Capítulo 10 – Similaridade de Cosseno e Busca Semântica

Aléssio Miranda Jr.

Setembro - 2026/2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

O que você verá neste capítulo

Agora que sabemos converter palavras em vetores densos, a pergunta natural é: como medimos o quão "parecidos" dois textos são? E, mais importante, como usamos essa medida para construir sistemas que buscam documentos pelo *significado*, não pela presença literal de palavras-chave? Neste capítulo, formalizaremos a **Similaridade de Cosseno** como a métrica geométrica padrão para comparação de vetores em espaços de alta dimensionalidade, demonstraremos por que a distância euclidiana clássica falha em espaços de Embeddings, construiremos conceitualmente um **Motor de Busca Semântico**, e introduziremos o **RAG (Retrieval-Augmented Generation)** — a arquitetura que dominou a IA corporativa. Ao final, você entenderá cada multiplicação e norma por trás do buscador que alimenta o ChatGPT Enterprise.

1 Introdução: Medir Distâncias no Espaço Semântico

Na aula anterior, transformamos cada palavra e cada documento em um vetor numérico denso de d dimensões (tipicamente 300 a 1536). Com esses vetores em mãos, a tarefa fundamental da busca semântica se reduz a uma pergunta puramente geométrica: *qual vetor do meu banco de dados está mais próximo do vetor da consulta do usuário?*

Para responder a essa pergunta, precisamos de uma **métrica de similaridade** — uma função matemática que receba dois vetores e retorne um esca-

lar indicando o grau de “parentesco semântico” entre eles. A escolha dessa métrica não é trivial: ela determina a qualidade de todo o sistema de busca construído sobre ela.

2 Por que a Distância Euclidiana Falha?

A primeira intuição seria usar a distância euclidiana que aprendemos na geometria analítica — afinal, é a “distância natural” que vocês conhecem desde o Cálculo I:

$$d(\mathbf{a}, \mathbf{b}) = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2} = ||\mathbf{a} - \mathbf{b}||_2 \tag{1}$$

Contudo, em espaços de alta dimensionalidade (300+ dimensões), a distância euclidiana sofre de dois problemas graves.

O primeiro é a **Maldição da Dimensionalidade** (*Curse of Dimensionality*): à medida que o número de dimensões cresce, as distâncias entre quaisquer dois pontos convergem para valores muito similares. Em um espaço de 2 dimensões, a razão entre a maior e a menor distância entre pontos aleatórios é grande (fácil distinguir vizinhos próximos de pontos distantes). Em 300 dimensões, essa razão tende a 1, e todos os pontos parecem “igualmente distantes” uns dos outros.

O segundo problema é mais insidioso: a euclidiana é sensível à **magnitude** dos vetores. Considere dois cenários:

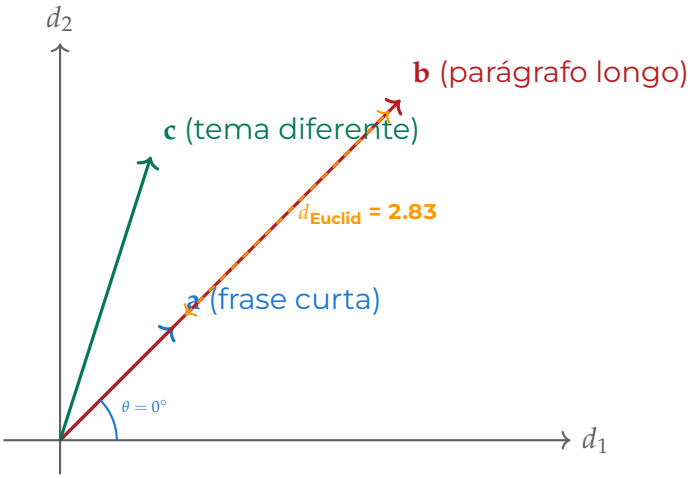


Figura 1: A armadilha da euclidiana: **a** e **b** apontam na mesma direção (mesmo significado!), mas a distância euclidiana entre eles é grande (2.83) por causa da diferença de magnitude. Já **c**, que fala de outro assunto, pode estar mais “perto” euclidianamente se sua norma for similar à de **a**.

Um parágrafo de 200 palavras sobre “política econômica” e uma frase de 5 palavras sobre “política econômica” podem ter o mesmo *sentido*, mas magnitudes de Embedding drasticamente diferentes. A euclidiana os classifica

como “distantes”. Precisamos de uma métrica que ignore o *comprimento* do vetor e foque exclusivamente na *direção*.

3 A Similaridade de Cosseno

A solução elegante adotada universalmente pela comunidade de NLP é a **Similaridade de Cosseno**. Ela mede o **ângulo** θ entre dois vetores, ignorando completamente suas magnitudes. A derivação parte da definição geométrica do produto escalar que vocês viram em Álgebra Linear:

$$\mathbf{a} \cdot \mathbf{b} = ||\mathbf{a}|| \cdot ||\mathbf{b}|| \cdot \cos(\theta) \tag{2}$$

Isolando o cosseno:

$$\cos(\theta) = \frac{\mathbf{a} \cdot \mathbf{b}}{||\mathbf{a}|| \cdot ||\mathbf{b}||} = \frac{\sum_{i=1}^{n} a_i \cdot b_i}{\sqrt{\sum_{i=1}^{n} a_i^2} \cdot \sqrt{\sum_{i=1}^{n} b_i^2}} \tag{3}$$

O resultado varia entre -1 e $+1$:

- $+1$**:** Os vetores apontam exatamente na mesma direção — significado idêntico.
- 0**:** Os vetores são ortogonais — sem relação semântica.
- -1**:** Os vetores apontam em direções opostas — significados antagônicos (raro em NLP).

3.1 Exemplo Numérico Passo-a-Passo

Vamos calcular a similaridade entre uma Query $A = [3, 4]$ (“problemas cardíacos”) e dois documentos: $B = [6, 8]$ (“anomalia cardíaca”) e $C = [4, -3]$ (“coração partido”).

Cosseno de A e B:

$$A \cdot B = 3 \times 6 + 4 \times 8 = 50 \tag{4}$$

$$||A|| = \sqrt{9 + 16} = 5, \quad ||B|| = \sqrt{36 + 64} = 10 \tag{5}$$

$$\cos\theta = \frac{50}{5 \times 10} = \mathbf{1.0} \quad \text{(mesma direção!)} \tag{6}$$

Cosseno de A e C:

$$A \cdot C = 3 \times 4 + 4 \times (-3) = 0 \tag{7}$$

$$||C|| = \sqrt{16 + 9} = 5 \tag{8}$$

$$\cos\theta = \frac{0}{5 \times 5} = \mathbf{0.0} \quad \text{(ortogonais!)} \tag{9}$$

Apesar de B ter o dobro da magnitude de A, o cosseno é 1.0: eles apontam na mesma direção (mesmo significado). C é ortogonal a A: semanticamente irrelevante. A euclidiana teria "achado" C mais perto de A ($d = 7.07$) do que B ($d = 5.0$), invertendo completamente o ranking correto.

3.2 A Normalização L2 e a Hiperesfera Unitária

Quando dividimos cada vetor por sua norma L2, todos os vetores são projetados para **comprimento unitário**:

$$\hat{\mathbf{a}} = \frac{\mathbf{a}}{||\mathbf{a}||} \tag{10}$$

Após essa normalização, todos os vetores repousam sobre a superfície de uma **hiperesfera de raio 1**. Nessa superfície, o produto escalar entre dois vetores normalizados *já é* o cosseno:

$$\hat{\mathbf{a}} \cdot \hat{\mathbf{b}} = \cos(\theta) \tag{11}$$

◇ Informação

Muitos bancos de dados vetoriais (como o ChromaDB) pré-normalizam os vetores durante a inserção. Assim, a busca se resume a um simples `dot product` — a operação mais rápida em hardware GPU. Essa otimização é invisível para o usuário, mas crucial para a performance.

4 Busca Léxica vs. Busca Semântica

Com a Similaridade de Cosseno formalizada, podemos finalmente contrastar os dois paradigmas de recuperação de informação:

Tabela 1: Comparação entre busca léxica (tradicional) e busca semântica (vetorial).

	Busca Léxica	**Busca Semântica**
Opera sobre	Caracteres/strings	Vetores/embeddings
Compara	Presença de palavras-chave	Ângulo entre vetores
Sinonímia	Falha ("carro" ≠ "automóvel")	Funciona (vetores próximos)
Polissemia	Falha ("banco" = "banco")	Funciona (contextos distintos)
Tecnologia	SQL `LIKE`, ElasticSearch	Embeddings + Cosseno

▷ Exemplo: title=Caso Real: Busca em Prontuários Médicos

Um médico pesquisa: *"Pacientes com problemas no coração."*

- **Busca léxica:** Retorna o prontuário "João teve problemas no trabalho e quebrou o coração após separação" (match de "problemas" + "coração") e **ignora** "Paciente com anomalia cardíaca grave" (nenhuma palavra-chave em comum).
- **Busca semântica:** Retorna corretamente "anomalia cardíaca" e "suspeita de infarto" como Top-2, pois seus vetores apontam na mesma direção que "problemas no coração" no espaço latente.

5 O Fluxo de uma Busca Semântica

Com Embeddings e Similaridade de Cosseno em mãos, o fluxo de uma busca semântica é elegantemente simples:

1. **Indexação (offline):** Para cada documento do acervo, gerar o vetor de Embedding usando o mesmo modelo (ex: spaCy `pt_core_news_md`) e armazená-lo.
2. **Consulta (online):** Converter a pergunta do usuário em um vetor usando o *mesmo* modelo.
3. **Ranking:** Calcular a Similaridade de Cosseno entre o vetor da consulta e *todos* os vetores do banco. Retornar os Top-K mais similares.

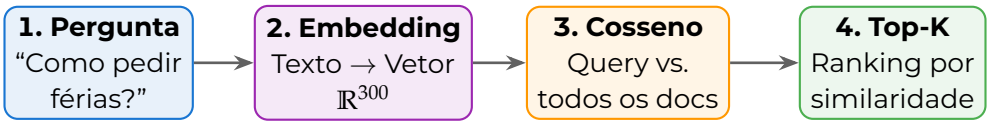


Figura 2: Pipeline de busca semântica: pergunta em linguagem natural → vetor → cosseno contra todos os documentos → ranking Top-K.

▶ Prática: title=Implementação do Cosseno em NumPy

```python
import numpy as np

def similaridade_cosseno(a, b):
    """Calcula a Similaridade de Cosseno entre dois vetores."""
    dot_product = np.dot(a, b)       # Numerador: produto escalar
    norm_a = np.linalg.norm(a)       # Denominador: norma L2 de A
    norm_b = np.linalg.norm(b)       # Denominador: norma L2 de B
    if norm_a == 0 or norm_b == 0:
        return 0.0                   # Previne divisao por zero
    return dot_product / (norm_a * norm_b)
```

6 RAG: Retrieval-Augmented Generation

A busca semântica não é apenas uma curiosidade acadêmica — ela é o núcleo da arquitetura que dominou a IA corporativa entre 2023 e 2026: o **RAG (Retrieval-Augmented Generation)**.

O problema que o RAG resolve é direto: LLMs como o GPT "alucinam" fatos e desconhecem dados internos da empresa. Re-treinar o modelo com dados corporativos custa milhões de dólares. A alternativa elegante é buscar semanticamente os trechos mais relevantes dos documentos da empresa e **injetar esses trechos no prompt** do LLM antes de gerar a resposta.

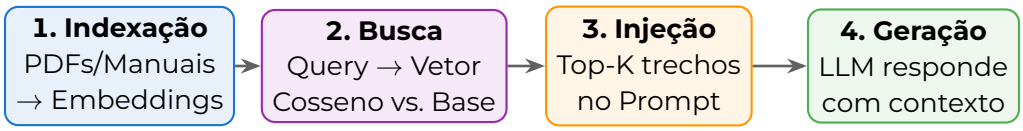


Figura 3: O ciclo de vida do RAG: documentos são pré-indexados como vetores; a consulta do usuário é vetorizada; os Top-K trechos mais similares são injetados no prompt do LLM; o LLM gera a resposta baseando-se exclusivamente nos documentos reais da empresa.

7 Limitações e o Problema da Escala

O método ingênuo de calcular o cosseno contra *todos* os vetores do banco é $O(N \times D)$, onde N é o número de documentos e D a dimensionalidade. Para um banco com 10 milhões de documentos e vetores de 1536 dimensões, isso significa $1{,}5 \times 10^{10}$ multiplicações por consulta — inviável em tempo real. Se 100 usuários perguntarem simultaneamente, são $1{,}5 \times 10^{12}$ operações. Servidores entram em colapso.

△ Importante: title=Busca Híbrida (Hybrid Search)

Na prática industrial, muitos sistemas combinam busca semântica com busca léxica (BM25) via *Reciprocal Rank Fusion*. A semântica captura sinônimos e intenção; a léxica captura termos exatos (códigos de produto, nomes próprios). A fusão dos dois rankings produz resultados superiores a qualquer abordagem isolada.

Na próxima aula, resolveremos o gargalo de escala com **Bancos de Dados Vetoriais** especializados que utilizam algoritmos de busca aproximada (ANN — *Approximate Nearest Neighbors*) para encontrar os vizinhos mais próximos em milissegundos.

8 Conclusão

Neste capítulo, formalizamos a ponte entre a representação vetorial (Aula 9) e a busca por significado. A Similaridade de Cosseno emerge naturalmente do produto escalar que vocês já dominavam desde a Álgebra Linear, mas sua aplicação ao NLP transforma a recuperação de informação de um problema de string matching para um problema de **geometria em alta dimensionalidade**. O RAG, por sua vez, é a prova de que essa geometria tem valor industrial concreto: é o motor que alimenta os chatbots corporativos mais avançados do planeta.

✔ Takeaways

- A **distância euclidiana** falha em espaços de Embeddings por ser sensível à magnitude — textos de tamanhos diferentes sobre o mesmo tema parecem "distantes".
- A **Similaridade de Cosseno** ($\cos\theta = \frac{A \cdot B}{\|A\|\|B\|}$) foca no ângulo entre vetores, ignorando magnitude. Valor de -1 a $+1$.
- A **normalização L2** projeta todos os vetores na hiperesfera unitária, transformando o cosseno em um simples `dot product`.
- A **busca semântica** encontra documentos por significado: "automóvel" e "carro" têm cosseno > 0.9 apesar de não compartilharem caracteres.
- O **RAG** injeta os Top-K trechos mais similares no prompt do LLM, eliminando alucinações sem re-treinar o modelo.
- O gargalo é $O(N \times D)$ por query — inviável para milhões de documentos sem indexação ANN.

✎ Questões: Autoavaliação

1 **[Direta]** Dado $\vec{A} = [1, 2, 3]$ e $\vec{B} = [2, 4, 6]$, calcule a Similaridade de Cosseno. O que o resultado revela sobre a relação entre os dois vetores?

2 **[Direta]** Explique por que, após normalização L2, o produto escalar entre dois vetores é numericamente igual à Similaridade de Cosseno.

3 **[Direta]** Um sistema RAG retornou documentos com scores de cosseno 0.92, 0.87 e 0.43. Quais deles você injetaria no prompt do LLM? Justifique com base nos thresholds apresentados.

4 **[Reflexiva]** A busca semântica "resolve" sinonímia, mas pode falhar em domínios altamente especializados (ex: jargão jurídico). Proponha uma estratégia para mitigar esse problema usando conceitos deste capítulo.

5 **[Reflexiva]** O RAG permite que empresas usem LLMs sem re-treinar. Isso democratiza a IA ou cria dependência de fornecedores de modelos? Discuta os trade-offs.

Lab. 10: Buscador Semântico com Cosseno

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

⋆ Objetivo do Laboratório

No Lab 09, você provou que palavras viram vetores e que vetores capturam semântica. Agora, você implementará um **motor de busca semântica do zero**, sem nenhuma abstração. Usando spaCy para gerar vetores e NumPy para álgebra linear pura, você construirá manualmente a função de Similaridade de Cosseno, criará um corpus de frases diversas, implementará o ranking Top-K completo, e provará que a busca semântica encontra documentos relevantes **mesmo sem nenhuma palavra-chave em comum**.

9 Parte 1: O Problema J — Estilo Maratona

Problema J: O Oráculo de Contratos da NeuroLex

Contexto: A NeuroLex (do Lab 09) gostou da sua PoC com vetores e agora quer o protótipo seguinte: um **buscador semântico** que receba uma query em linguagem natural e retorne os 3 documentos mais relevantes de um corpus de 10 cláusulas contratuais. O CTO impôs uma restrição educativa: *"Quero que o time implemente o cosseno na mão, sem importar nenhuma função pronta. Só NumPy puro."*
O diferencial é que as queries não compartilham **nenhuma palavra-chave** com os documentos — a busca funciona por *significado*, não por *casamento de strings*.

Modelo de Entrada: Query em texto livre → vetorização spaCy → cosseno manual.

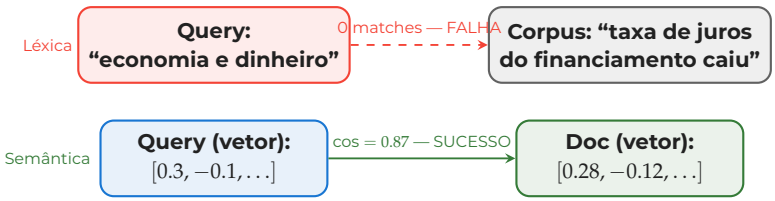
Modelo de Saída: Ranking Top-3 com scores de similaridade.

Critérios de Aprovação:

1. Cosseno de vetores idênticos $= 1.0$ (autoverificação)
2. Cosseno de vetores ortogonais $= 0.0$
3. Query "economia e dinheiro" retorna documentos financeiros no Top-2

10 Parte 2: Entendendo o Problema — Léxica vs Semântica

Para entender por que este lab é necessário, considere o seguinte cenário: um usuário digita "economia e dinheiro" no buscador. O corpus tem a frase "A taxa de juros do financiamento imobiliário caiu ontem". Um buscador léxico (Ctrl+F) não encontraria essa frase, pois nenhuma palavra da query aparece no documento. Já o buscador semântico entende que "juros" e "financiamento" estão no mesmo campo semântico que "economia" e "dinheiro".



A busca léxica compara **caracteres**; a busca semântica compara **vetores**. E como vimos no Lab 09, vetores capturam o *significado*, não a forma escrita.

11 Parte 3: Montando o Corpus (5 min)

Crie o arquivo `src/buscador_cosseno.py`. O primeiro passo é definir um "banco de dados" de frases diversas. Observe que o corpus foi construído intencionalmente com **zero sobreposição de palavras-chave** com as queries que testaremos depois:

```python
import spacy
import numpy as np
from numpy.linalg import norm

# 1. Carrega modelo em portugues (com vetores embutidos)
nlp = spacy.load("pt_core_news_md")

# 2. Nosso "Banco de Dados" --- NENHUMA frase contem keywords!
banco_documentos = [
    "O felino descansava tranquilamente no sofa da sala.",
    "A taxa de juros do financiamento imobiliario caiu ontem.",
    "O presidente sancionou a nova lei orcamentaria hoje.",
    "Receitas com chocolate precisam de bastante acucar.",
    "O medico diagnosticou uma anomalia cardiaca grave.",
    "A vacina da gripe estara disponivel na UBS amanha.",
    "O professor explicou a multiplicacao de matrizes.",
    "O cachorro brincava com as criancas no parque.",
    "Investidores apostam na queda do dolar este mes.",
    "A inteligencia artificial transformou a industria.",
]
```

Leia as 10 frases com atenção. Note que nenhuma contém as palavras "economia", "dinheiro", "saúde" ou "animal" — as queries que usaremos adiante. Se o buscador funcionar, será prova irrefutável de que a busca é por *significado*, não por strings.

12 Parte 4: Implementando o Cosseno na Mão (10 min)

A. A Fórmula

A Similaridade de Cosseno mede o ângulo θ entre dois vetores. Se ambos apontam na mesma direção, $\cos(\theta) = 1$; se são ortogonais, $\cos(\theta) = 0$:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{||\mathbf{A}|| \times ||\mathbf{B}||} = \frac{\sum a_i \cdot b_i}{\sqrt{\sum a_i^2} \times \sqrt{\sum b_i^2}}$$

O numerador é o **produto escalar** (`np.dot`) que vocês já viram em Álgebra Linear; o denominador é o produto das **normas L2** (`np.linalg.norm`).

B. Traduzindo para Código

```
# 3. Funcao de Similaridade de Cosseno --- SEM ABSTRACOES!
def cosine_similarity(vec_a, vec_b):
    """
    Calcula a Similaridade de Cosseno entre dois vetores.
    Formula: cos(theta) = (A . B) / (||A|| * ||B||)
    """
    dot_product = np.dot(vec_a, vec_b)   # Numerador
    norm_a = norm(vec_a)                  # ||A||
    norm_b = norm(vec_b)                  # ||B||
    if norm_a == 0 or norm_b == 0:        # Previne /0
        return 0.0
    return dot_product / (norm_a * norm_b)
```

A guard clause `if norm_a == 0` previne divisão por zero caso alguma palavra esteja fora do vocabulário (vetor nulo). Na prática, isso raramente acontece com o modelo `_md`.

△ Importante: Conexão com Labs Anteriores

O `np.dot()` é o mesmo produto escalar do Perceptron (Lab 01) e da MLP (Lab 02). A única novidade aqui é dividi-lo pelas normas para normalizar o resultado entre $[-1, 1]$. Você está reutilizando a mesma matemática em contextos cada vez mais sofisticados.

13 Parte 5: Ranking Top-K — Buscador Completo (15 min)

A. O Motor de Busca

Com o cosseno implementado, agora construímos o motor de busca que varre todo o banco e retorna os K documentos mais relevantes. O algoritmo é simples: vetorizar a query, calcular o cosseno contra cada documento, ordenar em ordem decrescente e retornar os K primeiros:

```
# 4. Motor de Busca Semantica com Ranking Top-K
def buscar_semanticamente(query_texto, banco, top_k=3):
    """
    Busca os top_k documentos mais semanticamente
    similares a query.
    """
    vetor_query = nlp(query_texto).vector
    resultados = []
    for idx, doc_texto in enumerate(banco):
        vetor_doc = nlp(doc_texto).vector
        score = cosine_similarity(vetor_query, vetor_doc)
        resultados.append((score, idx, doc_texto))
```

```
    resultados.sort(key=lambda x: x[0], reverse=True)
    return resultados[:top_k]
```

B. Testando com Queries Diversas

Agora vem o momento da verdade. Execute as 4 queries abaixo e observe quais documentos são retornados:

```
# 5. Testando com queries diversas
queries_teste = [
    "Me fale sobre economia e dinheiro.",
    "Problemas de saude e doencas.",
    "Animais domesticos e seus habitos.",
    "Tecnologia e computacao moderna.",
]

if __name__ == "__main__":
    for query in queries_teste:
        print(f"\n{'='*60}")
        print(f"QUERY: '{query}'")
        print(f"{'='*60}")
        top = buscar_semanticamente(query, banco_documentos)
        for rank, (score, idx, texto) in enumerate(top, 1):
            print(f"  #{rank} [Score: {score:.4f}] {texto}")
```

Observe: “economia e dinheiro” deve trazer “taxa de juros” e “investidores” como Top-2, mesmo sem palavra em comum. “Problemas de saúde” deve retornar “anomalia cardíaca” e “vacina da gripe”. Se “chocolate” aparecer no Top-3 de “economia”, algo está errado.

14 Parte 6: Comparando Léxica vs. Semântica (10 min)

Para completar a PoC, implemente um buscador léxico (equivalente ao Ctrl+F) e compare lado a lado com o semântico:

```
# 6. Busca Lexica (baseline ingenue) para comparacao
def buscar_lexicamente(query_texto, banco):
    """Busca documentos que contenham ALGUMA palavra da query."""
    palavras_query = set(query_texto.lower().split())
    resultados = []
    for doc in banco:
        palavras_doc = set(doc.lower().split())
        intersecao = palavras_query & palavras_doc
        resultados.append((len(intersecao), doc))
    resultados.sort(key=lambda x: x[0], reverse=True)
    return resultados[:3]

```

```
# 7. Comparacao direta
if __name__ == "__main__":
    query_teste = "Me fale sobre economia e dinheiro."
    print("\n=== COMPARACAO: LEXICA vs. SEMANTICA ===")
    print(f"Query: '{query_teste}'\n")

    print("--- Busca LEXICA (Ctrl+F) ---")
    for score, doc in buscar_lexicamente(
            query_teste, banco_documentos):
        print(f"  [Matches: {score}] {doc}")

    print("\n--- Busca SEMANTICA (Cosseno) ---")
    for score, idx, doc in buscar_semanticamente(
            query_teste, banco_documentos):
        print(f"  [Score: {score:.4f}] {doc}")
```

A busca léxica retornará frases com palavras genéricas como “e” ou “o” (que aparecem em quase todas as frases), sem nenhuma relevância temática. A semântica retornará documentos sobre finanças. A diferença é gritante — e essa comparação é o argumento mais poderoso da sua PoC para o CTO.

15 Parte 7: Garantia de Qualidade — Testes CI/CD (5 min)

Crie o arquivo `tests/test_buscador.py`. Cada teste valida uma propriedade matemática ou semântica:

```
import numpy as np
from src.buscador_cosseno import (
    cosine_similarity, buscar_semanticamente, banco_documentos
)

def test_cosseno_vetores_identicos():
    """Cosseno de um vetor consigo mesmo deve ser 1.0."""
    v = np.array([1.0, 2.0, 3.0])
    assert abs(cosine_similarity(v, v) - 1.0) < 1e-6

def test_cosseno_vetores_ortogonais():
    """Cosseno de vetores ortogonais deve ser 0.0."""
    v1 = np.array([1.0, 0.0])
    v2 = np.array([0.0, 1.0])
    assert abs(cosine_similarity(v1, v2)) < 1e-6

def test_busca_economia_retorna_financas():
    """Query sobre economia deve retornar docs financeiros."""
    resultados = buscar_semanticamente(
        "economia e dinheiro", banco_documentos, top_k=2)
    textos_top2 = [r[2] for r in resultados]
    financeiro = any(
```

```
        "juros" in t or "investid" in t or "dolar" in t
        for t in textos_top2
    )
    assert financeiro, f"Top-2 nao contem financas: {textos_top2}"
```

Execute `pytest tests/test_buscador.py -v`. Os **3 testes** devem passar.

16 Parte 8: Commit e Push

```
git add src/buscador_cosseno.py tests/test_buscador.py
git commit -m "Lab 10: Buscador semantico + cosseno manual + CI/CD"
git push origin main
```

Conclusão: O que Você Construiu Hoje

✔ Takeaways

- Você implementou a **Similaridade de Cosseno do zero** com NumPy puro — usando a mesma fórmula de Álgebra Linear que vocês viram na graduação.
- Construiu um **buscador semântico completo** com ranking Top-K que encontra documentos por *significado*, não por palavras-chave.
- Provou empiricamente que a **busca semântica supera a léxica**: "economia e dinheiro" encontra "taxa de juros" sem nenhuma palavra em comum.
- Os 3 testes CI/CD garantem: cosseno correto para vetores unitários, ortogonalidade e relevância semântica.

◇ Informação

Gancho para a Próxima Aula: Seu buscador funciona, mas usa um laço `for` que varre N documentos: $O(N \times D)$. Se o banco tiver 500 milhões de parágrafos, essa busca levaria **minutos** por query. Na próxima aula, você substituirá esse laço por um **Banco de Dados Vetorial (ChromaDB)** que usa o algoritmo HNSW para buscar em $O(\log N)$ — milissegundos ao invés de minutos. Seu código artesanal de hoje é o "motor" que o ChromaDB executa internamente.