📘 Sistema Recomendador Básico de Películas

Curso: Inteligencia Artificial

Materia: Portafolio

Autores: Herminson Lezcano Agudelo, Sandra Patricia Berrio Londoño, Kimberly Celeita, Henry Muñoz R., Leslie Moreno

Fecha: noviembre 04 de 2025

📁 Tabla de Contenido

1. 🌞 Introducción

Este proyecto tiene como propósito construir un sistema básico de recomendación de películas, aplicando técnicas de análisis de datos e inteligencia artificial. Se busca que el sistema sea funcional, visualmente atractivo y comprensible para cualquier usuario, integrando imágenes interactivas y una interfaz web sencilla.

2. 🌟 Objetivos

- Implementar un recomendador de películas basado en similitud de contenido.
- Crear una interfaz visual con Streamlit.
- Integrar imágenes tipo catálogo que desplieguen información al hacer clic.
- Documentar el proceso utilizando herramientas de IA y flujos visuales.

3. 💪 Herramientas utilizadas

Herramienta	Función principal
Python	Lenguaje base para el motor de recomendación
Pandas	Manipulación de datos
Streamlit	Interfaz web visual
KNIME	Flujo visual alternativo para limpieza y recomendación
ChatGPT / Copilot	Generación de código, Prompts y documentación
Gemini / Claude	Apoyo en redacción y revisión de contenido
Canvas	Presentación visual del proyecto
TMDB API	Obtención de imágenes de películas

4. 🔍 Metodología

4.1 📂 Preparación de datos

Se utilizó el data set MovieLens 100k. Se cargaron los archivos movies_buscador_IN_ESP.csv, ratings.csv, .devcontainer, requeriments.txt se calcularon promedios de puntuación y se unieron con los títulos.

4.2 🧠 Algoritmo de recomendación

```
===== DATOS DE CARGA =====

#df = pd.read_excel("movies_buscador_IN_ESP.xlsx") SE ELIMINA BUSQUEDA EN EXCEL #df = pd.read_csv("movies_buscador_IN_ESP.csv", sep=';', encoding='utf-8', engine='python') df = pd.read_csv("movies_buscador_IN_ESP.csv", sep=';', encoding='latin-1', engine='python') df = df.dropna(axis=1, how='all') # Elimina columnas vacías



st.title("🍿 CineMatch - Tu recomendador de películas") st.markdown("Busca películas por título (inglés o español) o explora por género 🎥")



Botón refrescar alineado a la derecha

col1, col2 = st.columns([5, 1]) con col2: si st.button("🔄 Refrescar búsqueda"): st.rerun()



Buscador

title_search = st.text_input("🔎 Buscar por título (en inglés o español):")



Filtro por género

todos_los_géneros = pd.unique(df[['géneros','géneros.1','géneros.2','géneros.3','géneros.4']].values.ravel('K'))
```

```
todos_los_géneros = [g for g in todos_los_géneros if pd.notna(g)] géneros_seleccionados = st.multiselect("🎭 Filtrar por género:", sorted(todos_los_géneros))



===== FILTROS =====

df_filtrado = df.copiar()



if title_search: filtered_df = filtered_df[ filtered_df['title'].str.contains(title_search, case=False, na=False) | filtered_df['title_es'].str.contains(title_search, case=False, na=False) ]



if selected_genres: filtered_df = filtered_df[ filtered_df[['genres','genres.1','genres.2','genres.3','genres.4']] .apply(lambda row: any(g in row.values for g in selected_genres), axis=1) ]



===== MOSTRAR RESULTADOS =====

if not title_search and not selected_genres: st.markdown('



👉 Empieza a buscar una película o selecciona un género.



', unsafe_allow_html=True) else: if filtered_df.empty: st.warning("No se encontraron películas con esos filtros 😢") else: cols = st.columns(6)

    for i, (_, row) in enumerate(filtered_df.iterrows()):

        tmdb_link = row["LINK TMDBLD"]

        tmdb_id = tmdb_link.split("/")[-1] if tmdb_link else None

        rating = row.get("rating", None)



        # ===== CLASIFICAR RATING =====
```

```
        if pd.isna(rating):
            rating_html = '<div class="rating-badge rating-medium">⭐ N/A</div>'
        elif rating >= 4:
            rating_html = f'<div class="rating-badge rating-good">🟢 {rating:.1f}</div>'
        elif rating >= 2.5:
            rating_html = f'<div class="rating-badge rating-medium">🟡 {rating:.1f}</div>'
        else:
            rating_html = f'<div class="rating-badge rating-bad">🔴 {rating:.1f}</div>'

        # ===== POSTER =====
        poster_url = None
        if tmdb_id:
            try:
                response = requests.get(
                    f"https://api.themoviedb.org/3/movie/{tmdb_id}",
                    params={"api_key": "9f0bd3de6d61891223e72ec93de05459", "language": "es-
ES"}
                )
                if response.status_code == 200:
                    data = response.json()
                    poster_path = data.get("poster_path")
                    if poster_path:
                        poster_url = f"https://image.tmdb.org/t/p/w300{poster_path}"
            except:
                pass
```

```
        with cols[i % 6]:
            if poster_url:
                st.markdown(
                    f"""
                    <div class="movie-card">
                        <img src="{poster_url}" width="100%">
                        {rating_html}
                        <div class="overlay">
                            <a href="{tmdb_link}" target="_blank">Ver ficha</a>
                        </div>
                    </div>
                    """,
                    unsafe_allow_html=True
                )
            else:
                st.write("🎞️ Sin póster disponible")
```

Enlace paa acceder al sistema Recomendador de peliculas

https://github.com/henrymunoz/cineapp

4.4 ♻️ Flujo alternativo con KNIME

Infograma del flujo en KNIME:

- CSV Reader ← movies.csv
- CSV Reader ← ratings.csv
- GroupBy → Calcula promedio de rating por movieId

🔹 Joiner → Une movies.csv + ratings.csv por movieId

🔹 Python Script → Aplica CountVectorizer + cosine_similarity

🔹 Table View → Muestra títulos recomendados

🔹 Image Viewer → Muestra pósters desde TMDB API

Nodos utilizados:

- CSV Reader (para movies.csv y ratings.csv)
- GroupBy (para calcular promedio de puntuaciones)
- Joiner (para unir datos)
- Python Script (para similitud y recomendación)
- Table View (para mostrar resultados)
- Image Viewer (para mostrar pósters)

Ventajas:

- Visualización clara del flujo
- Documentación automática del proceso
- Ideal para presentaciones académicas

4.5 🖼️ Generación de imágenes tipo catálogo

```
import requests
import re


def clean_title(title):
```

```
    return re.sub(r"\s*\(.*?\)", "", title).strip()


def get_poster(title):
    api_key = 'TU_API_KEY'
    cleaned = clean_title(title)
    url = f"https://api.themoviedb.org/3/search/movie?api_key={api_key}&query={cleaned}"
    response = requests.get(url).json()

    if 'results' in response and response['results']:
        poster_path = response['results'][0].get('poster_path')
        if poster_path:
            return f"https://image.tmdb.org/t/p/w500{poster_path}"

    return "https://via.placeholder.com/300x450?text=No+Image"
```

En Streamlit:

```
for title in recs:
    poster_url = get_poster(title)
    st.image(poster_url, caption=title, use_column_width=True)
```

5. 🤖 Prompts y herramientas de IA utilizadas

Prompts clave:

- "Crea un recomendador de películas en Python con similitud de géneros"
- "Genera interfaz visual con Streamlit"

- "Diseña flujo en KNIME para recomendación"
- "Obtén imágenes de películas desde TMDB API"

Herramientas de IA:

- **Copilo**t: Generación de código y estructura del informe
- **ChatGPT / Gemini / Claude**: Redacción, revisión y mejora de contenido
- **Canvas**: Presentación visual del proyecto

6. 📊 Resultados

- Sistema funcional que recomienda películas similares por género.
- Interfaz visual clara y atractiva.
- Imágenes de catálogo integradas.
- Flujo alternativo en KNIME documentado y ejecutable.
- Código portable a Colab o local.

7. ⚠️ Errores encontrados y soluciones aplicadas

Error	Solución aplicada
NameError: name 'clean_title' is not defined	Se definió la función clean_title antes de usarla en get_poster
localhost rechazó la conexión	Se corrigió la ejecución usando python -m streamlit run recomendador.py
streamlit no se reconoce como comando	Se ejecutó Streamlit desde Python directamente y se revisó el PATH
No se mostraba imagen de película seleccionada	Se agregó visualización previa con get_poster(selected_movie)

8. 🙏 Bibliografía

- **MovieLens Dataset**: https://grouplens.org/datasets/movielens/
- **TMDB API**: https://www.themoviedb.org/documentation/api
- **KNIME Documentation**: https://docs.knime.com/
- **Streamlit Documentation**: https://docs.streamlit.io/
- **Scikit-learn**: https://scikit-learn.org/
- **Pandas:** https://pandas.pydata.org/
- **Python**: https://www.python.org/

9. 🙏 Agradecimientos

Agradecemos al equipo docente del curso de Inteligencia Artificial, a los desarrolladores de herramientas como Copilot, KNIME, Streamlit y TMDB, y a las plataformas que ofrecen datasets abiertos para el aprendizaje. También valoramos el trabajo colaborativo entre los integrantes del equipo.

10. ✅ Conclusiones

Este proyecto demuestra cómo aplicar técnicas de análisis de datos e inteligencia artificial para construir un sistema funcional y visualmente atractivo. La integración de herramientas como KNIME y Streamlit permite abordar el problema desde enfoques complementarios: uno visual y otro programado. El uso de IA generativa facilita la documentación, el desarrollo y la presentación del proyecto.