Relatório do Projeto Final
Agente de IA Para Recomendação de Filmes

Disciplina: COCADA

Tema: Integração do Método PCA com Agentes de Inteligência Artificial

Aluno: Isaac Maiolino Vianna

DRE: 124023330

Data: 9 de Dezembro de 2025

---

1. Visão Geral do Projeto

1.1. Introdução e Objetivo

O objetivo deste projeto foi desenvolver um sistema de recomendação de filme, que une a precisão matemática de métodos como o PCA com a capacidade de interpretação de linguagem natural dos Grandes Modelos de Linguagem (LLMs).

O projeto resolve o problema da **sobrecarga de escolha** (o paradoxo de ter muitos filmes disponíveis e não saber o que assistir), oferecendo recomendações personalizadas baseadas no histórico real de usuários, além de responder também qualquer pergunta sobre o banco de dados analisado.

1.2. A Solução Proposta

Diferente de sistemas de recomendação tradicionais, este agente é capaz de:

1. **Calcular recomendações matematicamente** usando uma implementação manual do PCA (Análise de Componentes Principais).
2. **Analisar o perfil do usuário** interpretando dados brutos para responder perguntas como "Qual o gênero favorito?".
3. **Buscar informações gerais** (RAG) sobre filmes, diretores e sinopses usando um banco de dados vetorial.
4. **Interagir em linguagem natural**, permitindo que o usuário converse com o sistema como se fosse um especialista em cinema.

1.3. Fonte de Dados

Para treinar o modelo matemático e alimentar o banco de dados, utilizou-se o dataset **MovieLens Small Latest**, disponível publicamente através no Kaggle.

Dados Utilizados: 100.000 avaliações de 600 usuários sobre 9.000 filmes.

- **Arquivos Principais: ratings.csv** (Matriz de notas) e **movies.csv** (Metadados dos filmes).

---

2. Arquitetura do Sistema

O sistema opera sobre três pilares principais, orquestrados por um grafo de decisão (LangGraph):

1. **Motor Matemático (Math Tool):** Responsável por processar a matriz de Usuários x Filmes e calcular previsões de notas.
2. **Memória de Longo Prazo (RAG Tool):** Um banco de dados vetorial local que permite ao agente buscar informações semânticas sobre os filmes.
3. **O Agente (Cérebro):** Um modelo Google Gemini que interpreta a intenção do usuário e decide qual ferramenta acionar.

---

3. Documentação Técnica dos Arquivos

A seguir, detalha-se a implementação de cada componente do projeto.

3.1. Configuração e Segurança (.env e .gitignore)

Para garantir a segurança e a portabilidade do código, foram utilizados arquivos de configuração padrão.

- **.env:** Arquivo responsável por armazenar variáveis de ambiente sensíveis, especificamente a **GEMINI_API_KEY**. Isso impede que chaves de acesso sejam escritas diretamente no código.
- **.gitignore:** Arquivo essencial para instruir o sistema a ignorar arquivos pesados (como a pasta **data/** com os CSVs), arquivos temporários (**__pycache__, venv/**) e, o arquivo de senhas (**.env**).

3.2. Interface de Execução (main.py)

Este é o ponto de entrada da aplicação. Ele serve como a interface entre o usuário e o Agente Inteligente.

- **Inicialização:** Carrega as variáveis de ambiente e importa o grafo compilado (**app_graph**).
- **Loop de Interação:** Mantém o programa rodando continuamente, aguardando inputs do usuário.
- **Robustez:** Implementa tratamento de exceções (**try/except**) para garantir que erros de API ou cálculo não encerrem o programa derrepente.
- **Fluxo:** Captura a pergunta → Envia para o Grafo → Recebe o estado final → Exibe apenas a resposta processada pelo agente.

3.3. O Orquestrador (src/agent.py)

Este arquivo é o controlador central, implementado com a biblioteca **LangGraph**. Ele define uma máquina de estados que conduz a conversa.

Destaques da Implementação:

- **Múltiplas Personalidades:** Utiliza duas instâncias do Gemini:
    - *Padrão (Temperatura 0):* Para triagem lógica e formatação de listas.
    - *Analista (Temperatura 0.5):* Para interpretar dados históricos e inferir gostos do usuário com criatividade.
- **Triagem Estruturada:** Usa um modelo Pydantic (**TriagemFilmes**) para classificar a intenção do usuário em quatro categorias: RECOMENDAR, HISTORICO_USUARIO, INFO_GERAL ou PEDIR_INFO.
- **Nós de Execução:**
    - **node_recomendacao:** Aciona o cálculo do PCA.
    - **node_historico:** Busca dados brutos e pede para o LLM "Analista" interpretá-los.
    - **node_info_rag:** Busca contexto no banco vetorial.
- **Grafo de Decisão:** Define as arestas condicionais. Por exemplo: se a intenção é "RECOMENDAR" mas o usuário não forneceu um ID, o grafo desvia para o nó **node_pedir_id**.

3.4. O Motor Matemático (src/PCA.py)

Esta classe implementa o método de **Aproximação de Posto K** (neste caso, k = 20)

Etapas do Cálculo Implementado (fit):

1. **Matriz de Utilidade:** Transforma os logs de avaliações em uma matriz A (Usuários x Filmes).
2. **Centralização:** Subtrai a média de cada coluna para centralizar os dados na origem.
3. **Matriz de Covariância:** Calcula C = A^T A. Esta matriz simétrica descreve como os filmes variam em conjunto.
4. **Decomposição Espectral:** Resolve a equação característica det(C - lambda) = 0 usando **np.linalg.eigh** para encontrar os autovalores (lambda) e autovetores (v).
5. **Redução de Dimensionalidade:** Seleciona os k autovetores associados aos maiores autovalores, formando a matriz de transformação V_k.
6. **Projeção:** Calcula os coeficientes dos usuários (c = A . V_k), reduzindo a representação de cada usuário para apenas 20 dimensões.

Por que escolhi k=20? A escolha do Posto 20 serve para encontrar um meio-termo ideal. Se usásse apenas k=1, o sistema recomendaria apenas os filmes mais populares para todo mundo, ignorando gostos pessoais. Se usássemos todos os componentes, o cálculo ficaria muito lento e pesado. Com 20 componentes, conseguimos capturar características suficientes do gosto das pessoas sem sobrecarregar o processador.

Método de Recomendação (recommend):

Realiza a reconstrução da matriz aproximada usando a fórmula. As maiores notas preditas (para filmes não assistidos) tornam-se as recomendações.

3.5. A Memória Semântica (src/RAG.py)

Este módulo gerencia o banco de dados vetorial para busca semântica (Retrieval-Augmented Generation).

Otimizações Implementadas:

- **Embeddings Locais:** Utiliza a biblioteca **HuggingFaceEmbeddings** (modelo all-MiniLM-L6-v2) para processar os textos localmente na CPU. Isso elimina a dependência de APIs pagas para vetorização.
- **Sistema de Cache (Persistência):**
    - Na primeira execução, o sistema processa os 9.000 filmes, cria o índice FAISS e o salva no disco (save_local).
    - Nas execuções seguintes, ele detecta a pasta existente e carrega o índice instantaneamente (load_local), reduzindo o tempo de inicialização de minutos para milissegundos.
- **FAISS:** Utiliza a biblioteca FAISS para realizar buscas de similaridade extremamente rápidas, retornando os filmes mais relevantes para o contexto da pergunta do usuário.

---

4. Guia de Instalação e Execução

Para rodar este projeto em qualquer máquina, é necessário seguir os passos de configuração abaixo.

4.1. Pré-requisitos

- Python 3.9+ instalado no computador.
- VS Code (ou outro editor de código).
- Uma chave de API do Google AI Studio (Gemini).

4.2. Instalação das Bibliotecas

O projeto possui um arquivo requirements.txt que lista todas as dependências necessárias (LangChain, FAISS, Pandas, Numpy, etc.). Para instalar tudo de uma vez, execute o seguinte comando no terminal:

pip install -r requirements.txt

4.3. Configuração do Banco de Dados

1. Crie uma pasta chamada data na raiz do projeto.
2. Baixe o dataset "MovieLens Small Latest" (do Kaggle).
3. Extraia e coloque os arquivos ratings.csv e movies.csv dentro da pasta data.

4.4. Configuração da API Key

1. Crie um arquivo chamado .env na raiz do projeto.
2. Cole sua chave de API seguindo o formato: **GEMINI_API_KEY=Sua_Chave_Aqui.**

---

5. Cenários de Teste

Abaixo estão listadas sugestões de perguntas para validar se cada funcionalidade do Agente (Matemática, Análise, RAG e Triagem) está operando corretamente.

5.1. Testando a Recomendação (PCA)

Objetivo: *Verificar se o PCA está calculando e retornando a lista de sugestões.*

- **Comando:** "Recomende filmes para o usuário 1."
- **Comando:** "O que o usuário 10 deveria assistir?"
- **Comando:** "Me dê 5 sugestões para o user 42."

5.2. Testando o Analista de Dados (Histórico e Perfil)

Objetivo: *Verificar se o Agente está lendo os dados brutos e interpretando o gosto do usuário.*

- **Comando:** "Quais filmes o usuário 1 já assistiu?"
- **Comando:** "Qual é o gênero favorito do usuário 1?" *(O agente deve deduzir isso)*
- **Comando:** "Quantos filmes o usuário 5 já viu e qual a média de notas dele?"
- **Comando:** "O usuário 10 gosta mais de Comédia ou de Terror?"

5.3. Testando o RAG (Conhecimento Geral)

Objetivo: *Verificar a busca semântica no banco vetorial local (FAISS).*

- **Comando:** "Qual o gênero do filme Toy Story?"
- **Comando:** "O que é o filme Jumanji?"
- **Comando:** "Me fale sobre o filme Matrix."

5.4. Testando a Triagem Inteligente

Objetivo: *Verificar se o sistema identifica quando faltam informações ou lida com erros.*

- **Comando:** "Quero uma recomendação de filme." *(O agente deve pedir o ID)*
- **Comando:** "Recomende filmes para o usuário 999999." *(O agente deve informar que o usuário não existe)*

---

6. Conclusão

O projeto demonstrou como é possível integrar conceitos da matemática em sistemas modernos de IA. A integração desses cálculos rígidos com a flexibilidade de um Agente Inteligente resultou em uma aplicação, capaz de oferecer não apenas números, mas insights personalizados sobre cinema.