Seção5.1_Part2_Embeddings_Locais

October 23, 2025

1 Seção 5.1 – Parte 2: Embeddings Locais

Objetivo: Gerar e analisar embeddings clássicos e modernos (TF-IDF, Word2Vec, BERT, Sentence-BERT) usando o dataset preparado no Notebook 1.

1.1 Conteúdo deste Notebook

1. **Carregamento de Dados**: Obter dataset diretamente do Elasticsearch
2. **TF-IDF**: Embeddings baseados em frequência de termos
3. **Word2Vec**: Embeddings contextuais clássicos
4. **BERT**: Embeddings bidirecionais modernos
5. **Sentence-BERT**: Otimizado para similaridade de sentenças
6. **Análise Detalhada**: Comparação de características de cada tipo

1.2 Sequência dos Notebooks

- **Notebook 1**: Preparação e Dataset
- **Notebook 2** (atual): Embeddings Locais
- **Notebook 3**: Embeddings OpenAI
- **Notebook 4**: Análise Comparativa dos Embeddings
- **Notebook 5**: Clustering e Machine Learning

1.3 IMPORTANTE: Carregamento de Dados

Este notebook **SEMPRE carrega os dados do Elasticsearch** (preparados no Notebook 1), garantindo: - Consistência entre notebooks - Mesmos IDs únicos (doc_0000, doc_0001, ...) - Rastreabilidade completa - Integridade dos dados

1.4 Configuração do Ambiente

Este notebook carrega as configurações do arquivo `setup/.env` e conecta ao Elasticsearch.

[1]:
```
#  Configuração de Variáveis de Ambiente
import os
from pathlib import Path

# Carregar python-dotenv
try:
    from dotenv import load_dotenv
```

```
    print("  python-dotenv disponível")

    env_paths = [
        Path.cwd() / 'setup' / '.env',
        Path.cwd() / '.env',
        Path.cwd() / 'setup' / 'config_example.env'
    ]

    env_loaded = False
    for env_path in env_paths:
        if env_path.exists():
            load_dotenv(env_path)
            print(f"  Arquivo .env carregado: {env_path}")
            env_loaded = True
            break

    if not env_loaded:
        print("   Nenhum arquivo .env encontrado")

except ImportError:
    print("   python-dotenv não instalado")

# Carregar configurações (otimizadas para 20 classes)
MAX_CHARS_PER_REQUEST = int(os.getenv('MAX_CHARS_PER_REQUEST', 32000))
BATCH_SIZE_SMALL_TEXTS = int(os.getenv('BATCH_SIZE_SMALL_TEXTS', 4))
BATCH_SIZE_MEDIUM_TEXTS = int(os.getenv('BATCH_SIZE_MEDIUM_TEXTS', 2))
BATCH_SIZE_LARGE_TEXTS = int(os.getenv('BATCH_SIZE_LARGE_TEXTS', 1))
DATASET_SIZE = int(os.getenv('DATASET_SIZE', 20000))
TEXT_MIN_LENGTH = int(os.getenv('TEXT_MIN_LENGTH', 20))
MAX_CLUSTERS = int(os.getenv('MAX_CLUSTERS', 20))
CLUSTERING_RANDOM_STATE = int(os.getenv('CLUSTERING_RANDOM_STATE', 42))
ELASTICSEARCH_HOST = os.getenv('ELASTICSEARCH_HOST', 'localhost')
ELASTICSEARCH_PORT = int(os.getenv('ELASTICSEARCH_PORT', 9200))

print(f"\n Configurações carregadas!")
print(f"   ELASTICSEARCH: {ELASTICSEARCH_HOST}:{ELASTICSEARCH_PORT}")
print(f"   CLUSTERING_RANDOM_STATE: {CLUSTERING_RANDOM_STATE}")
```

```
 python-dotenv disponível
 Arquivo .env carregado: /Users/ivanvarella/Documents/Dados/9 - Mestrado/1 -
Disciplinas 2025/2025.2/PPGEP9002 - INTELIGÊNCIA COMPUTACIONAL PARA ENGENHARIA
DE PRODUÇÃO - T01/1 - Extra - Professor/Projetos/Embeddings_5.1/src/setup/.env

 Configurações carregadas!
   ELASTICSEARCH: localhost:9200
   CLUSTERING_RANDOM_STATE: 42
```

[2]:
```
#  Imports Essenciais
print(" CARREGANDO IMPORTS")
print("=" * 40)

import re
import json
import warnings
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
from typing import List, Dict, Tuple, Optional
from sklearn.feature_extraction.text import TfidfVectorizer

print(" Imports básicos carregados")

# Configurações
warnings.filterwarnings('ignore')
pd.set_option('display.max_colwidth', 200)
plt.style.use('default')
sns.set_palette("husl")
plt.rcParams['figure.figsize'] = (10, 6)

print(" Configurações aplicadas")
```

```
 CARREGANDO IMPORTS
========================================
 Imports básicos carregados
 Configurações aplicadas
```

1.5 Carregamento de Dados

Este notebook carrega o dataset do Elasticsearch (salvo no Notebook 1) para garantir consistência.

[3]:
```
#  Inicializar Elasticsearch e Carregar Dataset
print(" INICIALIZANDO ELASTICSEARCH")
print("=" * 60)

# Importar módulo de cache
try:
    from elasticsearch_manager import (
        init_elasticsearch_cache, get_cache_status,
        save_embeddings_to_cache, load_embeddings_from_cache,
        check_embeddings_in_cache
    )
    print(" Módulo de cache carregado")
    CACHE_AVAILABLE = True
```

```
except ImportError as e:
    print(f"  Erro ao carregar módulo: {e}")
    CACHE_AVAILABLE = False

# Conectar ao Elasticsearch
if CACHE_AVAILABLE:
    print("\n  Conectando...")
    cache_connected = init_elasticsearch_cache(
        host=ELASTICSEARCH_HOST,
        port=ELASTICSEARCH_PORT
    )

    if cache_connected:
        print("  Conectado ao Elasticsearch!")
        status = get_cache_status()
        if status.get("connected"):
            print(f"  Total de documentos no cache: {status.get('total_docs',␣
 ↪0):,}")
    else:
        print("  Falha na conexão")
        CACHE_AVAILABLE = False
else:
    print("   Cache não disponível")
    cache_connected = False

print(f"\n  STATUS: {'  Cache ativo' if CACHE_AVAILABLE and cache_connected else␣
 ↪'  Cache inativo'}")
```

```
 INICIALIZANDO ELASTICSEARCH
============================================================
 Módulo de cache carregado

 Conectando…
 Conectado ao Elasticsearch (localhost:9200)
 Conectado ao Elasticsearch!
 Total de documentos no cache: 109,266

 STATUS:   Cache ativo
```

[4]:
```
#   CARREGAR DATASET DO ELASTICSEARCH
print("  CARREGANDO DATASET DO ELASTICSEARCH")
print("=" * 60)
print("   IMPORTANTE: Carregando dados salvos no Notebook 1")
print("              NÃO recriando o dataset!")

# Executar carregamento
if CACHE_AVAILABLE and cache_connected:
```

```
    try:
        from elasticsearch import Elasticsearch
        from elasticsearch_helpers import␣
 ↪load_all_documents_from_elasticsearch, print_dataframe_summary

        # Conectar ao Elasticsearch
        es = Elasticsearch([{
            'host': ELASTICSEARCH_HOST,
            'port': ELASTICSEARCH_PORT,
            'scheme': 'http'
        }])

        # Carregar TODOS os documentos usando Scroll API
        # Esta função está em elasticsearch_helpers.py e usa Scroll API
        # para buscar TODOS os documentos, mesmo que sejam >10.000
        df = load_all_documents_from_elasticsearch(
            es_client=es,
            index_name="documents_dataset",
            batch_size=1000,      # Docs por lote
            scroll_timeout='2m',  # Tempo de contexto
            verbose=True          # Mostrar progresso
        )

        # Gerar lista de doc_ids para uso posterior
        doc_ids = df['doc_id'].tolist()

        # Exibir resumo detalhado
        print_dataframe_summary(df, expected_docs=18000)

    except Exception as e:
        print(f"\n ERRO CRÍTICO ao carregar dataset: {e}")
        print(" Possíveis causas:")
        print("   1. Notebook 1 não foi executado")
        print("   2. Elasticsearch não está rodando")
        print("   3. Índice 'documents_dataset' não existe")
        raise
else:
    print("\n ERRO: Elasticsearch não disponível!")
    print(" Verifique:")
    print("   1. Docker está rodando: docker ps")
    print("   2. Elasticsearch ativo: http://localhost:9200")
    print("   3. Execute o Notebook 1 primeiro")
    raise RuntimeError("Elasticsearch não disponível")
```

```
 CARREGANDO DATASET DO ELASTICSEARCH
============================================================
  IMPORTANTE: Carregando dados salvos no Notebook 1
              NÃO recriando o dataset!
```

```
  Buscando documentos do índice 'documents_dataset'
    Método: Scroll API (recomendado para >10k docs)
    Tamanho do lote: 1,000 documentos
    Timeout do scroll: 2m

  Total de documentos disponíveis: 18,211
  Iniciando busca em lotes…
    Lote 1: 1,000 docs | Total acumulado: 1,000/18,211
    Lote 2: 1,000 docs | Total acumulado: 2,000/18,211
    Lote 3: 1,000 docs | Total acumulado: 3,000/18,211
    Lote 4: 1,000 docs | Total acumulado: 4,000/18,211
    Lote 5: 1,000 docs | Total acumulado: 5,000/18,211
    Lote 6: 1,000 docs | Total acumulado: 6,000/18,211
    Lote 7: 1,000 docs | Total acumulado: 7,000/18,211
    Lote 8: 1,000 docs | Total acumulado: 8,000/18,211
    Lote 9: 1,000 docs | Total acumulado: 9,000/18,211
    Lote 10: 1,000 docs | Total acumulado: 10,000/18,211
    Lote 11: 1,000 docs | Total acumulado: 11,000/18,211
    Lote 12: 1,000 docs | Total acumulado: 12,000/18,211
    Lote 13: 1,000 docs | Total acumulado: 13,000/18,211
    Lote 14: 1,000 docs | Total acumulado: 14,000/18,211
    Lote 15: 1,000 docs | Total acumulado: 15,000/18,211
    Lote 16: 1,000 docs | Total acumulado: 16,000/18,211
    Lote 17: 1,000 docs | Total acumulado: 17,000/18,211
    Lote 18: 1,000 docs | Total acumulado: 18,000/18,211
    Lote 19: 211 docs | Total acumulado: 18,211/18,211

  Scroll concluído e recursos liberados

  Processando 18,211 documentos em DataFrame…
  DataFrame criado com sucesso!

============================================================
  DATASET CARREGADO COM SUCESSO!
============================================================
  Shape: (18211, 4)
  Colunas: ['doc_id', 'text', 'category', 'target']
    Classes únicas: 20
  Total de documentos: 18,211
  IDs (amostra): ['doc_0000', 'doc_0001', 'doc_0002'] … ['doc_9997',
'doc_9998', 'doc_9999']

  VALIDAÇÃO:
     PASSOU: 18,211 documentos
     Dentro da expectativa: ~18,000 ±1,000
```

1.6 Embeddings Clássicos: TF-IDF

1.6.1 O que é TF-IDF?

TF-IDF (Term Frequency-Inverse Document Frequency) é um método clássico que pondera a importância de palavras:

- **TF (Term Frequency)**: Frequência do termo no documento
- **IDF (Inverse Document Frequency)**: Raridade do termo no corpus
- **Fórmula**: TF-IDF = TF × log(N/DF)
  - N = número total de documentos
  - DF = número de documentos contendo o termo

1.6.2 Características

- Simples e interpretável
- Rápido para calcular
- Baseline sólido
- Matriz esparsa (muitos zeros)
- Não captura contexto semântico

[5]:
```
# from elasticsearch import Elasticsearch
# es = Elasticsearch([{'host': 'localhost', 'port': 9200, 'scheme': 'http'}])
# if es.indices.exists(index='embeddings_tfidf'):
#     es.indices.delete(index='embeddings_tfidf')
#     print('  Índice embeddings_tfidf deletado')
# else:
#     print('   Índice não existe')
```

[6]:
```
#  Gerar Embeddings TF-IDF
print(" GERANDO EMBEDDINGS TF-IDF")
print("=" * 60)

# Verificar se já existe no cache
use_cache = os.getenv('USE_ELASTICSEARCH_CACHE', 'true').lower() == 'true'
force_regenerate = os.getenv('FORCE_REGENERATE_EMBEDDINGS', 'false').lower() ==␣
↪'true'

if use_cache and not force_regenerate and CACHE_AVAILABLE:
    all_exist, existing, missing =␣
↪check_embeddings_in_cache('embeddings_tfidf', doc_ids)

    if all_exist:
        print(" TF-IDF já existe no cache, carregando...")
        tfidf_embeddings = load_embeddings_from_cache('embeddings_tfidf',␣
↪doc_ids)
        if tfidf_embeddings is not None:
            print(f" TF-IDF carregado: {tfidf_embeddings.shape}")
            # Criar vectorizer vazio (será usado para análise)
```

```
            tfidf_vectorizer = TfidfVectorizer(max_features=4096, max_df=0.95,␣
 ↪min_df=2)
            tfidf_vectorizer.fit(df['text'])
        else:
            print("  Falha ao carregar, regenerando...")
            force_regenerate = True

if not use_cache or force_regenerate or not all_exist or tfidf_embeddings is␣
 ↪None:
    print("  Gerando TF-IDF...")

    # Criar vectorizer
    tfidf_vectorizer = TfidfVectorizer(
        max_features=4096,  # Limitar a 4096 features (máximo do Elasticsearch)
        max_df=0.95,        # Ignorar termos muito frequentes
        min_df=2,           # Ignorar termos muito raros
        ngram_range=(1, 2)  # Unigramas e bigramas
    )

    # Gerar embeddings
    tfidf_matrix = tfidf_vectorizer.fit_transform(df['text'])
    tfidf_embeddings = tfidf_matrix.toarray()

    print(f"  TF-IDF gerado: {tfidf_embeddings.shape}")
    print(f"    Vocabulário: {len(tfidf_vectorizer.vocabulary_):,} termos")
    print(f"    Densidade: {(tfidf_embeddings != 0).mean():.3f}")

    # Salvar no cache
    if use_cache and CACHE_AVAILABLE:
        print("  Salvando no Elasticsearch...")
        save_embeddings_to_cache(
            'embeddings_tfidf',
            tfidf_embeddings,
            doc_ids,
            df['text'].tolist(),
            'tfidf'
        )

print(f"\n  TF-IDF pronto: {tfidf_embeddings.shape}")
```

```
  GERANDO EMBEDDINGS TF-IDF
============================================================
  TF-IDF já existe no cache, carregando…
  Embeddings carregados: (18211, 4096) de 'embeddings_tfidf'
  TF-IDF carregado: (18211, 4096)

  TF-IDF pronto: (18211, 4096)
```

1.7 Word2Vec: Embeddings Contextuais

1.7.1 O que é Word2Vec?

Word2Vec (2013) foi revolucionário ao capturar similaridade semântica através de contexto:

- **Skip-gram**: Prediz palavras vizinhas dado uma palavra central
- **CBOW**: Prediz palavra central dado contexto
- **Janela deslizante**: Considera palavras próximas
- **Resultado**: Palavras similares ficam próximas no espaço vetorial

1.7.2 Características

- Captura similaridade semântica
- Embeddings densos
- Rápido após treinamento
- Palavras isoladas (não considera ordem global)
- Requer treinamento no corpus

[7]:
```
#  Carregar bibliotecas para Word2Vec, BERT e SBERT
print(" CARREGANDO BIBLIOTECAS DE EMBEDDINGS")
print("=" * 60)

# Gensim para Word2Vec
try:
    from gensim.models import Word2Vec
    print(" Gensim carregado")
    GENSIM_OK = True
except:
    print(" Gensim não disponível")
    GENSIM_OK = False

# Sentence Transformers para BERT e SBERT
try:
    from sentence_transformers import SentenceTransformer
    print(" Sentence Transformers carregado")
    TRANSFORMERS_OK = True
except:
    print(" Sentence Transformers não disponível")
    TRANSFORMERS_OK = False

print(f"\nStatus: Gensim={' ' if GENSIM_OK else ' '}, Transformers={' ' if␣
 ↪TRANSFORMERS_OK else ' '}")
```

```
 CARREGANDO BIBLIOTECAS DE EMBEDDINGS
============================================================
 Gensim carregado
 Sentence Transformers carregado

Status: Gensim= , Transformers=
```

[8]:
```
#  Gerar Embeddings Word2Vec
print("  GERANDO EMBEDDINGS WORD2VEC")
print("=" * 60)

if not GENSIM_OK:
    print("  Gensim não disponível, pulando Word2Vec")
    word2vec_embeddings = None
else:
    # Verificar cache
    if use_cache and not force_regenerate and CACHE_AVAILABLE:
        all_exist, _, _ = check_embeddings_in_cache('embeddings_word2vec',␣
 ↪doc_ids)
        if all_exist:
            print("  Word2Vec já existe, carregando...")
            word2vec_embeddings =␣
 ↪load_embeddings_from_cache('embeddings_word2vec', doc_ids)
            if word2vec_embeddings is not None:
                print(f"  Word2Vec carregado: {word2vec_embeddings.shape}")
            else:
                force_regenerate = True

    if not use_cache or force_regenerate or not all_exist or␣
 ↪word2vec_embeddings is None:
        print("  Treinando Word2Vec...")

        # Tokenizar textos
        tokenized_texts = [text.lower().split() for text in df['text']]

        # Treinar Word2Vec
        w2v_model = Word2Vec(
            sentences=tokenized_texts,
            vector_size=100,
            window=5,
            min_count=2,
            workers=4,
            epochs=10,
            seed=CLUSTERING_RANDOM_STATE
        )

        # Gerar embeddings por documento (média dos vetores de palavras)
        word2vec_embeddings = []
        for tokens in tokenized_texts:
            valid_vectors = [w2v_model.wv[word] for word in tokens if word in␣
 ↪w2v_model.wv]
            if valid_vectors:
                word2vec_embeddings.append(np.mean(valid_vectors, axis=0))
            else:
```

```
                word2vec_embeddings.append(np.zeros(100))

        word2vec_embeddings = np.array(word2vec_embeddings)

        print(f"  Word2Vec gerado: {word2vec_embeddings.shape}")
        print(f"    Vocabulário: {len(w2v_model.wv):,} palavras")

        # Salvar no cache
        if use_cache and CACHE_AVAILABLE:
            print("  Salvando no Elasticsearch...")
            save_embeddings_to_cache(
                'embeddings_word2vec',
                word2vec_embeddings,
                doc_ids,
                df['text'].tolist(),
                'word2vec'
            )

if word2vec_embeddings is not None:
    print(f"\n  Word2Vec pronto: {word2vec_embeddings.shape}")
```

```
  GERANDO EMBEDDINGS WORD2VEC
============================================================
  Word2Vec já existe, carregando…
  Embeddings carregados: (18211, 100) de 'embeddings_word2vec'
  Word2Vec carregado: (18211, 100)

  Word2Vec pronto: (18211, 100)
```

1.8 Embeddings Modernos: BERT e Sentence-BERT

1.8.1 BERT (2018) - Contextualização Bidirecional

Características: - Lê texto em ambas as direções simultaneamente - Attention mechanism - 768 dimensões (bert-base-uncased) - Contextualizado: mesma palavra, contextos diferentes

1.8.2 Sentence-BERT (2019) - Otimizado para Similaridade

Características: - Baseado em BERT mas otimizado para similaridade - 384 dimensões (all-MiniLM-L6-v2) - Ideal para clustering e busca semântica - Normalizado por padrão

[9]:
```
#  Gerar Embeddings BERT e Sentence-BERT
print("  GERANDO EMBEDDINGS BERT E SENTENCE-BERT")
print("=" * 60)

if not TRANSFORMERS_OK:
    print("  Sentence Transformers não disponível")
    bert_embeddings = None
    sbert_embeddings = None
```

```
else:
    # BERT
    print("\n BERT (bert-base-uncased)...")
    if use_cache and not force_regenerate and CACHE_AVAILABLE:
        all_exist, _, _ = check_embeddings_in_cache('embeddings_bert', doc_ids)
        if all_exist:
            print(" BERT já existe, carregando...")
            bert_embeddings = load_embeddings_from_cache('embeddings_bert',␣
 ↪doc_ids)
            if bert_embeddings is None:
                force_regenerate = True

    if not use_cache or force_regenerate or not all_exist or bert_embeddings is␣
 ↪None:
        print(" Gerando BERT...")
        bert_model = SentenceTransformer('bert-base-uncased')
        bert_embeddings = bert_model.encode(
            df['text'].tolist(),
            show_progress_bar=True,
            batch_size=32
        )
        print(f" BERT gerado: {bert_embeddings.shape}")

        if use_cache and CACHE_AVAILABLE:
            save_embeddings_to_cache(
                'embeddings_bert',
                bert_embeddings,
                doc_ids,
                df['text'].tolist(),
                'bert'
            )

    # Sentence-BERT
    print("\n Sentence-BERT (all-MiniLM-L6-v2)...")
    if use_cache and not force_regenerate and CACHE_AVAILABLE:
        all_exist, _, _ = check_embeddings_in_cache('embeddings_sbert', doc_ids)
        if all_exist:
            print(" Sentence-BERT já existe, carregando...")
            sbert_embeddings = load_embeddings_from_cache('embeddings_sbert',␣
 ↪doc_ids)
            if sbert_embeddings is None:
                force_regenerate = True

    if not use_cache or force_regenerate or not all_exist or sbert_embeddings␣
 ↪is None:
        print(" Gerando Sentence-BERT...")
        sbert_model = SentenceTransformer('all-MiniLM-L6-v2')
```

```
        sbert_embeddings = sbert_model.encode(
            df['text'].tolist(),
            show_progress_bar=True,
            batch_size=32
        )
        print(f"  Sentence-BERT gerado: {sbert_embeddings.shape}")

        if use_cache and CACHE_AVAILABLE:
            save_embeddings_to_cache(
                'embeddings_sbert',
                sbert_embeddings,
                doc_ids,
                df['text'].tolist(),
                'sbert'
            )

print(f"\n RESUMO DOS EMBEDDINGS LOCAIS:")
print(f"   TF-IDF: {tfidf_embeddings.shape if tfidf_embeddings is not None else␣
 ↪'N/A'}")
print(f"   Word2Vec: {word2vec_embeddings.shape if word2vec_embeddings is not␣
 ↪None else 'N/A'}")
print(f"   BERT: {bert_embeddings.shape if bert_embeddings is not None else 'N/
 ↪A'}")
print(f"   Sentence-BERT: {sbert_embeddings.shape if sbert_embeddings is not␣
 ↪None else 'N/A'}")
```

```
 GERANDO EMBEDDINGS BERT E SENTENCE-BERT
============================================================

 BERT (bert-base-uncased)…
 BERT já existe, carregando…
 Embeddings carregados: (18211, 768) de 'embeddings_bert'

 Sentence-BERT (all-MiniLM-L6-v2)…
 Sentence-BERT já existe, carregando…
 Embeddings carregados: (18211, 384) de 'embeddings_sbert'

 RESUMO DOS EMBEDDINGS LOCAIS:
   TF-IDF: (18211, 4096)
   Word2Vec: (18211, 100)
   BERT: (18211, 768)
   Sentence-BERT: (18211, 384)
```

1.9 Resumo e Próximos Passos

1.9.1 O que foi realizado neste notebook:

1. **Dados carregados do Elasticsearch** - Consistência garantida

2. **TF-IDF** - Embeddings clássicos baseados em frequência
3. **Word2Vec** - Embeddings contextuais treinados
4. **BERT** - Embeddings bidirecionais modernos
5. **Sentence-BERT** - Otimizado para similaridade

1.9.2 Próximo Notebook: Parte 3 - Embeddings OpenAI

No próximo notebook: - Embeddings da API OpenAI (text-embedding-3-small) - Tratamento de textos longos (sem truncamento) - Economia de custos com cache inteligente

[10]:
```
#  Resumo Final
print(" RESUMO FINAL - NOTEBOOK 2 COMPLETO")
print("=" * 60)
print(f" Dataset: {len(df):,} documentos (carregados do Elasticsearch)")
print(f" Embeddings gerados:")
print(f"   • TF-IDF: {tfidf_embeddings.shape if tfidf_embeddings is not None␣
 ↪else 'N/A'}")
print(f"   • Word2Vec: {word2vec_embeddings.shape if word2vec_embeddings is not␣
 ↪None else 'N/A'}")
print(f"   • BERT: {bert_embeddings.shape if bert_embeddings is not None else␣
 ↪'N/A'}")
print(f"   • Sentence-BERT: {sbert_embeddings.shape if sbert_embeddings is not␣
 ↪None else 'N/A'}")
print(f"\n Pronto para o Notebook 3: Embeddings OpenAI!")
```

```
 RESUMO FINAL - NOTEBOOK 2 COMPLETO
============================================================
 Dataset: 18,211 documentos (carregados do Elasticsearch)
 Embeddings gerados:
   • TF-IDF: (18211, 4096)
   • Word2Vec: (18211, 100)
   • BERT: (18211, 768)
   • Sentence-BERT: (18211, 384)

 Pronto para o Notebook 3: Embeddings OpenAI!
```