entre el punto x_j y todos los demás.

Así, la matriz K es una matriz cuadrada de tamaño $n \times n$ que cuantifica la similitud entre todas las configuraciones de hiperparámetros observadas. Este objeto es fundamental para el ajuste del GP, pues determina la estructura de dependencia entre los valores ya evaluados y la forma en que el surrogate generaliza hacia nuevos puntos mediante sus predicciones de media y varianza.

Para garantizar estabilidad numérica, se utiliza un término de ruido σ_n^2:

$$K_{\text{ruidosa}} = K + \sigma_n^2 I,$$

donde típicamente $\sigma_n^2 = 10^{-6}$.

1.3 Ajuste del GP

A diferencia de una implementación completa de GPs, aquí no se optimizan hiperparámetros ni se maximiza la verosimilitud. La aproximación consiste en resolver el sistema:

$$(K + \sigma_n^2 I)\alpha = y,$$

donde α captura la información necesaria para predecir valores futuros.
El vector α se obtiene mediante:

$$\alpha = (K + \sigma_n^2 I)^{-1} y.$$

1.4 Predicción para un nuevo punto

Para un punto nuevo $x^\star$, se define el vector:

$$k(x^\star) = [k(x^\star, x_1), \ldots, k(x^\star, x_n)]^\top.$$

La **media predictiva** del GP es:

$$\mu(x^\star) = k(x^\star)^\top \alpha.$$

La **varianza predictiva** (incertidumbre) es:

$$\sigma^2(x^\star) = k(x^\star, x^\star) - k(x^\star)^\top (K + \sigma_n^2 I)^{-1} k(x^\star).$$

Esta varianza indica qué tan "segura" es la predicción del surrogate: valores grandes indican regiones poco exploradas del espacio de hiperparámetros.

2. Función de adquisición: Upper Confidence Bound (UCB)

La función de adquisición determina qué punto explorar a continuación. En este laboratorio se utilizará **Upper Confidence Bound (UCB)**, que combina explotación (alta media) con exploración (alta incertidumbre).

2.1 Definición

$$\mathrm{UCB}(x) = \mu(x) + \kappa\, \sigma(x)$$

donde:

- $\mu(x)$ = predicción media del GP,
- $\sigma(x)$ = incertidumbre del GP,
- κ = parámetro que balancea **exploración** y **explotación**.

Se recomienda usar $\kappa = 2.0$.

2.2 Interpretación

- Si $\mu(x)$ es alto $\rightarrow$ el modelo cree que el punto tendrá buen desempeño.
- Si $\sigma(x)$ es alto $\rightarrow$ el modelo está poco seguro (zona poco explorada).
- UCB combina ambas cualidades.

El siguiente punto a evaluar es:

$$x_{\text{next}} = \arg\max_x \text{UCB}(x).$$

donde x representa un punto del **espacio de hiperparámetros** definido para el modelo; es decir, una combinación candidata de valores (por ejemplo, C, γ, $n\_estimators$, max_depth, etc.) cuya utilidad se evalúa mediante la función de adquisición.

3. Dominios discretos de búsqueda

Para permitir una implementación eficiente y consistente entre equipos, los hiperparámetros de cada modelo deberán elegirse de un **dominio discreto**.

3.1 SVM

- $C \in \{0.1, 1, 10, 100\}$
- $\gamma \in \{0.001, 0.01, 0.1, 1\}$

3.2 Random Forest

- `n_estimators` $\in \{10, 20, 50, 100\}$
- `max_depth` $\in \{2, 4, 6, 8\}$

3.3 MLP

- `hidden_layer_sizes` $\in \{(16,),\ (32,),\ (64,),\ (32, 16)\}$
- `alpha` $\in \{10^{-4},\ 10^{-3},\ 10^{-2}\}$

4. Ciclo iterativo de BO

El proceso completo que deberán implementar los equipos es:

1. Seleccionar aleatoriamente n_{init} puntos del dominio.
2. Evaluar el modelo (SVM, RF o MLP) en cada punto.
3. Ajustar el GP con los pares (X, y) observados.
4. Construir una rejilla con todos los puntos del dominio.
5. Para cada punto, calcular:
   - media $\mu(x)$,
   - varianza $\sigma(x)$,
   - adquisición $\text{UCB}(x)$.
6. Seleccionar el punto con mayor valor de UCB.

7. Evaluarlo con el modelo real.
8. Agregar el nuevo par (x, y) al conjunto histórico.
9. Repetir hasta completar las iteraciones establecidas.

Este ciclo permite explorar sistemáticamente el espacio de hiperparámetros, balanceando regiones prometedoras con zonas poco exploradas.

Nota: En este ciclo, cada x representa un punto del *espacio de hiperparámetros* del modelo (por ejemplo, una combinación (C, γ) para SVM, o $(n\_estimators, max\_depth)$ para Random Forest). No se refiere a los registros del dataset, sino a las configuraciones que la BO evalúa.

Criterios de aceptación (mínimos):

- `optimize_model(model_name)` debe ejecutar sin errores.
- Debe regresar una tupla: `(best_params, best_metric)` donde `best_metric` es un **float**.
- Debe haberse implementado:
  - Kernel RBF
  - Ajuste del GP (inversión o resolución del sistema)
  - Predicción del GP
  - Función UCB
  - Búsqueda en rejilla
  - Ciclo iterativo de BO
- La implementación no debe usar Optuna, skopt, bayes_opt u otras bibliotecas de BO.
- La función debe explorar y explotar (i.e., usar UCB correctamente).

Notas operativas:

- La implementación puede trabajar con listas de Python o con `numpy`.
- Para resolver el sistema lineal se sugiere `np.linalg.solve`.
- Se recomienda mantener dimensiones pequeñas del dominio para que las iteraciones sean rápidas.

Elemento 3 — Análisis comparativo de resultados

Objetivo: Analizar y comparar el desempeño obtenido por la Optimización Bayesiana (BO) frente a una estrategia de *Random Search* para cada modelo (SVR, Random Forest Regressor y MLPRegressor), integrando métricas, tablas y conclusiones.

Descripción general:
Cada equipo deberá utilizar los resultados producidos en la Elemento 2 para elaborar un análisis comparativo que incluya:

- La mejor combinación de hiperparámetros encontrada por BO para cada modelo.
- La mejor combinación encontrada por Random Search.
- La métrica resultante (MSE, RMSE o R^2) para ambas estrategias.

- Una visualización que muestre el comportamiento de BO a lo largo de las iteraciones (por ejemplo: mejor valor observado por iteración).

Este análisis se reportará en el PR correspondiente y quedará registrado en el cuaderno final del equipo.

Entregables requeridos

1. **Tabla comparativa** (por modelo) con los siguientes elementos:
   - hiperparámetros óptimos por BO,
   - hiperparámetros óptimos por Random Search,
   - métrica alcanzada por cada método (MSE, RMSE o R^2).
2. **Gráfica de evolución de BO**: Cualquiera de las siguientes visualizaciones es válida:
   - mejor valor de la métrica por iteración,
   - evolución del surrogate (media predictiva),
   - trayectoria del punto seleccionado por UCB a través del dominio.
3. **Comentario interpretativo** respondiendo:
   - ¿Por qué BO converge más rápido que Random Search?
   - ¿Cómo influye la función de adquisición en el proceso?
   - ¿Qué modelo mostró menor error y por qué pudo ser más fácil de optimizar?

Criterios de aceptación del PR

- La tabla comparativa debe ser clara, legible y contener únicamente hiperparámetros válidos.
- La gráfica debe generarse a partir de los valores obtenidos en la Elemento 2 (no se permiten ejemplos artificiales).
- El análisis debe tener una interpretación técnica coherente, basada en los resultados del propio equipo.
- El PR debe incluir referencias explícitas a las funciones implementadas en la Elemento 2.

Notas operativas

- Las métricas deben calcularse sobre el mismo split utilizado en la Elemento 1 (o un subconjunto consistente del dataset).
- No se permite reutilizar valores, figuras o tablas de otros equipos.
- Se recomienda incluir capturas del resultado de BO para facilitar la revisión del PR.
- La calidad del análisis será evaluada junto con la claridad del código y las visualizaciones.
- Se permite utilizar únicamente las siguientes bibliotecas:
  - `numpy` (operaciones numéricas, manejo de vectores y matrices),

  - `pandas` (carga y manipulación del dataset),
  - `scikit-learn` (modelos de regresión: SVR, RandomForestRegressor, MLPRegressor; métricas: MSE, RMSE, R^2),
  - `matplotlib` o `seaborn` (visualizaciones para la Elemento 3).
- No se permite utilizar ninguna biblioteca de optimización bayesiana (`optuna`, `bayes_opt`, `scikit-optimize`, etc.). La implementación debe ser completamente manual según lo descrito en la Elemento 2.

Elemento 4 — API + Contenedor Docker + Makefile

Objetivo: Exponer el mejor modelo encontrado mediante Optimización Bayesiana (SVM, RF o MLP) a través de una **API REST** implementada con FastAPI o Flask, empaquetar el servicio dentro de un **contenedor Docker** y automatizar su ejecución mediante un `Makefile`. El objetivo es que el docente pueda evaluar cualquier entrega usando únicamente unos cuantos comandos reproducibles.

1. Requisitos mínimos de la API

La API debe ubicarse en:

`api/main.py`

y debe exponer, al menos, los siguientes endpoints:

- **/health** — Verificación rápida:

```
{"status": "ok"}
```

- **/info** — Información del equipo y del modelo:
  - nombre del equipo,
  - tipo de modelo usado (SVM, RF o MLP),
  - hiperparámetros óptimos encontrados,
  - breve descripción del preprocesamiento aplicado.
- **/predict** — Endpoint principal de inferencia. Debe recibir un JSON con las características crudas:

```
{
    "features": {
        "intelligence": ...,
        "strength": ...,
        "speed": ...,
        "durability": ...,
        "combat": ...,
        "height_cm": ...,
        "weight_kg": ...
```

```
    }
}
```

La API deberá:

- aplicar el **mismo preprocesamiento** usado en entrenamiento,
- cargar el modelo desde `model/model.pkl`,
- devolver un número real con la predicción del `power`.

Notas importantes:

- La API debe encargarse del preprocesamiento interno, no el cliente.
- No se permiten dependencias no declaradas en `requirements.txt`.
- El modelo debe cargarse una única vez al iniciar la app.

2. Contenedor Docker

El repositorio debe incluir un archivo:

`deployments/Dockerfile`

El Dockerfile debe:

1. copiar todo el proyecto al contenedor,
2. instalar dependencias desde `api/requirements.txt`,
3. exponer el puerto `8000`,
4. ejecutar la API con:

```
uvicorn api.main:app --host 0.0.0.0 --port 8000
```

Se recomienda basarse en la guía establecida en prácticas previas **(cf. Lab 9) turn3file0**.

3. Makefile obligatorio

El repositorio debe contener un archivo `Makefile` con las reglas mínimas:

- **make build** — Construye la imagen Docker.
- **make run** — Levanta el contenedor en segundo plano.
- **make status** — Muestra el estado del contenedor.
- **make stop** — Detiene y elimina el contenedor.
- **make clean** — Limpia recursos de Docker.
- **make package** — Genera:

```
equipo_<nombre>.tar.gz
```

Cada comando debe ejecutarse sin errores en un entorno limpio.

4. Flujo de evaluación del docente

Para asegurar reproducibilidad, el docente ejecutará exactamente este flujo:

1. Descomprimir el tarball:

   ```
   tar -xzvf equipo_<nombre>.tar.gz
   ```

2. Entrar al proyecto:

   ```
   cd gitwars-superheroes/
   ```

3. Construir el contenedor:

   ```
   make build
   ```

4. Ejecutarlo:

   ```
   make run
   ```

5. Probar los endpoints:

   ```
   curl http://localhost:8000/health
   curl http://localhost:8000/info
   curl -X POST http://localhost:8000/predict \
   -H "Content-Type: application/json" \
   -d '{"features": {"intelligence": 50, "strength": 80,
                     "speed": 60, "durability": 70,
                     "combat": 55, "height_cm": 185,
                     "weight_kg": 90}}'
   ```

6. Detener el contenedor:

   ```
   make stop
   ```

Si cualquiera de estos pasos falla, el PR no será aceptado.

5. Subtareas prácticas

Cada equipo debe entregar:

- API funcional en `api/main.py`.
- Archivo `model/model.pkl` (modelo óptimo entrenado).
- Preprocesador serializado (si aplica).
- Dockerfile funcional.
- Makefile funcional.
- `equipo_<nombre>.tar.gz` reproducible.

6. Criterios de aceptación del PR

- La API responde correctamente en `/health`, `/info`, `/predict`.
- El contenedor construye sin errores usando `make build`.
- El servicio expone el puerto 8000 con `make run`.
- `make stop` y `make clean` funcionan correctamente.
- El modelo cargado es el mejor obtenido en la Elemento 2.
- El tarball se extrae y funciona en un entorno limpio.
- El README explica cómo levantar el servicio.

7. Validación del despliegue en Render

Además del funcionamiento local mediante Docker y Makefile, cada equipo deberá proporcionar un **URL público** generado por la plataforma Render. El docente verificará que el servicio esté correctamente desplegado y que los endpoints respondan de forma consistente con el comportamiento local.

7.1 Requisitos del despliegue en Render

El despliegue deberá realizarse mediante un archivo `render.yaml` incluido en el repositorio del equipo. Render detectará automáticamente este archivo y construirá el servicio sin necesidad de configuraciones manuales en la interfaz web.

- El archivo `render.yaml` debe definir un **Web Service** con:

```
runtime: docker
```

- Render debe construir la imagen utilizando **exclusivamente** el `Dockerfile` incluido en el repositorio.
- El modelo (`model.pkl`) y cualquier artefacto asociado (preprocesadores, normalizadores, etc.) deben estar presentes en el repositorio para que Render pueda cargarlos al iniciar el contenedor.
- El arranque del servicio debe ejecutar el comando definido en el Dockerfile (por ejemplo, `uvicorn api.main:app --host 0.0.0.0 --port 10000`).
- El servicio debe exponer al menos los siguientes endpoints públicos:

```
/health      -> chequeo de estado
/info        -> metadatos del modelo
/predict     -> predicción del atributo "power"
```

- El URL público generado por Render debe ser accesible desde cualquier navegador, con una estructura similar a:

```
https://<subdominio-del-equipo>.onrender.com/predict
```

- No se aceptan errores 404, 422 (bad request), 500 (internal server error), timeouts, ni comportamientos no deterministas.

7.2 Flujo de validación del docente en Render

El docente ejecutará las siguientes verificaciones sobre la URL pública del equipo:

1. **Verificar disponibilidad del servicio:**

   ```
   curl https://<subdominio>.onrender.com/health
   ```

   Debe responder:

   ```
   {"status": "ok"}
   ```

2. **Verificar información del modelo:**

   ```
   curl https://<subdominio>.onrender.com/info
   ```

   La respuesta debe incluir:

   - tipo de modelo (SVM/RF/MLP),
   - hiperparámetros óptimos,
   - resumen del preprocesamiento aplicado.

3. **Verificar función de predicción:**

   ```
   curl -X POST https://<subdominio>.onrender.com/predict \
   -H "Content-Type: application/json" \
   -d '{"features": {
           "intelligence": 60, "strength": 80, "speed": 55,
           "durability": 70, "combat": 65,
           "height": "6'8", "weight_kg": "980 lb"
       }}'
   ```

   La respuesta debe ser un JSON válido con un valor numérico:

   ```
   {"prediction": <valor_real>}
   ```

4. **Comparar coherencia:** La predicción en Render debe ser coherente (mismo orden de magnitud) con la predicción local generada en Docker. No se exige que sean idénticas, pero sí que:

   - el preprocesamiento sea el mismo,
   - no existan cambios inesperados,
   - el modelo cargado sea el mismo.

7.3 Criterios de aceptación del despliegue

Para aprobar la Elemento 4, el servicio en Render debe cumplir:

- Disponibilidad: respuesta válida a `/health`.
- Transparencia: respuesta correcta en `/info`.
- Funcionalidad: `/predict` debe responder sin errores.
- Consistencia: el modelo debe ser el mismo que el probado localmente.
- Reproducibilidad: la API debe funcionar igual después de redeploy.

Evaluación

La calificación final del laboratorio se dividirá en cuatro elementos técnicos. El Elemento 0 (dataset mediante API) y la bitácora Git son requisitos de entrega, pero **no aportan puntaje**.

Elemento	Ponderación
Elemento 1 — Orquestador de modelos (SVM, RF, MLP) Correcta implementación de `evaluate_svm`, `evaluate_rf` y `evaluate_mlp`; retorno de métricas válidas; código claro y modular.	20%
Elemento 2 — Optimización Bayesiana desde cero (GP + UCB) Construcción del modelo surrogate (GP), cálculo de media y varianza, implementación de UCB, ciclo iterativo completo y resultados coherentes.	40%
Elemento 3 — Análisis comparativo de resultados Tablas, gráficas de convergencia e interpretación clara BO vs. Random Search; calidad del análisis y claridad de las conclusiones.	20%
Elemento 4 — API + Docker + Render Contenedor funcional mediante `make`; servicio FastAPI/Flask operativo; endpoints `/health`, `/info`, `/predict`; despliegue correcto en Render usando `render.yaml`.	20%
Total	**100%**

Notas importantes:

- El **Elemento 0** (dataset mediante API) y la **bitácora Git** son necesarios para validar el trabajo, pero no otorgan puntos.
- Cada elemento se evalúa de manera independiente.

- La reproducibilidad es obligatoria: cualquier script, contenedor o servicio debe funcionar desde cero.
- Los PRs deben ser funcionales; código que no ejecuta no se considera válido.

Puntaje total de la competencia (Git Wars)

La competencia del laboratorio se compone de tres partes:

1. **Puntaje por PR ganador en cada Elemento** (progresión obligatoria).
2. **Puntaje por ranking de predicciones** (calidad del modelo vía RMSE).
3. **Puntaje adicional por documentación** (mejor README y mejor cuaderno).

Estos puntajes son independientes de la evaluación académica formal y funcionan como bonificaciones dentro de la dinámica Git Wars.

1. Puntaje por Elemento (PR ganador)

Para cada Elemento únicamente se otorgarán puntos al **primer Pull Request (PR)** aceptado que cumpla satisfactoriamente con todos los criterios técnicos. Los demás equipos obtendrán **0 puntos** en ese Elemento.

Elemento	**Puntos para el PR ganador**
Elemento 0 — Dataset generado desde API	20
Elemento 1 — Orquestador de modelos	30
Elemento 2 — Optimización Bayesiana (GP + UCB)	65
Elemento 3 — Análisis comparativo	35
Elemento 4 — API + Docker + Render	50

Reglas del sistema de PRs:

- Los equipos **deben completar los Elementos en orden**. No se puede competir por un Elemento sin haber entregado correctamente todos los anteriores.
- Si un equipo intenta enviar un PR de un Elemento sin haber completado los previos, **el PR será rechazado automáticamente**.
- Si otro equipo ya obtuvo el PR ganador de un Elemento, los demás equipos no podrán obtener puntos de competencia en ese Elemento, aun si su PR es correcto.
- Un PR solo se considera válido si:
  - ejecuta sin errores,
  - cumple todos los criterios técnicos,
  - está correctamente documentado,
  - se envió desde la rama adecuada.

2. Puntaje por ranking de predicciones (RMSE)

Además del puntaje por PR ganador, cada equipo podrá obtener un puntaje adicional según la **calidad final de su modelo**, medida mediante el RMSE promedio entre el servicio local (Docker) y el servicio desplegado en Render.

Procedimiento del docente:

1. Ejecutar 150 predicciones contra el contenedor local del equipo (levantado vía `make build` y `make run`) para obtener $\text{RMSE}_{\text{local}}$.
2. Ejecutar las mismas 150 predicciones contra su URL público en Render para obtener $\text{RMSE}_{\text{Render}}$.
3. Calcular el RMSE promedio:

$$\text{RMSE}_{\text{prom}} = \frac{\text{RMSE}_{\text{local}} + \text{RMSE}_{\text{Render}}}{2}.$$

4. Ordenar a los equipos del menor al mayor $\text{RMSE}_{\text{prom}}$.

Puntaje asignado según ranking:

Posición (menor RMSE promedio)	Puntos
1er lugar	100
2do lugar	70
3er lugar	45
4to–5to lugar	25
6to lugar en adelante	10

Notas importantes:

- Si el contenedor o el servicio en Render del equipo fallan, no respondan o arrojen error, el equipo recibirá **0 puntos** en esta sección.
- El modelo debe ser coherente: discrepancias extremas entre local y Render podrán ser penalizadas.
- Todos los equipos participan en esta sección, independientemente de si ganaron o no algún PR.

3. Puntaje adicional por documentación (README y cuaderno)

No se proporcionará un template fijo para la documentación. Cada equipo deberá proponer su propia estructura tanto para el **README** del repositorio como para el **cuaderno** de la práctica, con especial énfasis en explicar de forma clara y detallada la **Optimización Bayesiana** implementada y los resultados obtenidos.

Se otorgarán puntos adicionales de la siguiente forma:

- **Mejor README del laboratorio** (claridad, organización, explicación del flujo, instrucciones de ejecución, resumen de BO y del modelo): **150 puntos**.

- **Mejor cuaderno** `nb_<equipo>.ipynb` (narrativa técnica clara, visualizaciones, interpretación de BO, comparación vs. Random Search, explicaciones propias): **200 puntos**.

Criterios generales de selección:

- Claridad y orden de la explicación.
- Profundidad conceptual al describir la Optimización Bayesiana (GP, UCB, ciclo iterativo).
- Conexión entre teoría e implementación concreta del equipo.
- Calidad de las figuras y tablas usadas para justificar resultados.
- Facilidad con la que otra persona podría reproducir los experimentos.

Solo se premiará a un equipo por categoría (un README ganador y un cuaderno ganador); el resto de los equipos recibirá **0 puntos** en estos rubros.

En conjunto, el puntaje por PR ganador, el ranking de predicciones y la calidad de la documentación constituyen el sistema completo de competencia Git Wars para este laboratorio.