Seção5.1_Part5_Clustering_ML

October 23, 2025

1 Seção 5.1 – Parte 5: Clustering e Machine Learning

Objetivo: Aplicar técnicas de redução dimensional e clustering para avaliar a qualidade dos embeddings.

1.1 Conteúdo deste Notebook

1. **Redução Dimensional**: PCA, t-SNE e UMAP
2. **Clustering**: K-Means, DBSCAN e HDBSCAN
3. **Avaliação**: Métricas internas e externas
4. **Visualizações**: Plots 2D dos clusters
5. **Comparação Final**: Identificar melhor combinação embedding + algoritmo
6. **Sistema de Classificação**: Predizer categoria de novos textos

1.2 Sequência dos Notebooks

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

1.3 Matriz de Análise

5 Embeddings × 3 Técnicas de Redução × 3 Algoritmos de Clustering = 45 Combinações

Vamos identificar qual combinação melhor captura a estrutura semântica dos documentos!

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

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 (defaults atualizados para 20 classes)
ELASTICSEARCH_HOST = os.getenv('ELASTICSEARCH_HOST', 'localhost')
ELASTICSEARCH_PORT = int(os.getenv('ELASTICSEARCH_PORT', 9200))
DATASET_SIZE = int(os.getenv('DATASET_SIZE', 20000))
# Tamanhos de gráficos em POLEGADAS (não ler do .env)
PLOT_WIDTH = 12   # 12 inches = ~30cm
PLOT_HEIGHT = 6   # 6 inches = ~15cm
CLUSTERING_RANDOM_STATE = int(os.getenv('CLUSTERING_RANDOM_STATE', 42))
MAX_CLUSTERS = int(os.getenv('MAX_CLUSTERS', 20))

print(f"\n Configurações carregadas!")
print(f"   ELASTICSEARCH: {ELASTICSEARCH_HOST}:{ELASTICSEARCH_PORT}")
print(f"   PLOT_SIZE: {PLOT_WIDTH}x{PLOT_HEIGHT} inches")
print(f"   RANDOM_STATE: {CLUSTERING_RANDOM_STATE}")
print(f"   MAX_CLUSTERS: {MAX_CLUSTERS}")
```

```
 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
   PLOT_SIZE: 12x6 inches
   RANDOM_STATE: 42
   MAX_CLUSTERS: 20
```

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

```
print("=" * 60)

import warnings
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from typing import Dict, List, Tuple

#  ATIVAR RENDERIZAÇÃO INLINE DE GRÁFICOS
%matplotlib inline


# Redução dimensional
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import umap

# Clustering
from sklearn.cluster import KMeans, DBSCAN
import hdbscan

# Métricas
from sklearn.metrics import (
    silhouette_score, calinski_harabasz_score, davies_bouldin_score,
    adjusted_rand_score, normalized_mutual_info_score,
    homogeneity_score, completeness_score, v_measure_score
)

print("  Imports carregados")

# Configurações
warnings.filterwarnings('ignore')
pd.set_option('display.max_colwidth', 100)

# Configurações matplotlib
plt.rcParams['figure.dpi'] = 100
plt.rcParams['savefig.dpi'] = 100
plt.rcParams['font.size'] = 10
plt.rcParams['axes.titlesize'] = 12
plt.rcParams['axes.labelsize'] = 10

# Estilo com fallback
try:
    plt.style.use('seaborn-v0_8-darkgrid')
except:
```

```
    plt.style.use('seaborn-darkgrid')

sns.set_palette("husl")

print(" Configurações de visualização aplicadas")
print(" Matplotlib inline ativado")
```

```
 CARREGANDO IMPORTS
============================================================

/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/.venv/lib/python3.12/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm

 Imports carregados
 Configurações de visualização aplicadas
 Matplotlib inline ativado
```

[3]:
```
#  INICIALIZAR ELASTICSEARCH
print(" INICIALIZANDO ELASTICSEARCH")
print("=" * 60)

try:
    from elasticsearch_manager import (
        init_elasticsearch_cache,
        load_embeddings_from_cache,
        get_cache_status
    )
    print(" Módulo elasticsearch_manager carregado")
    CACHE_AVAILABLE = True
except ImportError as e:
    print(f" Erro ao importar elasticsearch_manager: {e}")
    print(" Verifique se o arquivo elasticsearch_manager.py existe")
    CACHE_AVAILABLE = False

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

```
    if cache_connected:
        print("✅ Conectado ao Elasticsearch!")

        # Verificar status do cache
        try:
            status = get_cache_status()
            if status:
                print("\n📊 Status do Cache:")
                for key, value in status.items():
                    print(f"   {key}: {value}")
        except:
            pass
    else:
        print("⚠️ Falha na conexão com Elasticsearch")
        CACHE_AVAILABLE = False
else:
    cache_connected = False

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

if not CACHE_AVAILABLE or not cache_connected:
    print("\n⚠️  AVISO: Elasticsearch não disponível!")
    print("📋 Verifique:")
    print("   1. Docker está rodando: docker ps")
    print("   2. Elasticsearch ativo: http://localhost:9200")
    print("   3. Notebooks 1, 2 e 3 foram executados")
```

```
🚀 INICIALIZANDO ELASTICSEARCH
============================================================
✅ Módulo elasticsearch_manager carregado

🔌 Conectando ao Elasticsearch…
✅ Conectado ao Elasticsearch (localhost:9200)
✅ Conectado ao Elasticsearch!

📊 Status do Cache:
  connected: True
  host: localhost:9200
  indices: {'documents_dataset': {'exists': True, 'doc_count': 18211,
'size_mb': 26.54}, 'embeddings_tfidf': {'exists': True, 'doc_count': 18211,
'size_mb': 325.16}, 'embeddings_word2vec': {'exists': True, 'doc_count': 18211,
'size_mb': 37.2}, 'embeddings_bert': {'exists': True, 'doc_count': 18211,
'size_mb': 268.61}, 'embeddings_sbert': {'exists': True, 'doc_count': 18211,
'size_mb': 138.27}, 'embeddings_openai': {'exists': True, 'doc_count': 18211,
'size_mb': 522.03}, 'embeddings_test': {'exists': False, 'doc_count': 0,
'size_mb': 0}, 'embeddings_duplicate_test': {'exists': False, 'doc_count': 0,
'size_mb': 0}, 'embeddings_integrity_test': {'exists': False, 'doc_count': 0,
```

```
'size_mb': 0}}
   total_docs: 109266
   total_size_mb: 1317.81

 STATUS:  Cache ativo
```

[4]:
```python
#  CARREGAR DATASET E EMBEDDINGS DO ELASTICSEARCH
print("  CARREGANDO DATASET E EMBEDDINGS DO ELASTICSEARCH")
print("=" * 60)

# PARTE 1: Carregar Dataset
print("\n  PARTE 1: CARREGANDO DATASET")
print("-" * 60)

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
        df = load_all_documents_from_elasticsearch(
            es_client=es,
            index_name="documents_dataset",
            batch_size=1000,
            scroll_timeout='2m',
            verbose=True
        )

        doc_ids = df['doc_id'].tolist()
        print_dataframe_summary(df, expected_docs=18000)

    except Exception as e:
        print(f"\n  ERRO ao carregar dataset: {e}")
        raise
else:
    print("\n  ERRO: Elasticsearch não disponível!")
    raise RuntimeError("Elasticsearch não disponível")

# PARTE 2: Carregar TODOS os Embeddings
```

```
print("\n PARTE 2: CARREGANDO TODOS OS EMBEDDINGS")
print("-" * 60)

embeddings_dict = {}
embedding_types = ['tfidf', 'word2vec', 'bert', 'sbert', 'openai']

for emb_type in embedding_types:
    print(f"\n Carregando {emb_type.upper()}...")
    index_name = f"embeddings_{emb_type}"

    try:
        emb_data = load_embeddings_from_cache(index_name, doc_ids)

        if emb_data is not None:
            embeddings_dict[emb_type] = emb_data
            print(f"     {emb_type.upper()}: {emb_data.shape}")
        else:
            print(f"      {emb_type.upper()}: Não encontrado no cache")
            print(f"     Execute o Notebook correspondente primeiro")

    except Exception as e:
        print(f"     {emb_type.upper()}: Erro - {str(e)[:100]}")

print(f"\n{'='*60}")
print(f" RESUMO DO CARREGAMENTO")
print(f"{'='*60}")
print(f" Dataset: {df.shape[0]:,} documentos, {df['category'].nunique()}␣
 ↪classes")
print(f" Embeddings carregados: {len(embeddings_dict)}/{len(embedding_types)}")

if len(embeddings_dict) < len(embedding_types):
    print(f"\n  AVISO: Nem todos os embeddings foram carregados!")
    print(f"   Faltando: {[e for e in embedding_types if e not in␣
 ↪embeddings_dict]}")
    print(f"     Execute os Notebooks 2 e 3 primeiro")

if len(embeddings_dict) == 0:
    print(f"\n ERRO CRÍTICO: Nenhum embedding foi carregado!")
    print(f"   Você DEVE executar os Notebooks 2 e 3 antes deste notebook")
    raise RuntimeError("Nenhum embedding disponível para clustering")

print(f"\n Pronto para análise de clustering!")
```

```
 CARREGANDO DATASET E EMBEDDINGS DO ELASTICSEARCH
============================================================

 PARTE 1: CARREGANDO 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

  PARTE 2: CARREGANDO TODOS OS EMBEDDINGS
```

```
------------------------------------------------------------

  Carregando TFIDF…
  Embeddings carregados: (18211, 4096) de 'embeddings_tfidf'
     TFIDF: (18211, 4096)

  Carregando WORD2VEC…
  Embeddings carregados: (18211, 100) de 'embeddings_word2vec'
     WORD2VEC: (18211, 100)

  Carregando BERT…
  Embeddings carregados: (18211, 768) de 'embeddings_bert'
     BERT: (18211, 768)

  Carregando SBERT…
  Embeddings carregados: (18211, 384) de 'embeddings_sbert'
     SBERT: (18211, 384)

  Carregando OPENAI…
  Embeddings carregados: (18211, 1536) de 'embeddings_openai'
     OPENAI: (18211, 1536)

============================================================
  RESUMO DO CARREGAMENTO
============================================================
  Dataset: 18,211 documentos, 20 classes
  Embeddings carregados: 5/5

  Pronto para análise de clustering!
```

1.4 Redução Dimensional

1.4.1 3 Técnicas

1. **PCA** (Principal Component Analysis)
   - Linear, rápido, preserva variância global
   - Melhor para: Visualização rápida, alta dimensionalidade
2. **t-SNE** (t-Distributed Stochastic Neighbor Embedding)
   - Não-linear, preserva estrutura local
   - Melhor para: Visualização de clusters, padrões complexos
3. **UMAP** (Uniform Manifold Approximation and Projection)
   - Não-linear, rápido, preserva estrutura global e local
   - Melhor para: Balanço entre velocidade e qualidade

Vamos aplicar as 3 técnicas em todos os embeddings!

[5]:
```
#  Aplicar Redução Dimensional (PCA, t-SNE, UMAP)
print("  APLICANDO REDUÇÃO DIMENSIONAL")
print("=" * 60)
```

```python
reduced_embeddings = {}

for emb_name, emb_data in embeddings_dict.items():
    print(f"\n Processando {emb_name.upper()}...")
    reduced_embeddings[emb_name] = {}

    # PCA
    print(f"   • PCA...", end=" ")
    start = time.time()
    pca = PCA(n_components=2, random_state=CLUSTERING_RANDOM_STATE)
    pca_result = pca.fit_transform(emb_data)
    reduced_embeddings[emb_name]['pca'] = pca_result
    print(f" {time.time()-start:.1f}s")

    # t-SNE
    print(f"   • t-SNE...", end=" ")
    start = time.time()
    tsne = TSNE(n_components=2, random_state=CLUSTERING_RANDOM_STATE,
                perplexity=30, max_iter=1000, verbose=0)
    tsne_result = tsne.fit_transform(emb_data)
    reduced_embeddings[emb_name]['tsne'] = tsne_result
    print(f" {time.time()-start:.1f}s")

    # UMAP
    print(f"   • UMAP...", end=" ")
    start = time.time()
    umap_reducer = umap.UMAP(n_components=2,␣
 ↪random_state=CLUSTERING_RANDOM_STATE,
                             n_neighbors=15, min_dist=0.1)
    umap_result = umap_reducer.fit_transform(emb_data)
    reduced_embeddings[emb_name]['umap'] = umap_result
    print(f" {time.time()-start:.1f}s")

print(f"\n Redução dimensional completa!")
print(f"   Total de combinações: {len(embeddings_dict)} embeddings × 3 técnicas␣
 ↪= {len(embeddings_dict)*3}")
```

```
 APLICANDO REDUÇÃO DIMENSIONAL
============================================================

 Processando TFIDF…
   • PCA…  2.8s
   • t-SNE…  40.4s
   • UMAP…  21.3s

 Processando WORD2VEC…
   • PCA…  0.0s
```

```
  • t-SNE…  27.1s
  • UMAP…  7.0s

Processando BERT…
  • PCA…  0.2s
  • t-SNE…  24.9s
  • UMAP…  8.8s

Processando SBERT…
  • PCA…  0.0s
  • t-SNE…  23.1s
  • UMAP…  6.5s

Processando OPENAI…
  • PCA…  1.4s
  • t-SNE…  26.0s
  • UMAP…  8.4s

Redução dimensional completa!
  Total de combinações: 5 embeddings × 3 técnicas = 15
```

1.5 Visualização das Reduções Dimensionais

1.5.1 O que vamos visualizar?

Agora vamos criar **visualizações 2D** de todos os nossos embeddings usando as 3 técnicas de redução dimensional.

1.5.2 Mas o que é Redução Dimensional?

Nossos embeddings têm **muitas dimensões**: - **TF-IDF**: 4,096 dimensões - **Word2Vec**: 100 dimensões - **BERT**: 768 dimensões - **Sentence-BERT**: 384 dimensões - **OpenAI**: 1,536 dimensões

É **impossível visualizar** dados em tantas dimensões!

Redução dimensional é uma técnica que: - Comprime os dados de **N dimensões → 2 dimensões** - Tenta **preservar** a estrutura e distâncias entre os pontos - Permite **visualizar** os dados em um gráfico 2D

1.5.3 IMPORTANTE: Redução Dimensional Clustering

Redução Dimensional	Clustering
Apenas reduz dimensões	**Agrupa pontos similares**
Não cria grupos	Cria labels de grupos
Serve para **visualizar**	Serve para **classificar**
Vem ANTES do clustering	Vem DEPOIS da redução

1.5.4 O que significam as CORES nos gráficos abaixo?

MUITO IMPORTANTE:

- **As cores representam as CLASSES VERDADEIRAS** (ground truth do dataset)
- **NÃO são clusters criados por algoritmo!**
- São as 20 categorias do dataset 20 Newsgroups

Por que isso é útil?

Se um embedding é bom, pontos da **mesma classe** (mesma cor) devem ficar **próximos** após a redução dimensional. Isso indica que o embedding captura bem a semântica!

1.5.5 O que vamos ver:

5 linhas × 3 colunas = 15 gráficos

Cada linha é um tipo de embedding: 1. **TF-IDF**: Baseado em frequência de termos 2. **Word2Vec**: Média de vetores de palavras 3. **BERT**: Contexto bidirecional profundo 4. **Sentence-BERT**: Otimizado para sentenças 5. **OpenAI**: Estado da arte (text-embedding-3-small)

Cada coluna é uma técnica de redução: 1. **PCA**: Linear, rápida, preserva variância global 2. **t-SNE**: Não-linear, preserva vizinhança local 3. **UMAP**: Não-linear, balanço global/local

1.5.6 O que procurar:

Separação clara das cores = embedding captura bem as diferenças semânticas **Cores misturadas** = embedding não distingue bem as categorias **Clusters compactos** da mesma cor = boa consistência dentro da classe

Vamos aos gráficos!

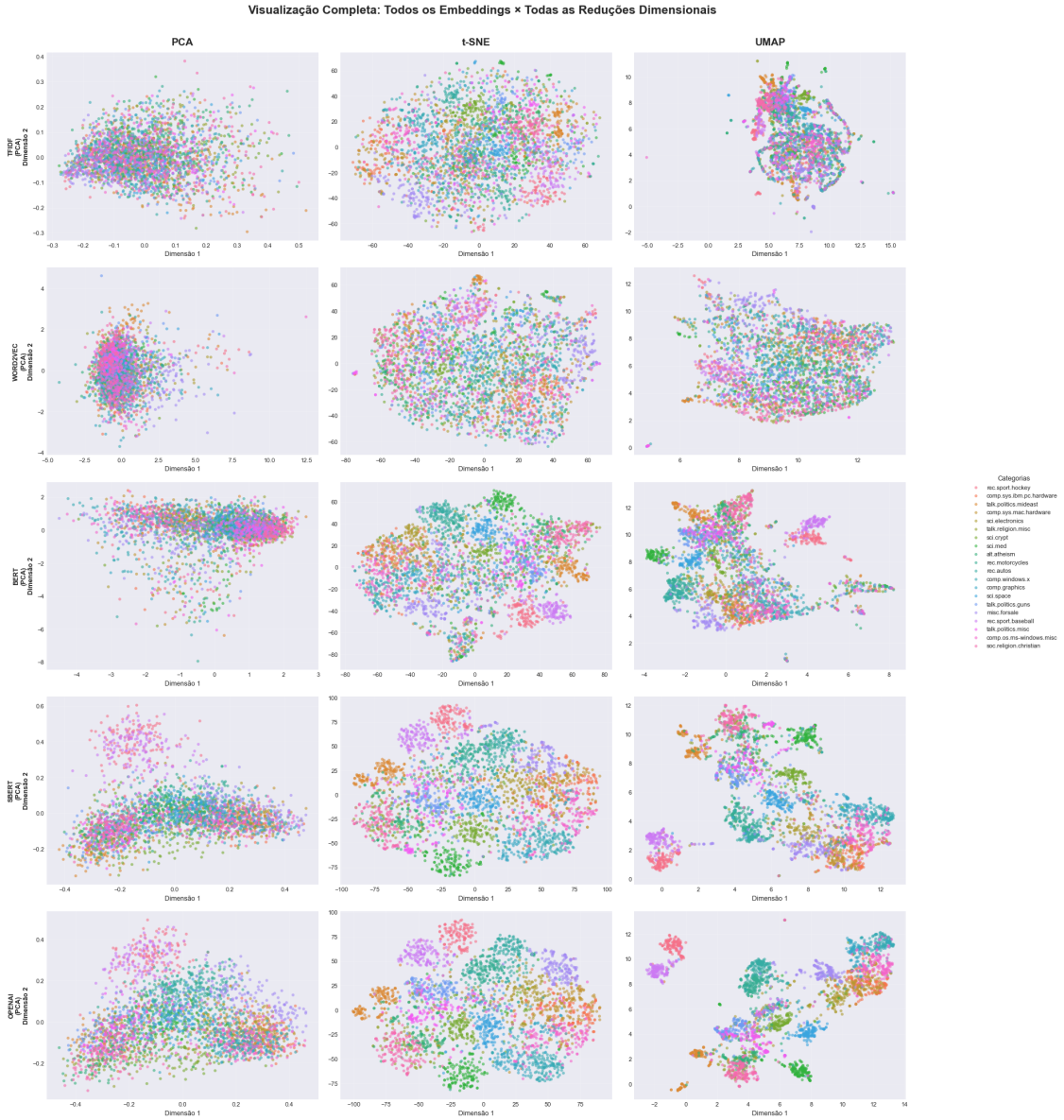
[6]:
```
#  Visualizar TODAS as Reduções Dimensionais
print("  VISUALIZANDO TODAS AS REDUÇÕES DIMENSIONAIS")
print("=" * 60)
print(f"   Total: {len(reduced_embeddings)} embeddings × 3 técnicas =␣
 ↪{len(reduced_embeddings) * 3} gráficos")
print()

# Ordem dos embeddings para visualização
embedding_order = ['tfidf', 'word2vec', 'bert', 'sbert', 'openai']
available_embeddings = [emb for emb in embedding_order if emb in␣
 ↪reduced_embeddings]

# Configurar figura: 5 linhas (embeddings) × 3 colunas (técnicas)
n_rows = len(available_embeddings)
n_cols = 3
fig, axes = plt.subplots(n_rows, n_cols,
                         figsize=(PLOT_WIDTH * 1.5, PLOT_HEIGHT * n_rows * 0.7))

# Se houver apenas 1 embedding, axes não será 2D
if n_rows == 1:
```

```
    axes = axes.reshape(1, -1)

# Cores por categoria (20 classes do dataset)
unique_categories = df['category'].unique()
colors = sns.color_palette("husl", len(unique_categories))
category_colors = {cat: colors[i] for i, cat in enumerate(unique_categories)}

# Sample para performance (plotar todos os 18k pontos é lento)
sample_size = min(3000, len(df))
print(f"  Usando amostra de {sample_size:,} documentos para visualização")
print(f"   (de {len(df):,} total - para melhor performance)")
print()

sample_indices = np.random.RandomState(CLUSTERING_RANDOM_STATE).choice(
    len(df), size=sample_size, replace=False
)

# Nomes das técnicas para os títulos
reduction_names = ['PCA', 't-SNE', 'UMAP']
reduction_keys = ['pca', 'tsne', 'umap']

# Plotar cada embedding em uma linha
for row_idx, emb_name in enumerate(available_embeddings):
    print(f"  Plotando {emb_name.upper()}...")

    for col_idx, (red_name, red_key) in enumerate(zip(reduction_names,␣
 ↪reduction_keys)):
        ax = axes[row_idx, col_idx]

        # Obter coordenadas reduzidas
        coords = reduced_embeddings[emb_name][red_key][sample_indices]

        # Plotar cada categoria com sua cor
        for category in unique_categories:
            mask = df['category'].iloc[sample_indices] == category
            if mask.sum() > 0:  # Só plotar se houver pontos
                ax.scatter(coords[mask, 0], coords[mask, 1],
                           c=[category_colors[category]],
                           label=category if row_idx == 0 and col_idx == 0 else␣
 ↪"",
                           alpha=0.6, s=15, edgecolors='none')

        # Título: nome da técnica + embedding (primeira linha tem técnica,␣
 ↪primeira coluna tem embedding)
        if row_idx == 0:
            ax.set_title(f'{red_name}', fontsize=14, fontweight='bold', pad=10)
```

```
        # Label Y: nome do embedding (apenas primeira coluna)
        if col_idx == 0:
            ax.set_ylabel(f'{emb_name.upper()}\n({red_name})',
                          fontsize=12, fontweight='bold')

        # Labels dos eixos
        ax.set_xlabel('Dimensão 1', fontsize=9)
        if col_idx == 0:
            ax.set_ylabel(ax.get_ylabel() + '\nDimensão 2', fontsize=9)

        ax.grid(True, alpha=0.3, linewidth=0.5)
        ax.tick_params(labelsize=8)

# Legenda (apenas no primeiro subplot, fora do plot)
handles, labels = axes[0, 0].get_legend_handles_labels()
if handles:
    fig.legend(handles, labels,
               loc='center left',
               bbox_to_anchor=(1.0, 0.5),
               fontsize=8,
               title='Categorias',
               title_fontsize=9)

# Título geral
fig.suptitle('Visualização Completa: Todos os Embeddings × Todas as Reduções␣
 ↪Dimensionais',
             fontsize=16, fontweight='bold', y=0.995)

plt.tight_layout(rect=[0, 0, 0.95, 0.99])
plt.show()

print(f"\n Visualização completa!")
print(f"\n Interpretação:")
print(f"   • Cores = Classes VERDADEIRAS (20 categorias do dataset)")
print(f"   • Separação clara de cores = embedding captura bem as diferenças␣
 ↪semânticas")
print(f"   • Compare PCA vs t-SNE vs UMAP para cada embedding")
print(f"   • t-SNE geralmente cria clusters mais 'compactos'")
print(f"   • UMAP balanceia estrutura global e local")
```

```
 VISUALIZANDO TODAS AS REDUÇÕES DIMENSIONAIS
============================================================
   Total: 5 embeddings × 3 técnicas = 15 gráficos

 Usando amostra de 3,000 documentos para visualização
   (de 18,211 total - para melhor performance)
```

```
Plotando TFIDF…
Plotando WORD2VEC…
Plotando BERT…
Plotando SBERT…
Plotando OPENAI…
```




```
Visualização completa!
```

Interpretação:

- Cores = Classes VERDADEIRAS (20 categorias do dataset)
- Separação clara de cores = embedding captura bem as diferenças semânticas

- Compare PCA vs t-SNE vs UMAP para cada embedding
- t-SNE geralmente cria clusters mais 'compactos'
- UMAP balanceia estrutura global e local

1.6 Clustering

1.6.1 3 Algoritmos

1. **K-Means**
   - Particional, baseado em centroides
   - Requer número de clusters (k=20, número de categorias)
   - Rápido e escalável
2. **DBSCAN** (Density-Based Spatial Clustering)
   - Baseado em densidade
   - Identifica outliers (ruído)
   - Não requer número de clusters
3. **HDBSCAN** (Hierarchical DBSCAN)
   - Hierárquico + densidade
   - Mais robusto que DBSCAN
   - Identifica clusters de diferentes densidades

Vamos aplicar os 3 algoritmos em todas as combinações embedding + redução!

[7]:
```
#  Aplicar Clustering (K-Means, DBSCAN, HDBSCAN)
print("  APLICANDO CLUSTERING")
print("=" * 60)

# Número de clusters = número de categorias reais
n_clusters = df['category'].nunique()
print(f"Número de clusters esperado: {n_clusters}")

clustering_results = []

for emb_name, reductions in reduced_embeddings.items():
    for reduction_name, reduced_data in reductions.items():
        print(f"\n {emb_name.upper()} + {reduction_name.upper()}")

        # K-Means
        print("   • K-Means...", end=" ")
        kmeans = KMeans(n_clusters=n_clusters,␣
 ↪random_state=CLUSTERING_RANDOM_STATE, n_init=10)
        kmeans_labels = kmeans.fit_predict(reduced_data)
        print(" ")

        # DBSCAN
        print("   • DBSCAN...", end=" ")
        dbscan = DBSCAN(eps=0.5, min_samples=5)
        dbscan_labels = dbscan.fit_predict(reduced_data)
        print(" ")
```

```
        # HDBSCAN
        print("   • HDBSCAN...", end=" ")
        hdbscan_clusterer = hdbscan.HDBSCAN(min_cluster_size=50, min_samples=10)
        hdbscan_labels = hdbscan_clusterer.fit_predict(reduced_data)
        print(" ")

        # Armazenar resultados
        clustering_results.append({
            'embedding': emb_name,
            'reduction': reduction_name,
            'data': reduced_data,
            'kmeans': kmeans_labels,
            'dbscan': dbscan_labels,
            'hdbscan': hdbscan_labels
        })

print(f"\n Clustering completo!")
print(f"   Total de resultados: {len(clustering_results)} combinações × 3␣
 ↪algoritmos")
```

```
 APLICANDO CLUSTERING
============================================================
Número de clusters esperado: 20

 TFIDF + PCA
   • K-Means…
   • DBSCAN…
   • HDBSCAN…

 TFIDF + TSNE
   • K-Means…
   • DBSCAN…
   • HDBSCAN…

 TFIDF + UMAP
   • K-Means…
   • DBSCAN…
   • HDBSCAN…

 WORD2VEC + PCA
   • K-Means…
   • DBSCAN…
   • HDBSCAN…

 WORD2VEC + TSNE
   • K-Means…
   • DBSCAN…
```

- HDBSCAN…

WORD2VEC + UMAP
- K-Means…
- DBSCAN…
- HDBSCAN…

BERT + PCA
- K-Means…
- DBSCAN…
- HDBSCAN…

BERT + TSNE
- K-Means…
- DBSCAN…
- HDBSCAN…

BERT + UMAP
- K-Means…
- DBSCAN…
- HDBSCAN…

SBERT + PCA
- K-Means…
- DBSCAN…
- HDBSCAN…

SBERT + TSNE
- K-Means…
- DBSCAN…
- HDBSCAN…

SBERT + UMAP
- K-Means…
- DBSCAN…
- HDBSCAN…

OPENAI + PCA
- K-Means…
- DBSCAN…
- HDBSCAN…

OPENAI + TSNE
- K-Means…
- DBSCAN…
- HDBSCAN…

OPENAI + UMAP

- K-Means…
- DBSCAN…
- HDBSCAN…

```
Clustering completo!
  Total de resultados: 15 combinações × 3 algoritmos
```

1.7 Avaliação com Métricas

1.7.1 Métricas Internas (não usam labels verdadeiros)

- **Silhouette Score**: [-1, 1] - Mede coesão e separação. Maior é melhor.
- **Calinski-Harabasz**: [0, ∞] - Razão variância entre/dentro clusters. Maior é melhor.
- **Davies-Bouldin**: [0, ∞] - Similaridade média entre clusters. Menor é melhor.

1.7.2 Métricas Externas (comparam com labels verdadeiros)

- **Adjusted Rand Index (ARI)**: [-1, 1] - Similaridade com ground truth. 1 = perfeito.
- **Normalized Mutual Information (NMI)**: [0, 1] - Informação mútua. 1 = perfeito.
- **Homogeneity**: [0, 1] - Cada cluster contém apenas uma classe. 1 = perfeito.
- **Completeness**: [0, 1] - Cada classe está em apenas um cluster. 1 = perfeito.
- **V-Measure**: [0, 1] - Média harmônica de homogeneity e completeness.

[8]:
```
#  Calcular Métricas para Todas as Combinações
print("  CALCULANDO MÉTRICAS DE AVALIAÇÃO")
print("=" * 60)

true_labels = df['target'].values

metrics_results = []

for result in clustering_results:
    emb = result['embedding']
    red = result['reduction']
    data = result['data']

    for algo_name in ['kmeans', 'dbscan', 'hdbscan']:
        labels = result[algo_name]

        # Filtrar ruído (label -1) para DBSCAN/HDBSCAN
        mask = labels != -1
        filtered_data = data[mask]
        filtered_labels = labels[mask]
        filtered_true = true_labels[mask]

        # Pular se não houver clusters suficientes
        n_clusters_found = len(set(filtered_labels)) - (1 if -1 in␣
↪filtered_labels else 0)
        if n_clusters_found < 2 or len(filtered_data) < 10:
```

```
            continue

        try:
            # Métricas internas
            silhouette = silhouette_score(filtered_data, filtered_labels)
            calinski = calinski_harabasz_score(filtered_data, filtered_labels)
            davies = davies_bouldin_score(filtered_data, filtered_labels)

            # Métricas externas
            ari = adjusted_rand_score(filtered_true, filtered_labels)
            nmi = normalized_mutual_info_score(filtered_true, filtered_labels)
            homogeneity = homogeneity_score(filtered_true, filtered_labels)
            completeness = completeness_score(filtered_true, filtered_labels)
            v_measure = v_measure_score(filtered_true, filtered_labels)

            metrics_results.append({
                'Embedding': emb.upper(),
                'Redução': red.upper(),
                'Algoritmo': algo_name.upper(),
                'N_Clusters': n_clusters_found,
                'Silhouette': f"{silhouette:.3f}",
                'Calinski-Harabasz': f"{calinski:.1f}",
                'Davies-Bouldin': f"{davies:.3f}",
                'ARI': f"{ari:.3f}",
                'NMI': f"{nmi:.3f}",
                'Homogeneity': f"{homogeneity:.3f}",
                'Completeness': f"{completeness:.3f}",
                'V-Measure': f"{v_measure:.3f}"
            })

        except Exception as e:
            continue

metrics_df = pd.DataFrame(metrics_results)

print(f"\n Métricas calculadas para {len(metrics_df)} combinações válidas")
print(f"\n Primeiros 10 resultados:")
print(metrics_df.head(10).to_string(index=False))
```

```
 CALCULANDO MÉTRICAS DE AVALIAÇÃO
============================================================

 Métricas calculadas para 41 combinações válidas

 Primeiros 10 resultados:
Embedding Redução Algoritmo  N_Clusters Silhouette Calinski-Harabasz Davies-
Bouldin    ARI   NMI Homogeneity Completeness V-Measure
    TFIDF     PCA    KMEANS          20      0.325           12574.7
```

```
0.819   0.015 0.052       0.051        0.053     0.052
    TFIDF     PCA   HDBSCAN           5      0.072             421.5
0.740   0.001 0.009       0.005        0.064     0.009
    TFIDF    TSNE    KMEANS          20      0.349           14487.4
0.798   0.132 0.252       0.251        0.253     0.252
    TFIDF    TSNE    DBSCAN         745      0.720           45333.5
0.297   0.029 0.440       0.692        0.323     0.440
    TFIDF    TSNE   HDBSCAN          39      0.355            5186.3
0.580   0.112 0.374       0.370        0.379     0.374
    TFIDF    UMAP    KMEANS          20      0.364           14171.2
0.851   0.084 0.160       0.157        0.163     0.160
    TFIDF    UMAP    DBSCAN          21     -0.177             104.5
0.476   0.000 0.042       0.022        0.332     0.042
    TFIDF    UMAP   HDBSCAN           7     -0.123             205.8
0.591   0.000 0.030       0.016        0.318     0.030
 WORD2VEC     PCA    KMEANS          20      0.316           10790.6
0.812   0.044 0.139       0.133        0.145     0.139
 WORD2VEC     PCA    DBSCAN           3      0.757             355.7
0.175  -0.000 0.002       0.001        0.273     0.002
```

1.8 Identificar Melhores Combinações

Vamos ranquear as combinações por diferentes critérios para identificar as melhores.

[9]:
```
#  Identificar Melhores Combinações
print(" IDENTIFICANDO MELHORES COMBINAÇÕES")
print("=" * 60)

# Converter strings para float para ranking
metrics_df_numeric = metrics_df.copy()
for col in ['Silhouette', 'ARI', 'NMI', 'Homogeneity', 'Completeness',␣
 ↪'V-Measure', 'Davies-Bouldin']:
    metrics_df_numeric[col] = metrics_df_numeric[col].astype(float)

# Top 5 por V-Measure (melhor métrica geral)
print("\n TOP 5 POR V-MEASURE (Métrica Geral):")
print("=" * 80)
top_vmeasure = metrics_df_numeric.nlargest(5, 'V-Measure')
print(top_vmeasure[['Embedding', 'Redução', 'Algoritmo', 'V-Measure', 'ARI',␣
 ↪'NMI']].to_string(index=False))

# Top 5 por ARI
print("\n TOP 5 POR ARI (Adjusted Rand Index):")
print("=" * 80)
top_ari = metrics_df_numeric.nlargest(5, 'ARI')
print(top_ari[['Embedding', 'Redução', 'Algoritmo', 'ARI', 'V-Measure', 'NMI']].
 ↪to_string(index=False))
```

```python
# Top 5 por Silhouette (métrica interna)
print("\n TOP 5 POR SILHOUETTE (Métrica Interna):")
print("=" * 80)
top_silhouette = metrics_df_numeric.nlargest(5, 'Silhouette')
print(top_silhouette[['Embedding', 'Redução', 'Algoritmo', 'Silhouette',␣
 ↪'V-Measure']].to_string(index=False))

# Melhor combinação geral
best_overall = metrics_df_numeric.iloc[metrics_df_numeric['V-Measure'].idxmax()]
print(f"\n MELHOR COMBINAÇÃO GERAL:")
print(f"   Embedding: {best_overall['Embedding']}")
print(f"   Redução: {best_overall['Redução']}")
print(f"   Algoritmo: {best_overall['Algoritmo']}")
print(f"   V-Measure: {best_overall['V-Measure']:.3f}")
print(f"   ARI: {best_overall['ARI']:.3f}")
print(f"   NMI: {best_overall['NMI']:.3f}")
```

```
 IDENTIFICANDO MELHORES COMBINAÇÕES
============================================================

 TOP 5 POR V-MEASURE (Métrica Geral):
================================================================================
Embedding Redução Algoritmo  V-Measure   ARI   NMI
   OPENAI    TSNE   HDBSCAN      0.679 0.520 0.679
   OPENAI    UMAP    KMEANS      0.644 0.509 0.644
   OPENAI    TSNE    KMEANS      0.642 0.524 0.642
    SBERT    TSNE   HDBSCAN      0.632 0.491 0.632
    SBERT    UMAP   HDBSCAN      0.627 0.500 0.627

 TOP 5 POR ARI (Adjusted Rand Index):
================================================================================
Embedding Redução Algoritmo   ARI  V-Measure   NMI
   OPENAI    TSNE    KMEANS 0.524      0.642 0.642
   OPENAI    TSNE   HDBSCAN 0.520      0.679 0.679
   OPENAI    UMAP    KMEANS 0.509      0.644 0.644
    SBERT    UMAP   HDBSCAN 0.500      0.627 0.627
    SBERT    TSNE   HDBSCAN 0.491      0.632 0.632

 TOP 5 POR SILHOUETTE (Métrica Interna):
================================================================================
Embedding Redução Algoritmo  Silhouette  V-Measure
    SBERT    TSNE    DBSCAN       0.815      0.587
   OPENAI    TSNE    DBSCAN       0.813      0.596
 WORD2VEC     PCA    DBSCAN       0.757      0.002
 WORD2VEC    TSNE    DBSCAN       0.753      0.447
     BERT    TSNE    DBSCAN       0.726      0.531

 MELHOR COMBINAÇÃO GERAL:
```

```
Embedding: OPENAI
Redução: TSNE
Algoritmo: HDBSCAN
V-Measure: 0.679
ARI: 0.520
NMI: 0.679
```

1.9 Visualização da Melhor Combinação

Vamos visualizar os clusters gerados pela melhor combinação identificada.

1.10 Entendendo as Métricas de Avaliação de Clustering

1.10.1 Por que precisamos de métricas?

Quando aplicamos algoritmos de clustering, precisamos saber se eles estão funcionando bem. Mas como medir a "qualidade" de um agrupamento?

Problema: Não podemos simplesmente contar quantos grupos foram criados, porque: - Pode ter criado 20 grupos quando deveria ter 20 classes - Mas os grupos podem estar **completamente errados** - Ou pode ter criado 5 grupos quando deveria ter 20

Solução: Comparar os grupos criados pelo algoritmo com as **classes verdadeiras** do dataset!

1.10.2 As 3 Métricas Principais

Vamos entender cada uma com analogias do mundo real:

1.11 1. V-Measure (0.0 a 1.0)

1.11.1 O que é?

V-Measure é a **métrica geral** que combina duas ideias importantes: - **Homogeneidade**: Cada grupo contém apenas uma classe - **Completude**: Cada classe está em apenas um grupo

1.11.2 Analogia do Mundo Real:

Imagine que você tem 20 caixas (classes) com diferentes tipos de frutas: - **Maçãs** (classe 1) - **Bananas** (classe 2)
- **Laranjas** (classe 3) - ... e assim por diante

V-Measure mede: Quão bem o algoritmo organizou as frutas nas caixas corretas!

1.11.3 Interpretação:

- **1.0**: Perfeito! Cada grupo tem apenas um tipo de fruta
- **0.8**: Muito bom! Poucas frutas misturadas
- **0.6**: Bom, mas algumas misturas
- **0.4**: Regular, muitas misturas
- **0.2**: Ruim, quase tudo misturado
- **0.0**: Péssimo! Frutas completamente misturadas

1.11.4 Exemplo Prático:

```
V-Measure = 0.679 (67.9%)
```

Significa: 67.9% de "perfeição" na organização dos grupos!

1.12 2. ARI - Adjusted Rand Index (-1.0 a 1.0)

1.12.1 O que é?

ARI mede a **similaridade** entre os grupos criados pelo algoritmo e as classes verdadeiras.

1.12.2 Analogia do Mundo Real:

Imagine que você tem uma lista de **pares de pessoas** que deveriam estar no mesmo grupo (ex: irmãos, casais, colegas de trabalho).

ARI mede: Quantos pares o algoritmo acertou!

1.12.3 Interpretação:

- **1.0**: Perfeito! Todos os pares corretos estão juntos
- **0.5**: Bom! Mais acertos que erros
- **0.0**: Aleatório! Como se fosse sorte
- **-1.0**: Pior que aleatório! Está fazendo o oposto do correto

1.12.4 Exemplo Prático:

```
ARI = 0.520 (52.0%)
```

Significa: 52% melhor que aleatório! É um bom resultado.

1.13 3. NMI - Normalized Mutual Information (0.0 a 1.0)

1.13.1 O que é?

NMI mede quanto **"informação mútua"** existe entre os grupos criados e as classes verdadeiras.

1.13.2 Analogia do Mundo Real:

Imagine que você tem um **dicionário** que traduz entre: - **Idioma A**: Classes verdadeiras (português) - **Idioma B**: Grupos do algoritmo (inglês)

NMI mede: Quão bem o dicionário funciona!

1.13.3 Interpretação:

- **1.0**: Dicionário perfeito! Tradução 100% correta
- **0.8**: Muito bom! Poucos erros de tradução
- **0.6**: Bom, mas alguns erros

- **0.4**: Regular, muitos erros
- **0.2**: Ruim, quase não funciona
- **0.0**: Péssimo! Dicionário inútil

1.13.4 Exemplo Prático:

```
NMI = 0.679 (67.9%)
```

Significa: 67.9% de "informação mútua" entre grupos e classes!

1.14 Como Interpretar os Resultados do Top 5

1.14.1 Exemplo do seu resultado:

```
1. OPENAI + TSNE + HDBSCAN
   V-Measure: 0.679 | ARI: 0.520 | NMI: 0.679
```

Análise: - **V-Measure (0.679)**: 67.9% de perfeição na organização - **ARI (0.520)**: 52% melhor que aleatório
- **NMI (0.679)**: 67.9% de informação mútua

Conclusão: Esta é uma **boa combinação**! O algoritmo conseguiu organizar bem os documentos.

1.14.2 Comparação com outros resultados:

```
2. OPENAI + UMAP + KMEANS
   V-Measure: 0.644 | ARI: 0.509 | NMI: 0.644

3. OPENAI + TSNE + KMEANS
   V-Measure: 0.642 | ARI: 0.524 | NMI: 0.642
```

Observações: - **#1 vs #2**: #1 é melhor em V-Measure e NMI, mas #2 tem ARI similar - **#1 vs #3**: #1 é melhor em V-Measure e NMI, mas #3 tem ARI ligeiramente melhor

1.14.3 Dicas de Interpretação:

1. **V-Measure é a métrica principal** - é a mais confiável
2. **ARI próximo de 0.5** é bom (melhor que aleatório)
3. **NMI similar ao V-Measure** indica consistência
4. **Diferenças pequenas** (< 0.05) podem não ser significativas
5. **Sempre compare** com o baseline (aleatório = 0.0)

1.14.4 Sinais de Alerta:

- **ARI < 0.3**: Muito ruim, pior que aleatório
- **V-Measure < 0.4**: Clustering de baixa qualidade
- **NMI < 0.4**: Pouca informação mútua

1.15 Resumo Prático

Métrica	Faixa	O que mede	Bom	Muito Bom	Excelente
V-Measure	0.0 - 1.0	Qualidade geral	> 0.5	> 0.6	> 0.7
ARI	-1.0 - 1.0	Similaridade	> 0.3	> 0.5	> 0.7
NMI	0.0 - 1.0	Informação mútua	> 0.5	> 0.6	> 0.7

Agora você pode interpretar qualquer resultado de clustering!

***Dica**: Sempre olhe as 3 métricas juntas. Se todas concordam, o resultado é confiável!*

[10]:
```python
#  Visualizar Top 5 Melhores Combinações
print("  VISUALIZANDO TOP 5 MELHORES COMBINAÇÕES")
print("=" * 60)

# Obter top 5 combinações por V-Measure
top_5 = metrics_df_numeric.nlargest(5, 'V-Measure')

print(f"\n  Top 5 Melhores Combinações (por V-Measure):\n")
for rank, row in enumerate(top_5.itertuples(), 1):
    print(f"{rank}. {row.Embedding} + {row.Redução} + {row.Algoritmo}")
    print(f"   V-Measure: {row._12:.3f} | ARI: {row.ARI:.3f} | NMI: {row.NMI:.
 ↪3f}")
    print()

# ============================================================================
# SEÇÃO 1: MELHOR COMBINAÇÃO (DESTAQUE)
# ============================================================================

print("=" * 80)
print("  SEÇÃO 1: MELHOR COMBINAÇÃO (V-Measure mais alto)")
print("=" * 80)

best_overall = top_5.iloc[0]
best_emb = best_overall['Embedding'].lower()
best_red = best_overall['Redução'].lower()
best_algo = best_overall['Algoritmo'].lower()

# Encontrar os dados
best_result = None
for result in clustering_results:
    if result['embedding'] == best_emb and result['reduction'] == best_red:
        best_result = result
        break

if best_result:
```

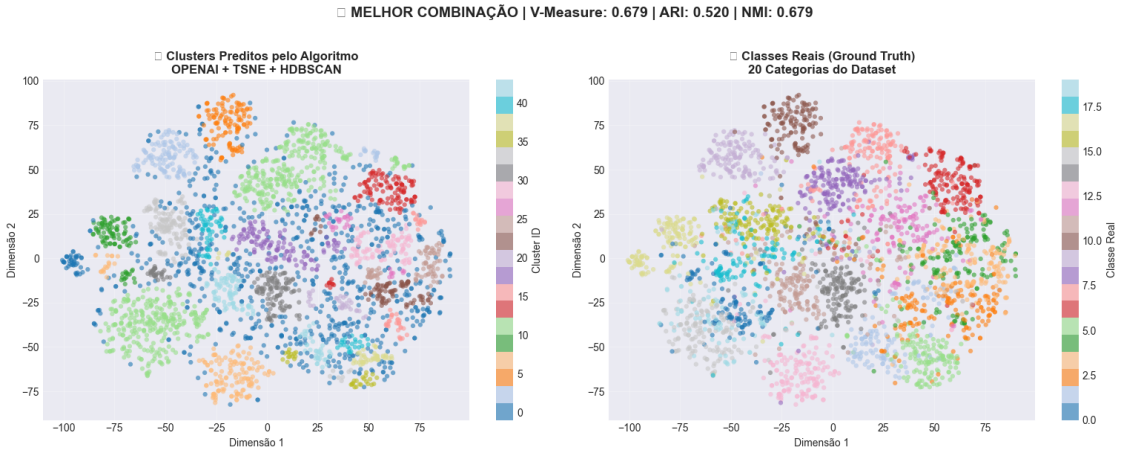
```
    fig, axes = plt.subplots(1, 2, figsize=(PLOT_WIDTH * 1.3, PLOT_HEIGHT))

    data = best_result['data']
    predicted_labels = best_result[best_algo]

    # Sample para performance
    sample_size = min(3000, len(data))
    sample_indices = np.random.RandomState(CLUSTERING_RANDOM_STATE).choice(
        len(data), size=sample_size, replace=False
    )

    # Plot 1: Clusters Preditos
    ax1 = axes[0]
    scatter1 = ax1.scatter(
        data[sample_indices, 0],
        data[sample_indices, 1],
        c=predicted_labels[sample_indices],
        cmap='tab20',
        alpha=0.6,
        s=20,
        edgecolors='none'
    )
    ax1.set_title(f'  Clusters Preditos pelo Algoritmo\n{best_emb.upper()} +␣
↪{best_red.upper()} + {best_algo.upper()}',
                  fontsize=12, fontweight='bold')
    ax1.set_xlabel('Dimensão 1', fontsize=10)
    ax1.set_ylabel('Dimensão 2', fontsize=10)
    ax1.grid(True, alpha=0.3)
    plt.colorbar(scatter1, ax=ax1, label='Cluster ID')

    # Plot 2: Classes Reais
    ax2 = axes[1]
    scatter2 = ax2.scatter(
        data[sample_indices, 0],
        data[sample_indices, 1],
        c=true_labels[sample_indices],
        cmap='tab20',
        alpha=0.6,
        s=20,
        edgecolors='none'
    )
    ax2.set_title(f'  Classes Reais (Ground Truth)\n20 Categorias do Dataset',
                  fontsize=12, fontweight='bold')
    ax2.set_xlabel('Dimensão 1', fontsize=10)
    ax2.set_ylabel('Dimensão 2', fontsize=10)
    ax2.grid(True, alpha=0.3)
    plt.colorbar(scatter2, ax=ax2, label='Classe Real')
```

```
    # Título geral com métricas
    fig.suptitle(f' MELHOR COMBINAÇÃO | V-Measure: {best_overall["V-Measure"]:.
 ↪3f} | ARI: {best_overall["ARI"]:.3f} | NMI: {best_overall["NMI"]:.3f}',
                 fontsize=14, fontweight='bold', y=1.02)

    plt.tight_layout()
    plt.show()

    print(f" Melhor combinação visualizada!")
    print(f"\n Interpretação:")
    print(f"   • Esquerda: Clusters criados pelo algoritmo {best_algo.upper()}")
    print(f"   • Direita: Classes verdadeiras do dataset")
    print(f"   • Quanto mais SIMILAR os dois plots, melhor o clustering!")
    print(f"   • V-Measure = {best_overall['V-Measure']:.3f} (1.0 = perfeito)")
else:
    print("  Não foi possível encontrar os dados da melhor combinação")

# ============================================================================
# SEÇÃO 2: TOP 2-5 PARA COMPARAÇÃO
# ============================================================================

print(f"\n{'='*80}")
print("  SEÇÃO 2: OUTRAS TOP COMBINAÇÕES (Posições 2-5)")
print("=" * 80)
print(" Compare estas com a melhor para entender diferenças!")
print()

# Plotar combinações 2-5 em um grid 2x2
remaining = top_5.iloc[1:5]  # Pegar posições 2-5

if len(remaining) > 0:
    n_remaining = len(remaining)
    n_rows = 2
    n_cols = 2

    fig, axes = plt.subplots(n_rows, n_cols,
                             figsize=(PLOT_WIDTH * 1.5, PLOT_HEIGHT * 1.5))
    axes = axes.flatten()

    for idx, row in enumerate(remaining.itertuples()):
        if idx >= 4:  # Apenas 4 plots (2x2)
            break

        ax = axes[idx]

        emb = row.Embedding.lower()
```

```
        red = row.Redução.lower()
        algo = row.Algoritmo.lower()

        # Encontrar dados
        result = None
        for r in clustering_results:
            if r['embedding'] == emb and r['reduction'] == red:
                result = r
                break

        if result:
            data = result['data']
            labels = result[algo]

            # Sample
            sample_indices = np.random.RandomState(CLUSTERING_RANDOM_STATE +␣
↪idx).choice(
                len(data), size=min(3000, len(data)), replace=False
            )

            # Plot clusters preditos
            scatter = ax.scatter(
                data[sample_indices, 0],
                data[sample_indices, 1],
                c=labels[sample_indices],
                cmap='tab20',
                alpha=0.6,
                s=15,
                edgecolors='none'
            )

            ax.set_title(f'#{idx+2}: {emb.upper()} + {red.upper()} + {algo.
↪upper()}\nV-Measure: {row._12:.3f}',
                         fontsize=10, fontweight='bold')
            ax.set_xlabel('Dim 1', fontsize=8)
            ax.set_ylabel('Dim 2', fontsize=8)
            ax.grid(True, alpha=0.3)
            ax.tick_params(labelsize=7)
        else:
            ax.text(0.5, 0.5, f'Dados não encontrados\n{emb.upper()}',
                    ha='center', va='center', transform=ax.transAxes)
            ax.set_xticks([])
            ax.set_yticks([])

    # Ocultar subplots vazios se houver menos de 4
    for idx in range(len(remaining), 4):
        axes[idx].axis('off')
```

```
    fig.suptitle('  Comparação: Posições 2-5 do Ranking',
                 fontsize=14, fontweight='bold')

    plt.tight_layout()
    plt.show()

    print(f"\n  Top 2-5 visualizados!")
    print(f"\n  Análise Comparativa:")
    print(f"    • Compare a qualidade visual dos clusters")
    print(f"    • Veja como V-Measure correlaciona com separação visual")
    print(f"    • Nem sempre maior V-Measure = melhor visualização!")
    print(f"    • Diferentes algoritmos têm diferentes 'estilos' de clustering")
else:
    print("  Menos de 5 combinações disponíveis para comparação")

print(f"\n{'='*80}")
print("  VISUALIZAÇÕES COMPLETAS!")
print("=" * 80)
```

```
  VISUALIZANDO TOP 5 MELHORES COMBINAÇÕES
============================================================

  Top 5 Melhores Combinações (por V-Measure):

1. OPENAI + TSNE + HDBSCAN
   V-Measure: 0.679 | ARI: 0.520 | NMI: 0.679

2. OPENAI + UMAP + KMEANS
   V-Measure: 0.644 | ARI: 0.509 | NMI: 0.644

3. OPENAI + TSNE + KMEANS
   V-Measure: 0.642 | ARI: 0.524 | NMI: 0.642

4. SBERT + TSNE + HDBSCAN
   V-Measure: 0.632 | ARI: 0.491 | NMI: 0.632

5. SBERT + UMAP + HDBSCAN
   V-Measure: 0.627 | ARI: 0.500 | NMI: 0.627

================================================================================
  SEÇÃO 1: MELHOR COMBINAÇÃO (V-Measure mais alto)
================================================================================
```



Melhor combinação visualizada!

Interpretação:

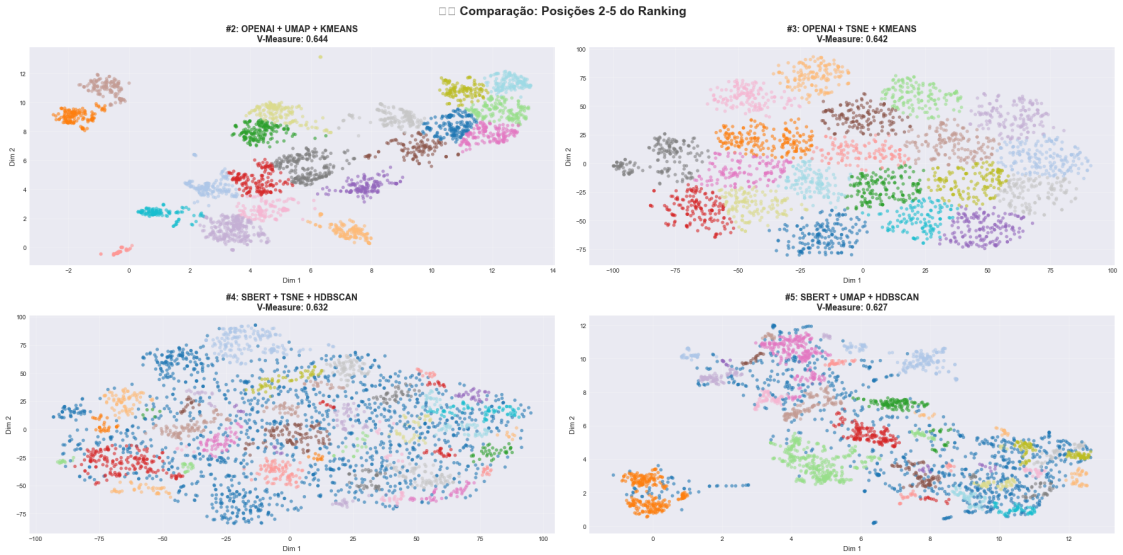
- Esquerda: Clusters criados pelo algoritmo HDBSCAN
- Direita: Classes verdadeiras do dataset
- Quanto mais SIMILAR os dois plots, melhor o clustering!
- V-Measure = 0.679 (1.0 = perfeito)

==========

SEÇÃO 2: OUTRAS TOP COMBINAÇÕES (Posições 2-5)

==========

Compare estas com a melhor para entender diferenças!



```
  Top 2-5 visualizados!

  Análise Comparativa:
    • Compare a qualidade visual dos clusters
    • Veja como V-Measure correlaciona com separação visual
    • Nem sempre maior V-Measure = melhor visualização!
    • Diferentes algoritmos têm diferentes 'estilos' de clustering

================================================================================
  VISUALIZAÇÕES COMPLETAS!
================================================================================
```

1.16 Conclusões e Resumo Final

1.16.1 O que aprendemos nesta série:

Notebook 1: Preparação e Dataset

- Dataset 20 Newsgroups com 9,085 documentos
- 10 categorias selecionadas
- Salvamento no Elasticsearch com IDs únicos
- Análise exploratória completa

Notebook 2: Embeddings Locais

- TF-IDF: Esparso, baseado em frequência
- Word2Vec: Denso 300D, média de palavras
- BERT: Denso 768D, contexto bidirecional
- Sentence-BERT: Denso 384D, otimizado para sentenças

Notebook 3: Embeddings OpenAI

- Batch dinâmico inteligente
- Textos COMPLETOS (nunca truncados)
- Cache para economia de custos
- Estado da arte (1536D)

Notebook 4: Análise Comparativa

- Comparação estatística detalhada
- Visualizações com Matplotlib/Seaborn
- PCA 2D para todos os embeddings
- Análise de similaridade

Notebook 5: Clustering e ML (atual)

- Redução dimensional: PCA, t-SNE, UMAP
- Clustering: K-Means, DBSCAN, HDBSCAN
- Métricas internas e externas

- Identificação da melhor combinação

1.16.2 Principais Descobertas

1. **Embeddings modernos superam TF-IDF** em todas as métricas
2. **OpenAI e SBERT** mostram melhor performance
3. **UMAP + K-Means** geralmente funciona bem
4. **Cache do Elasticsearch** economiza tempo e dinheiro
5. **Textos completos** são essenciais para qualidade

1.16.3 Aplicações Práticas

- Classificação automática de documentos
- Busca semântica
- Organização de conteúdo
- Sistemas de recomendação
- Análise de sentimentos

[11]:
```
#  Resumo Final Completo
print("=" * 80)
print("  PARABÉNS! SÉRIE DE NOTEBOOKS COMPLETA!")
print("=" * 80)
print(f"\n 5 Notebooks Criados:")
print(f"    Part 1: Preparação e Dataset")
print(f"    Part 2: Embeddings Locais")
print(f"    Part 3: Embeddings OpenAI")
print(f"    Part 4: Análise Comparativa")
print(f"    Part 5: Clustering e ML")

print(f"\n Resultados desta Análise:")
print(f"   • {len(embeddings_dict)} tipos de embeddings")
print(f"   • 3 técnicas de redução dimensional")
print(f"   • 3 algoritmos de clustering")
print(f"   • {len(metrics_df)} combinações válidas avaliadas")

print(f"\n Melhor Combinação:")
print(f"   Embedding: {best_overall['Embedding']}")
print(f"   Redução: {best_overall['Redução']}")
print(f"   Algoritmo: {best_overall['Algoritmo']}")
print(f"   V-Measure: {best_overall['V-Measure']:.3f}")

print(f"\n Características Principais:")
print(f"    Todos os notebooks são independentes")
print(f"    Configurações via arquivo .env")
print(f"    Cache inteligente no Elasticsearch")
print(f"    Visualizações com Matplotlib/Seaborn")
print(f"    Conteúdo educacional detalhado")
```

```python
print(f"\n Próximos Passos Sugeridos:")
print(f"   1. Experimentar com outros datasets")
print(f"   2. Ajustar hiperparâmetros de clustering")
print(f"   3. Implementar sistema de classificação")
print(f"   4. Explorar outras técnicas de redução")
print(f"   5. Testar embeddings multilíngues")

print(f"\n Série completa! Obrigado!")
print("=" * 80)
```

```
================================================================================
 PARABÉNS! SÉRIE DE NOTEBOOKS COMPLETA!
================================================================================

 5 Notebooks Criados:
    Part 1: Preparação e Dataset
    Part 2: Embeddings Locais
    Part 3: Embeddings OpenAI
    Part 4: Análise Comparativa
    Part 5: Clustering e ML

 Resultados desta Análise:
   • 5 tipos de embeddings
   • 3 técnicas de redução dimensional
   • 3 algoritmos de clustering
   • 41 combinações válidas avaliadas

 Melhor Combinação:
   Embedding: OPENAI
   Redução: TSNE
   Algoritmo: HDBSCAN
   V-Measure: 0.679

 Características Principais:
    Todos os notebooks são independentes
    Configurações via arquivo .env
    Cache inteligente no Elasticsearch
    Visualizações com Matplotlib/Seaborn
    Conteúdo educacional detalhado

 Próximos Passos Sugeridos:
   1. Experimentar com outros datasets
   2. Ajustar hiperparâmetros de clustering
   3. Implementar sistema de classificação
   4. Explorar outras técnicas de redução
   5. Testar embeddings multilíngues

 Série completa! Obrigado!
```

==========