

Modelo Preditivo de Recomendações IMDb: Arquitetura e Execução

Carlos et al Labs

15 de novembro de 2025

1 Visão Geral

O objetivo deste trabalho é recomendar filmes do catálogo IMDb para usuários da plataforma ALV a partir do histórico presente no Data Warehouse (`dw_alv`). O componente `src/recommender` encapsula todo o fluxo de preparação de dados, treinamento e inferência. O script `run_recommender.py` orquestra a execução e publica os resultados na tabela `imdb_alv.model_infer`, de onde o ETL incremental consome para alimentar a fato `dw_alv.ModelPrediction`.

2 Fontes de Dados

2.1 DW Operacional

- `dw_alv.avaliacao`: fato principal que contém notas atribuídas por usuário e filme.
- `dw_alv.filme`: dimensão de filmes com duração, ano e gênero principal.
- `dw_alv.receita` e `dw_alv.endereco`: usados para inferir o estado predominante de cada usuário, permitindo personalização geográfica.

Essas tabelas são lidas via SQLAlchemy. Para desenvolvimento offline, o módulo `datasets.py` também suporta carregar CSVs equivalentes localizados em `data/CSVs`.

2.2 IMDb Externo

As características dos filmes candidatos são extraídas do arquivo `aws/imdb_movies.parquet`, obtido pelo notebook `aws/external-data-imdb.ipynb`. Os campos utilizados são: título, ano, duração, gêneros, nota média e número de votos. Apenas títulos com $nota \geq 6.0$ e $votos \geq 1000$ entram no ranking para garantir qualidade.

3 Feature Engineering

O dataset supervisionado combina avaliações dos usuários com metadados de filme e do usuário, resultando nas seguintes features:

Feature	Tipo	Origem
anodelancamento	Numérica	dw_alv.filme
duracaomin	Numérica	dw_alv.filme
generonome	Categórica	dw_alv.filme
estado	Categórica	Derivada de dw_alv.receita + dw_alv.endereco

As notas (*nota*) formam o alvo do modelo. Para lidar com múltiplos gêneros no IMDb, selecionamos o gênero principal (primeiro da lista) a fim de manter compatibilidade com o esquema atual, mas preservando o sinal dominante de preferência.

4 Arquitetura do Modelo e Código

O modelo utiliza o `RandomForestRegressor` do `scikit-learn` dentro de um pipeline. O objetivo foi capturar relações não lineares entre atributos (ano, duração, gênero e estado) e as notas observadas, mantendo robustez a outliers e interpretabilidade básica (importâncias de features). Optamos pela floresta aleatória em vez de regressão linear ou redes neurais por entregar bom desempenho em dados tabulares heterogêneos e exigir pouco tuning, prática comum em projetos maduros.

4.1 Treinamento

O código responsável pela montagem e treinamento do pipeline está localizado em `src/recommender/model`.

```
def build_model() -> Pipeline:
    preprocessor = ColumnTransformer(
        transformers=[
            ("num", StandardScaler(), NUMERIC_COLUMNS),
            ("cat", OneHotEncoder(handle_unknown="ignore", sparse_output=False),
             CATEGORICAL_COLUMNS),
        ]
    )
    regressor = RandomForestRegressor(
        n_estimators=300,
        max_depth=12,
        min_samples_split=10,
        random_state=42,
        n_jobs=-1,
    )
    return Pipeline([("preprocess", preprocessor), ("regressor", regressor)])
```

O `ColumnTransformer` aplica normalização nas colunas numéricas (`anodelancamento` e `duracaomin`) e One-Hot Encoding nas categóricas (`generonome` e `estado`). Em seguida, o `RandomForestRegressor` aprende os padrões de avaliação. A função `train_model` simplesmente executa `model.fit(training_df[FEATURE_COLUMNS], training_df["nota"])`.

4.2 Montagem do Dataset

Em `src/recommender/datasets.py`, `build_training_dataset` integra as tabelas do DW ou os CSVs locais:

```
training = (
    avaliacao[["avaliacao", "usuarios", "filmes", "nota"]]
    .merge(filmes_subset, on="filmes", how="left")
    .merge(user_state, on="usuarios", how="inner")
)
```

As colunas `user_state` vêm de um agrupamento por usuário com escolha da moda (*most_frequent*). *Opipelinereproduzocomportamentooperacional, garantindoequivalenciaentreambiente*

4.3 Geração de Recomendações

O módulo `src/recommender/predictor.py` utiliza a saída do modelo para ranquear filmes:

```
state_frame = feature_ready.copy()
state_frame["estado"] = state
preds = model.predict(state_frame[FEATURE_COLUMNS])
state_frame["predicacao"] = np.clip(preds, 1.0, 5.0)
```

Após gerar previsões para cada estado, agrupamos e selecionamos o Top-25 por UF usando `groupby("estado").head(TOP_K_PER_STATE)`. O dataframe final é persistido na tabela `imdb_alv.model_infer` via `write_table` (módulo `database.py`), garantindo atomicidade com transações SQLAlchemy.

5 Inferência e Publicação

Após o treinamento, aplicamos o modelo aos filmes do IMDb:

1. Replicamos o conjunto de candidatos para cada estado disponível.
2. Geração de notas previstas (\hat{y}) e clipping no intervalo $[1, 5]$.
3. Ranqueamento e seleção do Top-25 por estado (valor configurável em `src/recommender/`).

O resultado final inclui as colunas exigidas pelo DW (título, ano, duração, gênero, métricas IMDb, estado e predição). O módulo `pipeline.py` persiste o dataframe tanto no PostgreSQL quanto em `aws/imdb_model_infer.parquet`.

6 Funcionamento Detalhado e Estrutura dos Dados

6.1 Dados Operacionais

As tabelas do DW seguem o formato de estrela clássico:

- Fato Avaliação: chaves `usuarios`, `filmes`, `produtor`, `calendario`, colunas de medida (nota). Nesse estágio, as notas estão no intervalo inteiro $[1, 5]$.

- Dimensão Filme: inclui duração, ano, gênero principal e texto do título; a chave filmesk é utilizada em todas as junções.
- Dimensão Usuário e Endereço: a tabela receita liga usuariosk a enderecosk. O script datasets.py utiliza esse elo para inferir o estado predominante do usuário.

Para manter consistência entre ambientes, a pasta data/CSVs replica o conteúdo final do DW, permitindo execuções locais sem depender da conexão PostgreSQL.

6.2 Fluxo de Execução

O script `src/recommender/pipeline.py` organiza todas as etapas:

1. Montagem do dataset: `build_training_dataset` retorna o dataframe supervisionado e a lista de estados encontrados. Caso o DW esteja indisponível, basta chamar a função com `engine=None` e `load_from_database=False` para utilizar os CSVs.
2. Treinamento: `train_model` recebe o dataframe e devolve o pipeline ajustado. Graças ao `ColumnTransformer`, não precisamos manter steps manuais de pré-processamento.
3. Inferência: `prepare_candidate_movies` filtra o parquet do IMDb, e `generate_predictions` replica cada título para todos os estados, alimentando o modelo e ranqueando os resultados.
4. Persistência: `write_table` grava em `imdb_alv.model_infer` via SQLAlchemy dentro de uma transação, garantindo idempotência.

6.3 Comportamento do Modelo

O `RandomForestRegressor` trabalha com árvores independentes; cada árvore aprende combinações de ano, duração, gênero e estado para prever a nota. O uso de `n_estimators=100` traz estabilidade (menos variância), enquanto `max_depth=12` evita overfitting.

Como o alvo está na escala de `[1,5]`, aplicamos `np.clip` nas previsões para manter o intervalo consistente com o domínio original e facilitar a interpretação downstream.

A arquitetura também foi pensada para suportar métricas internas futuramente: `pipeline.py` retorna o dataframe final, permitindo cálculos de RMSE/MAE em notebooks como `test_model.ipynb`. A estrutura modular garante facilidade de auditoria: qualquer dado utilizado passa pelos módulos `datasets`, `modeling` e `predictor`, todos com docstrings e comentários explicativos.