Informe de Machine Learning

Integrantes

- Dilan Mauricio Lemos - 202359416
- Diego Fernando Lenis - 202359540
- Jaime Andrés Noreña - 202359523
- Juan José Restrepo A - 202359517

---

1. Informe de Clustering

1.1 Introducción y Objetivo del Proyecto

El presente informe detalla la aplicación de la metodología de C*lustering* **K-Means** sobre el dataset **Terminal-Bench**, un conjunto de datos diseñado para evaluar agentes de Inteligencia Artificial en entornos de terminal. El objetivo principal de este análisis no fue la segmentación de clientes, sino la **identificación de grupos homogéneos de tareas** basándose en características cuantificables como el tiempo de ejecución, el tamaño de los archivos asociados y el nivel de dificultad.

Se buscó transformar datos complejos, como los String de descripción y etiquetas, en un conjunto de variables numéricas y categóricas limpias. Esta preparación fue crucial para que el algoritmo K-Means pudiera medir distancias de manera efectiva y agrupar las tareas con perfiles de complejidad, recursos y propósito similares. El resultado permite a la empresa evaluar y gestionar sus recursos de cómputo de manera más estratégica.

1.2 Desarrollo del Análisis de Datos

1.2.1 Preprocesamiento y Selección de Características

El primer paso consistió en la preparación del dataset. Dado que el algoritmo K-Means es sensible a variables no numéricas y a las diferencias de escala, se realizaron las siguientes acciones:

• **Eliminación de Datos Innecesarios:** Se descartaron columnas de texto largo (`base_description`), listas (`tags`) e identificadores únicos (`task_id`, *hashes*), ya que no aportan valor métrico directo al Clustering y podrían sesgarlo negativamente.

• **Codificación de Variables Categóricas:**
  ◦ La variable `difficulty` (easy, medium, hard) fue convertida a un valor **Ordinal** (1, 2, 3) para preservar la relación de orden jerárquico.
  ◦ La variable `category` (e.g., *software-engineering*, *debugging*) fue transformada mediante **One-Hot Encoding** para que el algoritmo pudiera interpretarla sin inferir un orden inexistente.

• **Escalamiento de Datos:** La matriz final de características fue sometida a **Estandarización (StandardScaler)**. Este paso igualó la influencia de todas las métricas (ej. *timeouts* y `archive_bytes`), asegurando que ninguna dominara el cálculo de las distancias debido a su rango de valores inherentemente mayor.

1.2.2 Determinación del Número Óptimo de Cluster (K)

Para justificar la elección del parámetro K (número de grupos), se empleó el **Método del Codo**.
Al analizar la gráfica de la Inercia (suma de cuadrados dentro del *cluster*), se observa que la reducción de la varianza interna se estabiliza visiblemente a partir del valor **K=4** o K=5. Se seleccionó **K=4** como el número óptimo, ya que proporciona una segmentación clara y manejable sin añadir una complejidad innecesaria al modelo.

1.2.3 Modelado y Visualización

El modelo K-Means se entrenó utilizando la matriz de datos estandarizados con K=4. Para validar visualmente la separación lograda, se aplicó la técnica de **Análisis de Componentes Principales (PCA)**. La gráfica de PCA proyectó el espacio de 19 dimensiones a solo 2 Componentes Principales (CP1 y CP2).

• **Componente Principal 1 (Eje X):** Este eje captura la mayor parte de la varianza en los datos, estando fuertemente correlacionado con el **Factor de**

Tiempo Extremo de Ejecución. El punto aislado del *Cluster 2* hacia la derecha indica una diferencia masiva en sus métricas de *timeout* con respecto a todas las demás tareas.

• **Componente Principal 2 (Eje Y):** Este eje diferencia principalmente los grupos restantes en función de la **Dificultad y la Complejidad** general de la tarea. Los grupos superiores corresponden a tareas más complejas (*Cluster 0*) y los inferiores a tareas más comunes (*Cluster 1*).

1.3 Análisis y Caracterización de los Cluster

El análisis de las medias de las características por *cluster* (observado en el apartado del código donde se ejecuta esta acción) confirma la existencia de cuatro perfiles de tareas bien diferenciados.

1.3.1 Perfiles Identificados

Los cuatro *cluster* representan estrategias distintas para la gestión de recursos:

• **Cluster 2 (Tarea Extrema Crítica):**
  ◦ **Identificación:** Un único *outlier* (Count=1).
  ◦ **Características:** Timeouts Máximos Absolutos (86,400 seg.) y Dificultad Máxima (3.0). El tamaño de archivo es mínimo.
  ◦ **Conclusión:** Requiere una cola de ejecución aislada y manejo especial de recursos debido a su duración excepcional.

• **Cluster 3 (Tareas Pesadas de I/O):**
  ◦ **Identificación:** Grupo pequeño (Count=3), exclusivo de la categoría `model-training`.
  ◦ **Características:** `archive_bytes` extremadamente altos (11.5 MB en promedio), mientras que los timeouts son bajos.
  ◦ Conclusión: El cuello de botella es el movimiento de datos. Se deben optimizar los sistemas de almacenamiento y red para la carga rápida de archivos.

• **Cluster 0 (Tareas Hard y de Habilidad):**

- **Identificación:** Grupo grande (Count=41).
- **Características:** Dificultad media muy alta (2.95). Dominancia de `software-engineering` (29%) y otras categorías complejas (`security`, `scientific-computing`).
- **Conclusión:** Representa la prueba de fuego del agente. Estas tareas deben usarse para medir la robustez, el razonamiento y las capacidades avanzadas de resolución de problemas.

- **Cluster 1 (Tareas Light y Comunes):**
  - **Identificación:** El grupo más grande (Count=67).
  - **Características:** Dificultad media más baja (1.69), con alta presencia de `system-administration` y `debugging`.
  - **Conclusión:** Constituye la línea base del benchmark. Ideal para pruebas de regresión rápidas, estabilidad y verificación de funcionalidad básica del agente.

1.4 Propuestas Estratégicas y Conclusiones

La segmentación basada en clustering valida que la plataforma Terminal-Bench no evalúa tareas de forma uniforme. La principal conclusión para la empresa es la necesidad de un sistema de gestión de colas de evaluación diferenciado.

Cluster	Estrategia Propuesta
2 y 3	**Gestión de Outliers:** Aislamiento en colas separadas y asignación de recursos especializados, ya sea por tiempo (Cluster 2) o por almacenamiento/ancho de banda (Cluster 3).
0	**Énfasis en Habilidad:** Usar estas tareas para la **validación final de la madurez** y la *performance* del agente, ya que son las que consumen la mayor capacidad intelectual.
1	**Prioridad en Estabilidad:** Ejecutar estas tareas en procesos de **prueba continua y rápida** para verificar la estabilidad diaria del agente sin comprometer recursos excesivos.

En resumen, el Clustering K-Means en el dataset ha proporcionado un marco analítico sólido que transforma los datos brutos de la plataforma en cuatro categorías de valor estratégico, permitiendo una asignación de recursos más eficiente y una evaluación más precisa de los agentes de IA.

---

2. Informe Sistemas de recomendación

2.1. Introducción

El objetivo de este proyecto es desarrollar un **sistema de recomendación** utilizando **reglas de asociación** sobre el dataset **“Amazon Reviews 2023”**.

Este sistema permite recomendar productos a los usuarios basado en los ítems que ya han revisado o comprado, aprovechando patrones frecuentes en las transacciones de la base de datos.

El dataset original contiene millones de registros de usuarios, productos (`asin`) y sus valoraciones (`rating`). Para poder trabajar eficientemente en un notebook, se tomó **una muestra del 1% de la categoria “books”**.

2.2. Carga del dataset

El notebook inicia cargando el dataset **"cogsci13/Amazon-Reviews-2023-Books-Review”** tomando unicamene el **1%** para permitir eficiencia en la ejecución ya que el datset es bastante grande.

> **NOTA:** El dataset **"cogsci13/Amazon-Reviews-2023-Books-Review”** representa la información de la categoría **“books-review”** del data set original **“McAuley-Lab/Amazon-Reviews-2023”** ya que este datset no permite ser cargado en el notebook y además es demasiado pesado para ser cargado por completo.

2.3. Preparación de los datos

2.3.1. Seleccionar solo las columnas que serán necesarias `df[['user_id', 'asin']]` creando una canasta de compras de los usuarios:

```
transactions = df_small.groupby('user_id')['asin'].apply(list)
```

2.3.2. Se aplica **One-Hot encoding** para transformar la canasta de compras `transactions` en una matriz de valores **binarios** `df_onehot` que representa cuando se ha comprado cierto articulo y cuando no, donde cada filas representa un usuario y cada columna representa un producto.

2.4. Aplicación de algoritmo Apriori

En este caso se optó por usar el algoritmo **apriori** para identificar la frecuencia de conjuntos de productos con un soporte mínimo de **0.002** siendo este

soporte la proporción de usuarios que contienen cierto conjunto de productos.

2.5. Reglas de asociación

Se aplicó la función `association_rules` de `mlxtend` para generar reglas de la forma:

Antecedente → Consecuente

Cada regla tiene métricas de interés:

- **Support**: frecuencia relativa del conjunto de productos completo
- **Confidence**: probabilidad de que ocurra el consecuente dado el antecedente
- **Lift**: fuerza de la regla comparada con ocurrencias aleatorias

Las reglas se ordenaron por **confianza descendente** para facilitar la recomendación.

2.6. Función de recomendación personalizada según usuario

Se implementa una función `recomendar_por_usuario(user_id, n=5)` para generar las recomendaciones personalizadas a un usuario seleccionado de la siguiente manera:

- Obtiene los productos ya revisados/comprados por el usuario verificando que el usuario si tenga un historial de compra.

```
    productos_usuario = set(df[df['user_id'] == user_id]['asin'])

    if len(productos_usuario) == 0:
        return f" El usuario {user_id} no tiene historial suficiente."
```

- Filtra las reglas cuyo antecedente esté incluido en el historial del usuario asignándolo a una variable `candidatos`

```
    candidatos = rules[rules['antecedents'].apply(lambda x: x.issubset(p
roductos_usuario))]

    if len(candidatos) == 0:
        return f"No hay reglas asociadas a este usuario."
```

- Devuelve los productos conseguidos en los consecuentes que aún no ha comprado.

```
sugerencias = (
    candidatos.assign(
        recomendacion=candidatos['consequents'].apply(lambda x: next(i
ter(x)))
    )
    .query("recomendacion not in @productos_usuario")
    .sort_values("confidence", ascending=False)
    .head(n)
)
```

  Ordenado los resultados por **confianza** para priorizar recomendaciones más seguras.

2.7. Visualizaciones

- **Heatmap:** Se utilizó un mapa de calor para evidenciar el conjuntos de productos más repetido según su soporte.
- **Grafo:** Se usó un grafo para mostrar las recomendaciones de productos teniendo que:

  nodo = producto, arista = reglas frecuentes entre productos.

2.8. Conclusión

- Las **reglas de asociación** permiten generar recomendaciones personalizadas por usuario basadas en patrones reales de compra/review.
- La **muestra del 1%** es suficiente para probar y validar el sistema en un notebook.
- La combinación de Apriori y visualizaciones facilita:
  - Identificar productos más populares
  - Detectar relaciones fuertes entre productos
  - Generar recomendaciones confiables y explicables
- Este enfoque puede **escalarse** usando muestras más grandes o categorías específicas para producción

---

3. Informe de Arboles de Decisión

3.1. Introducción

El documento analiza el proceso de construcción, entrenamiento y visualización de un modelo de **árbol de decisión** utilizando el dataset *Titanic* de Kaggle. El objetivo principal es predecir la supervivencia de los pasajeros a partir de variables relevantes, explorando además el rol de los **hiperparámetros** en el modelo.

3.2. Carga y preparación de los datos

El notebook inicia importando librerías esenciales:

- **pandas** para la manipulación de datos
- **scikit-learn** para los modelos de machine learning
- **matplotlib** para la visualización
- funciones de **sklearn.tree** para entrenar y graficar el árbol

Luego se carga el archivo `train.csv` de Kaggle y se realiza una selección de columnas relevantes para el modelo. Aunque no se muestran todas en la salida previa, comúnmente se incluyen variables como:

- `Pclass`
- `Sex` (codificado)
- `Age`
- `SibSp`
- `Parch`
- `Fare`
- `Embarked`

Estas columnas representan características útiles para predecir la variable objetivo: **Survived**.

3.3. Entrenamiento del Árbol de Decisión

Una vez preparado el dataset, se entrena un **DecisionTreeClassifier**. El notebook utiliza un árbol "básico" (sin hiperparámetros modificados), lo cual sirve como punto de partida pero puede llevar a:

- Sobreajuste (overfitting)
- Árboles demasiado profundos
- Baja capacidad de generalización

Por esto, es crucial hablar de **hiperparámetros**.

3.4. Hiperparámetros del Árbol de Decisión

Los **hiperparámetros** son configuraciones externas al modelo que controlan su comportamiento, su complejidad y su capacidad de generalización. En un árbol de decisión, estos son especialmente importantes porque el modelo tiende naturalmente a aprender reglas muy específicas del conjunto de entrenamiento.

Los hiperparámetros más relevantes incluyen:

3.4.1 `max_depth`

- Controla la **profundidad máxima** del árbol.
- Si no se restringe, el árbol crece hasta clasificar perfectamente el entrenamiento.
- Un valor bajo → modelo simple con riesgo de *underfitting*.
- Un valor alto → modelo complejo con riesgo de *overfitting*.

3.4.2 `min_samples_split`

- Número mínimo de muestras para dividir un nodo.
- Sirve para impedir divisiones sobre particiones muy pequeñas.

3.4.3 `min_samples_leaf`

- Número mínimo de muestras requeridas en una hoja.
- Reduce la variabilidad del modelo.

3.4.4 `criterion`

Indica la métrica usada para medir la calidad de las particiones:

- `"gini"` (por defecto)

3.4.5 `max_leaf_nodes`

- Limita la cantidad de hojas.
- Útil para simplificar el árbol.

3.4.6 `max_features`

- Número de características consideradas al dividir.
- Común en Random Forest, pero aplicable también a árboles simples.

3.5. Visualización

El notebook utiliza `plot_tree` para visualizar el árbol resultante.

Esto permite:

- Observar las reglas aprendidas
- Ver cómo se dividen los nodos
- Identificar sobreajuste si el árbol luce excesivamente ramificado

Un árbol demasiado profundo suele indicar la necesidad de ajustar hiperparámetros.

3.6. Evaluación del Modelo

Despues de entrenar el modelo se hizo la prediccion de archivo test.csv y se subio a kaggle y se obtuvo una precision de 0.77272.

3.7. Importancia de Ajustar los Hiperparámetros

El ajuste de hiperparámetros permite encontrar un balance entre:

- **Sesgo** (modelo muy simple)
- **Varianza** (modelo demasiado complejo)

Esto se hace típicamente con:

- **GridSearchCV**
- **RandomizedSearchCV**
- Validación cruzada (cross-validation)

3.8. Conclusiones

- El notebook presenta un flujo correcto para entrenar un árbol de decisión.

- Los hiperparámetros son **cruciales** para evitar sobreajuste y mejorar el rendimiento real del modelo.
- La visualización del árbol ayuda a interpretar el comportamiento del modelo.
- Para un trabajo completo es necesario incluir evaluación y un proceso de ajuste sistemático de hiperparámetros.