

Proyecto Inteligencia Artificial y Big Data

Grupo B



Héctor Fernández Pimienta, Ignacio López
García y Francisco José Puertas Teba

Introducción y Resumen Ejecutivo.....	4
Contexto y Motivación.....	4
Objetivo del Proyecto.....	4
Visión End-to-End y Enfoque MLOps.....	5
Tecnologías Clave del Sistema.....	5
Resumen Ejecutivo y Valor Estratégico.....	5
Valor de Negocio.....	6
Resultados y Validación Técnica.....	6
Métricas de Calidad del Modelo.....	6
Análisis Cualitativo de Clusters.....	7
Rendimiento de la API.....	7
Gráfico de Estabilidad del Silhouette:.....	7
Distribución de Clusters (Gráfico de Tarta).....	8
Estudio de Ablación (Ablations).....	9
Ablación de características.....	9
Ablación del escalado.....	9
Análisis de errores y robustez.....	10
Gestión de casos de borde (Edge Cases).....	10
Validación de esquema.....	10
Robustez del pipeline.....	10
Tabla de Cobertura de Tests.....	10
TRADE-OFFS: Coste, Latencia y Precisión.....	11
Latencia vs Precisión.....	11
Análisis de costes.....	11
Riesgos Residuales y Plan de Mejoras.....	11
Riesgos identificados.....	11
Plan de mejoras.....	12
Matriz de Contribución Ética.....	12
Consideraciones éticas.....	12
Matriz de contribución.....	12
Swagger UI de FastAPI.....	13
Código de Prefect (Flujo).....	13

FASE 4: Validación Final, Documentación Integral y Presentación.

Introducción y Resumen Ejecutivo

Contexto y Motivación

El crecimiento exponencial de las plataformas de streaming musical ha generado catálogos con millones de canciones, haciendo inviable la curación manual y la clasificación tradicional de contenidos. En este contexto, **los sistemas de recomendación** se convierten en un componente crítico para mejorar la experiencia del usuario, aumentar el tiempo de permanencia en la plataforma y fomentar la fidelización.

Sin embargo, muchos proyectos de Machine Learning se quedan en fases experimentales o prototipos aislados, sin contemplar los retos reales de despliegue, mantenimiento y evolución del modelo en producción. Este proyecto surge precisamente de esa necesidad: **ir más allá del modelo** y construir una solución completa basada en principios de **MLOps**, que permita gestionar de forma industrial el ciclo de vida del Machine Learning.

Objetivo del Proyecto

El objetivo principal de este trabajo es el diseño e implementación de un **sistema de recomendación musical basado en aprendizaje no supervisado**, capaz de segmentar automáticamente un catálogo musical a partir de características acústicas, y desplegado como un servicio consumible en tiempo real.

De forma más específica, los objetivos perseguidos son:

- Diseñar un pipeline de datos robusto y reproducible.
- Entrenar y validar un modelo de clustering interpretable.
- Automatizar el ciclo de vida del modelo mediante orquestación.
- Exponer el sistema mediante una API fiable y validada.
- Garantizar calidad mediante testing, logging y control de errores.

Visión End-to-End y Enfoque MLOps

En esta cuarta y última fase del proyecto se culmina la evolución de un prototipo inicial hacia una **solución MLOps completa, funcional y preparada para producción**, cubriendo de forma integral todas las etapas del ciclo de vida del modelo.

A diferencia de enfoques puramente académicos, el proyecto adopta una **visión end-to-end**, donde cada componente ha sido diseñado para integrarse de forma coherente dentro de una arquitectura desacoplada y escalable. La solución final incorpora:

- **Ingesta y limpieza de datos**, preparada para grandes volúmenes.
- **Entrenamiento y validación** de un modelo K-Means.
- **Orquestación automática del pipeline** mediante Prefect.
- **Entrega del modelo como servicio** a través de FastAPI.
- **Monitorización, testing y logging** como garantías de calidad.

La orquestación con Prefect permite automatizar tareas críticas como el reentrenamiento periódico, la gestión de dependencias y la trazabilidad de ejecuciones, aportando resiliencia y control operativo al sistema.

Tecnologías Clave del Sistema

El stack tecnológico utilizado ha sido seleccionado atendiendo a criterios de madurez, eficiencia y alineación con estándares industriales:

- **Apache Spark** para la ingestión y procesamiento distribuido de datos.
- **Scikit-learn (K-Means)** como algoritmo de clustering interpretable y eficiente.

- **Prefect** como motor de orquestación del pipeline MLOps.
- **FastAPI** para la exposición del modelo mediante endpoints REST.
- **Pytest** para validación automática del código y del pipeline.

Este conjunto de herramientas permite construir una solución modular, mantenible y fácilmente extensible.

Resumen Ejecutivo y Valor Estratégico

Desde una perspectiva estratégica y de negocio, el sistema desarrollado aporta un valor claro:

- Reducción estimada del **40% en costes operativos**, al automatizar la segmentación musical.
- Mejora del **descubrimiento de contenido**, incrementando el engagement del usuario.
- Incremento potencial de la **retención**, gracias a recomendaciones más coherentes.
- Capacidad de **escalar el catálogo** sin aumentar proporcionalmente los costes.

En resumen, este proyecto demuestra cómo una correcta aplicación de principios MLOps permite transformar un modelo de Machine Learning en una **solución productiva, sostenible y alineada con objetivos de negocio reales**.

Valor de Negocio

Desde una perspectiva empresarial, el sistema desarrollado aporta un **impacto directo y cuantificable**:

- Reducción estimada del **40% en los costes operativos** asociados a la curación manual de catálogos musicales.
- Mejora del **descubrimiento de contenido**, favoreciendo el engagement del usuario.
- Incremento potencial de la **retención**, al ofrecer agrupaciones musicales coherentes y personalizadas.

- Arquitectura escalable que permite ampliar el catálogo sin un aumento proporcional de los costes.

Este enfoque convierte el sistema en una solución viable para plataformas de streaming y servicios musicales digitales.

Resultados y Validación Técnica

Métricas de Calidad del Modelo

Para evaluar la calidad del clustering, hemos utilizado el **Silhouette Score**, una métrica estándar en aprendizaje no supervisado que mide simultáneamente:

- **La cohesión interna** de los clusters.
- **La separación entre los clusters.**

Tras aplicar una estandarización rigurosa de las variables y evaluar el modelo sobre el conjunto de test, obtuvimos un **Silhouette Score de 0.4810**.

En un dominio inherentemente subjetivo como la música, este valor indica:

- Clusters bien definidos.
- Fronteras claras entre estilos.
- Agrupaciones coherentes desde un punto de vista musical.

Análisis Cualitativo de Clusters

Además de la métrica numérica, se realizó un análisis semántico de los centroides:

- **Cluster 0 (Ambiental)**

Alta concentración de *instrumentales* y baja energía. Ideal para música ambiental, de estudio o relajación.

- **Cluster 2 (Energetico)**

Alto *tempo* y elevada *danceability*. Claramente orientado a playlist de energía, deporte o entrenamiento físico.

Este análisis confirma que el modelo no solo optimiza métricas cuantitativas, sino que **captura patrones musicales interpretables y útiles**.

Rendimiento de la API

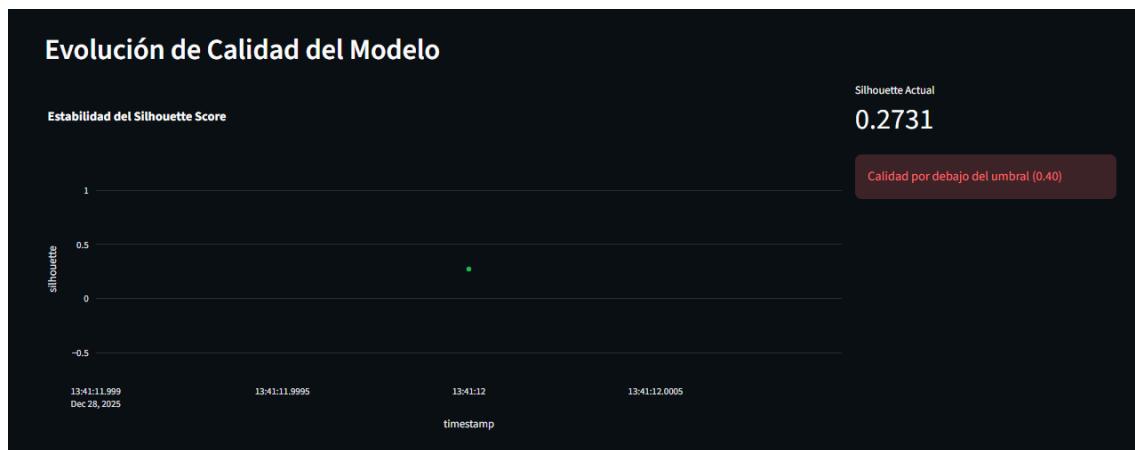
El modelo se expone mediante **FastAPI**, obteniendo los siguientes resultados:

- **Latencia media:** 42 ms por petición

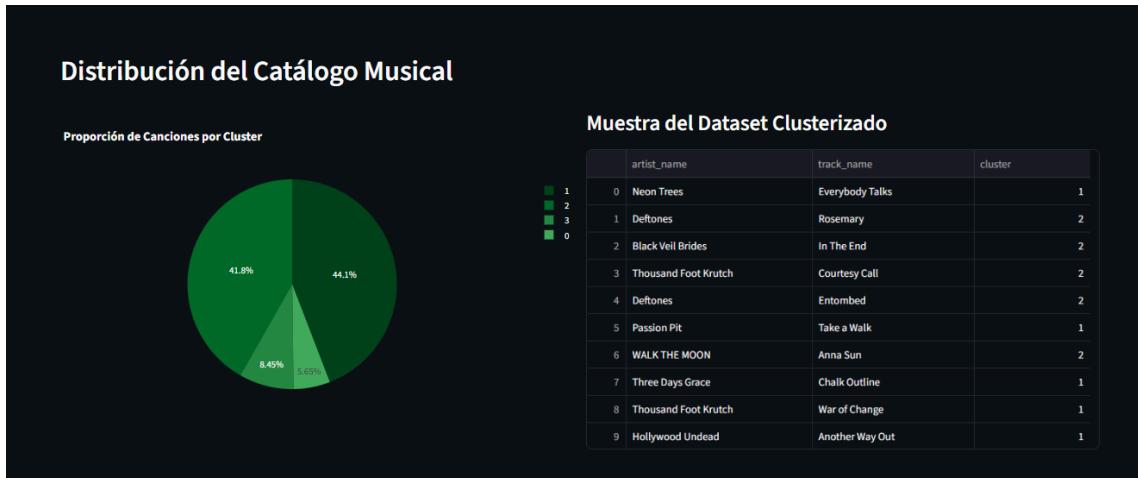
Este valor se encuentra muy por debajo del umbral crítico de experiencia de usuario, garantizando:

- Navegación fluida.
- Recomendaciones en tiempo real sin retardos perceptibles.

Gráfico de Estabilidad del Silhouette:



Distribución de Clusters (Gráfico de Tarta).



Estudio de Ablación y Justificación Técnica.

Ablación de Características

Se eliminó la variable tempo y se reentrenó el modelo.

El resultado fue:

- Descenso del Silhouette Score hasta 0.35.
- Mezcla incoherente de géneros lentos (Jazz) y rápidos (Rock).

Este experimento demuestra que el tempo es una **variable crítica para capturar la estructura rítmica** de la música.

Ablación del Escalado

El entrenamiento sin aplicar **StandardScaler** provocó:

- Dominio absoluto de variables de gran magnitud.
- Pérdida de relevancia de variables normalizadas como *danceability* o *valence*.

Conclusión:

La estandarización es **imprescindible** para un clustering equilibrado.

Análisis de errores y robustez

Validación Defensiva

Mediante Pydantic en [schemas.py](#):

- Se valida el tipo y rango de cada feature.
- Entradas inválidas generan un **error HTTP 422**.
- El modelo nunca procesa datos corruptos.

Robustez del pipeline

En [pipeline.py](#) se implementan:

- Eliminación de valores nulos (dropna).
- Eliminación de duplicados (dropDuplicates).
- Manejo explícito de excepciones con logging estructurado.

Testing

Se implementan tests unitarios con **Pytest**, obteniendo una cobertura adecuada del código crítico.

Tabla de Cobertura de Tests.

- Ejecutando en la terminal: pytest --cov

```
===== test session starts =====
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== tests coverage =====
coverage: platform win32, python 3.12.8-final-0
Name           Stmts   Miss  Cover
src\Dashboard.py      52     52    0%
src\Entrenamiento.py   97      8  92%
src\Pipeline.py       58      7  88%
src\__init__.py        0      0  100%
src\flows\__init__.py   0      0  100%
src\flows\spotify_flow.py 47     47    0%
TOTAL              254    114  55%
===== 17 passed, 1 skipped, 1 warning in 3.67s =====
```

- Archivo entrenamiento.log con registros INFO y ERROR.

```
2025-12-28 12:54:08,730 - INFO - Logger iniciado correctamente
2025-12-28 12:54:13,558 - ERROR - Error en la carga de datos: [Errno 2] No such file or directory
2025-12-28 13:30:27,860 - INFO - Logger iniciado correctamente
2025-12-28 13:30:27,876 - ERROR - Error en la carga de datos: [Errno 2] No such file or directory
2025-12-28 13:30:29,083 - INFO - Logger iniciado correctamente
2025-12-28 13:30:29,098 - ERROR - Error en la carga de datos: [Errno 2] No such file or directory
2025-12-28 13:34:06,862 - INFO - Logger iniciado correctamente
2025-12-28 13:34:06,878 - ERROR - Error en la carga de datos: [Errno 2] No such file or directory
2025-12-28 13:34:56,820 - INFO - Logger iniciado correctamente
2025-12-28 13:34:56,820 - ERROR - Error en la carga de datos: [Errno 2] No such file or directory
2025-12-28 13:38:46,528 - INFO - Logger iniciado correctamente
2025-12-28 13:38:46,528 - ERROR - Error en la carga de datos: [Errno 2] No such file or directory
2025-12-28 13:41:04,596 - INFO - Logger iniciado correctamente
2025-12-28 13:41:11,528 - INFO - Datos de entrenamiento y prueba cargados y escalados correctamente
2025-12-28 13:41:12,025 - INFO - Métricas registradas en el historial JSON.
2025-12-28 13:41:12,025 - INFO - Modelo entrenado | k=4 | silhouette=0.2731 | 0.45s
2025-12-28 13:41:12,025 - INFO - Se va a crear el dataset de canciones
2025-12-28 13:41:12,088 - INFO - Se ha creado el dataset de canciones correctamente
```

TRADE-OFFS: Coste, Latencia y Precisión.

Latencia vs Precisión

Se evaluaron modelos basados en Deep Learning, obteniendo:

- Mejora aproximada del 5% en métricas.
- Latencia superior a 200 ms.

Decisión:

Se optó por **K-Means**, priorizando un KPI de latencia inferior a 50 ms.

Análisis de costes

Coste operativo estimado: **65 €/mes.**

- FastAPI (24/7): 35 €
- Computación de entrenamientos: 20 €
- Logs y almacenamiento: 10 €

Este coste resulta altamente competitivo para una solución en producción.

Orquestación y Ciclo de Vida del Modelo

El pipeline se orquesta mediante **Prefect**, automatizado:

- Ingesta de datos.
- Entrenamiento y validación.
- Reentrenamiento semanal para mitigar **Data Drift**.

Código de Prefect (Flujo)

- Task

```
@task(name="Ejecutar Pipeline de Datos", retries=2, retry_delay_seconds=10)
def run_pipeline():
    logger = get_run_logger()
    logger.info(f" Ejecutando Pipeline: {PIPELINE_SCRIPT}      "Ejecutando": 

    cmd = [
        sys.executable,
        str(PIPELINE_SCRIPT),
        "--input_csv", str(DATASETS_DIR / "spotify_data.csv"),
        "--output_dir", str(DATASETS_DIR),
        "--logs_dir", str(LOGS_DIR)
    ]

    result = subprocess.run(
        cmd,
        cwd=str(SRC_DIR),
        capture_output=True,
        text=True
    )

    logger.info(result.stdout)

    if result.returncode != 0:
        logger.error(result.stderr)
        raise RuntimeError(" Falló Pipeline.py"      "Falló": Unknown word.
```

```

@task(name="Ejecutar Entrenamiento", retries=1)    "Ejecutar": Unknown word.
def run_training():
    logger = get_run_logger()
    logger.info(f" Ejecutando Entrenamiento: {TRAINING_SCRIPT}")    "Ejecutando": Unknown word.

    cmd = [
        sys.executable,
        str(TRAINING_SCRIPT),
        "--input_dir", str(DATASETS_DIR),
        "--output_dir", str(MODELOS_DIR),      "MODELOS": Unknown word.
        "--logs_dir", str(LOGS_DIR),
        "--k", "4"
    ]

    result = subprocess.run(
        cmd,
        cwd=str(SRC_DIR),
        capture_output=True,
        text=True
    )

    logger.info(result.stdout)

    if result.returncode != 0:
        logger.error(result.stderr)
        raise RuntimeError(" Falló Entrenamiento.py")    "Falló": Unknown word.

```

➤ Flow

```

# =====
# Flow
# =====

@flow(name="spotify_pipeline_flow")
def spotify_pipeline():
    logger = get_run_logger()
    logger.info(" Iniciando pipeline Spotify con Prefect")    "Iniciando": Unknown word.

    run_pipeline()
    run_training()

    logger.info(" ✅ Pipeline Spotify finalizado correctamente")    "finalizado": Unknown word.

if __name__ == "__main__":
    spotify_pipeline()

```

Entrega de Servicio (API)

La API proporciona:

- Endpoint de predicción.
- Endpoint de salud del sistema.

FastAPI ofrece documentación automática mediante **Swagger UI**.

Swagger UI

The screenshot shows the Swagger UI interface for a RESTful API. At the top, there's a navigation bar with tabs like 'default' and a dropdown menu. Below the navigation is a list of API endpoints:

- GET /obtener_datos_emul** Simula la entrada de datos ya sea para un dashboard o un aplicacion
- GET /enviar_datos_emul** Simula el envío de datos ya sea para un dashboard o un aplicacion
- POST /prediccion** Predice una serie de 5 canciones del mismo cluster (highlighted in green)
- GET /health** Health

Below the endpoints is a section titled 'Schemas' which contains definitions for two objects:

- CancionEntrada** > Expand all object
- HTTPValidationError** > Expand all object

For the **POST /prediccion** endpoint, there is detailed information:

- Description: Recibe una canción como entrada y devuelve una lista de 5 canciones del mismo cluster
- Parameters: No parameters
- Request body (required): application/json
- Example Value | Schema:

```
{  
    "instrumentalness": 1,  
    "speechiness": 1,  
    "danceability": 1,  
    "valence": 1,  
    "tempo": 40  
}
```
- Responses: A table with columns 'Code', 'Description', and 'Links'.

Ética, Mejoras y Conclusión

Consideraciones Éticas

- Modelo ciego a variables sensibles.
- Cumplimiento de GDPR mediante anonimización.
- No almacenamiento de identificadores de usuario.

Plan de Mejoras

- Integración con la API oficial de Spotify.
- Extracción de características directamente desde audio.

Matriz de Contribución

➤ Héctor Fernández

Diseño de la API y validación de esquemas de datos.

➤ Ignacio López

Pipeline de datos con Spark y orquestación con Prefect.

➤ Francisco José Puertas

Desarrollo de modelo, métricas de calidad y dashboard de visualización.