Busca-avancada

Sistema de busca avançada para catálogo de livros usando técnicas de busca fuzzy e semântica com Databricks Vector Search.

Características

- **Busca Fuzzy**: Correspondências aproximadas usando RapidFuzz
- **Busca Semântica**: Busca por significado usando embeddings e Databricks Vector Search
- **RRF (Reciprocal Rank Fusion)**: Combina resultados de diferentes métodos de busca
- **Execução Local**: Suporte para execução local usando credenciais Databricks via .env
- **AWS Bedrock**: Integração com AWS Bedrock como alternativa ao Databricks para LLM
- **Multi-Provider LLM**: Escolha entre Databricks ou AWS Bedrock para geração de sugestões

Requisitos

- **Python 3.10 ou 3.11** (Recomendado para Windows)
- Python 3.8+ (mínimo)
- Acesso ao Databricks (host e token)
- Dependências listadas em `requirements.txt`

Instalação

> **WINDOWS:** Se você está no Windows com Python 3.13, **recomendamos usar Python 3.10 ou 3.11** para evitar problemas de compilação. Veja o guia: PYTHON_SETUP.md

> **Problemas na instalação?** Consulte o INSTALL.md para troubleshooting detalhado.

Instalação Rápida

1. Clone o repositório
2. Instale as dependências:

```
# Método 1: Via requirements.txt (recomendado)
pip install -r requirements.txt

# Método 2: Instalação direta
pip install pandas python-dotenv rapidfuzz databricks-vectorsearch databricks-
langchain boto3
```

3. Edite o arquivo `.env.example`, renomeie para `.env` e adicione suas credenciais:

```
# Databricks (obrigatório para Vector Search)
DATABRICKS_HOST=seu-workspace.cloud.databricks.com
DATABRICKS_TOKEN=dapi1234567890abcdef
```

```
# AWS Bedrock (opcional - alternativa ao Databricks LLM)
AWS_BEARER_TOKEN_BEDROCK=bedrock-api-key-abcde
AWS_REGION=us-east-1
AWS_BEDROCK_MODEL_ID=anthropic.claude-3-sonnet-20240229-v1:0

# Escolha o provider LLM: 'databricks' ou 'bedrock'
LLM_PROVIDER=databricks

# Ambiente
WHO_IS_RUNNING_THIS=local
```

4. **(Opcional) Configure AWS CLI para usar Bedrock**:

```
aws configure
# Insira suas credenciais AWS
```

Execução Local

Existem duas formas de testar localmente:

Opção 1: Script de teste dedicado

```
python test_local.py
```

O script executa múltiplos testes:

- Busca por texto livre ("Lei Maria da Penha")
- Busca por campos específicos (título: "Python")

Opção 2: Executar o main.py diretamente

Navegue até a pasta do módulo e execute:

```
cd src/gold/busca
python main.py
```

O arquivo main.py agora possui um bloco `if __name__ == "__main__"` que:

- Carrega automaticamente as credenciais do `.env`
- Define `WHO_IS_RUNNING_THIS=local`
- Executa uma busca de exemplo
- Exibe os resultados formatados

Estrutura do Projeto

```
Busca-avancada-Gold/
├── src/
│   ├── gold/
│   │   └── busca/              # Módulo principal de busca
│   │       ├── main.py         # Função de consolidação
│   │       ├── fuzzy_search.py # Implementação de busca fuzzy
│   │       ├── semantic_search.py  # Implementação de busca semântica
│   │       ├── rrf.py          # Reciprocal Rank Fusion
│   │       └── books_search.csv    # Dataset de livros
│   └── utils/                  # Utilitários compartilhados
│       ├── dynamic_cat/        # Configuração dinâmica de catálogos
│       ├── aws_utils/          # Utilitários AWS
│       └── bugsnag_utils/      # Configuração Bugsnag
├── test_local.py               # Script de teste local
├── .env.example                # Exemplo de configuração
└── requirements.txt            # Dependências Python
```

Como Funciona

1. Busca por Texto Livre

```
data = {
    'searchQuery': 'Lei Maria da Penha',
    'selectedFields': {},
    'userCatalogs': ['uuid1', 'uuid2']
}
```

Combina busca exata, fuzzy e semântica usando RRF.

2. Busca por Campos Específicos

```
data = {
    'searchQuery': '',
    'selectedFields': {
        'titulo': 'Python',
        'autores': 'Guido',
        'isbn': ''
    },
    'userCatalogs': ['uuid1', 'uuid2']
}
```

Busca fuzzy nos campos especificados.

Modos de Execução

Ambiente de Execução

O código suporta três modos através da variável `WHO_IS_RUNNING_THIS`:

1. `local`: Execução local com credenciais do .env
2. `ENDPOINT_NOTEBOOK`: Execução em notebook Databricks
3. `ENDPOINT_MLFLOW`: Execução em endpoint MLflow Databricks

Provider LLM

Escolha qual LLM usar para geração de sugestões através da variável `LLM_PROVIDER`:

1. `databricks` (padrão): Usa ChatDatabricks
2. `bedrock`: Usa AWS Bedrock (Claude 3)

Exemplo de uso:

```
# No .env
LLM_PROVIDER=bedrock

# Ou via código
os.environ["LLM_PROVIDER"] = "bedrock"
```

Para mais detalhes sobre a integração com AWS Bedrock, consulte: BEDROCK_MIGRATION.md

Configuração do Databricks

Obter Token de Acesso

1. Acesse seu workspace Databricks
2. Vá em **User Settings** > **Access Tokens**
3. Clique em **Generate New Token**
4. Copie o token gerado

Host do Databricks

O host é o domínio do seu workspace (sem `https://`):

- Exemplo: `dbc-a1b2c3d4-e5f6.cloud.databricks.com`

Configuração do AWS Bedrock

Pré-requisitos

1. **Conta AWS com acesso ao Bedrock**
2. **AWS CLI configurado**:

```
aws configure
```

3. **Modelo habilitado no Bedrock**:
    - Acesse AWS Console > Bedrock
    - Vá em "Model access"
    - Verifique o acesso aos modelos

Teste de Conexão

```
cd src/gold/busca
python bedrock_chat.py
```

Para documentação completa sobre AWS Bedrock, incluindo troubleshooting e comparação de resultados, consulte: BEDROCK_MIGRATION.md