

Lab. 10 (Parte 2): Engenharia de Recuperação, Métricas e Busca Híbrida

Aléssio Miranda Júnior

Setembro - 2026/2

Lab. 10 (Parte 2): Engenharia de Recuperação e Busca Híbrida

★ Objetivo da Sessão 2

Na primeira parte do Lab 10, você construiu um motor de busca semântica artesanal em NumPy e provou o conceito de recuperação por significado. No entanto, no ambiente profissional de engenharia, **uma demonstração visual no terminal não é evidência de qualidade**.

Nesta segunda parte, você evoluirá seu código de um protótipo ilustrativo para um sistema robusto de **Engenharia de Recuperação de Informação (Information Retrieval - IR)**:

- 1 Implementará avaliação quantitativa formal com **Ground Truth**, calculando **Precision@2** e **Recall@2**.
- 2 Avaliará como a busca densa se comporta com códigos exatos e identificadores técnicos.
- 3 Implementará um algoritmo de **Busca Híbrida (Hybrid Search)** e medirá seus resultados contra os sinais isolados.
- 4 Construirá uma política de **Abstenção calibrável** e metadados de proveniência para um pipeline RAG auditável.

Dinâmica Operacional da Bancada:

- **Alunos concluindo a Parte 1:** Usem os primeiros 5 minutos para finalizar o cálculo do cosseno e aprovar os testes iniciais (pytest tests/test_busca.py).
- **Alunos iniciando a Parte 2:** Mantenham o arquivo src/busca_cosseno.py e incorporem as novas classes, métricas e testes estruturados a seguir.

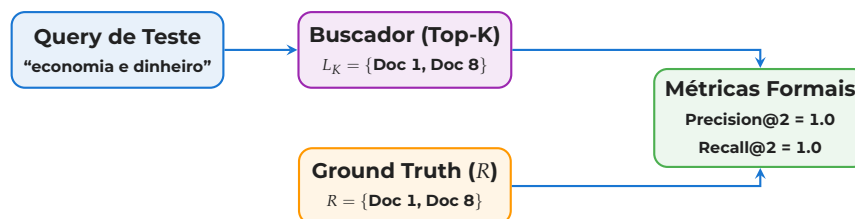
1 Parte 1: Por que Avaliar com Métricas Formais? (5 min)

Na Parte 1, rodamos uma consulta e verificamos visualmente que a frase de juro apareceu no topo. Mas o que acontece se o corpus tiver 100.000 documentos e 500 tópicos diferentes? Como garantir que uma alteração no modelo de embeddings não degradou o ranking de termos técnicos?

Em Recuperação de Informação (IR), a qualidade é avaliada comparando os documentos retornados (L_K) contra um conjunto de documentos rotulados como relevantes por humanos ou especialistas (R):

$$\text{Precision@K} = \frac{|R \cap L_K|}{K} \quad \text{e} \quad \text{Recall@K} = \frac{|R \cap L_K|}{|R|} \quad (1)$$

- **Precision@K (Precisão no Top-K):** De tudo o que o buscador trouxe nas primeiras K posições, que proporção é realmente relevante? Mede o *ruído*.
- **Recall@K (Revocação no Top-K):** De tudo o que era relevante no corpus para aquela query, que proporção conseguimos capturar dentro das primeiras K posições? Mede a *cobertura*.



2 Parte 2: Implementando Ground Truth e Métricas (8 min)

Abra o arquivo `src/buscador_cosseno.py` e adicione as funções de cálculo de métricas e o conjunto de consultas rotuladas para o nosso corpus de 10 documentos:

```

1 # 8. Metricas de Recuperacao de Informacao (IR)
2 def calcular_precision_at_k(relevantes_esperados, recuperados_top_k,
3   k):
4     """Calcula Precision@K: proporcao de recuperados que sao
5       relevantes."""
6     if k == 0:
7         return 0.0
8     recuperados_k = set(recuperados_top_k[:k])
9     relevantes = set(relevantes_esperados)
10    intersecao = recuperados_k & relevantes
11    return len(intersecao) / k
12
13 def calcular_recall_at_k(relevantes_esperados, recuperados_top_k, k):
14     """Calcula Recall@K: proporcao de relevantes esperados que foram
15       recuperados."""
16     if not relevantes_esperados:
17         return 0.0
18     recuperados_k = set(recuperados_top_k[:k])
19     relevantes = set(relevantes_esperados)
20     intersecao = recuperados_k & relevantes
21     return len(intersecao) / len(relevantes)
  
```

```

19
20 # 9. Conjunto de Consultas Rotuladas (Ground Truth)
21 # Mapeia a query para os indices exatos do banco_documentos que
    respondem a ela:
22 # Indices do banco:
23 # 0: gato sofa | 1: taxa juros | 2: lei orcamentaria | 3: chocolate
    acucar
24 # 4: cardiaca | 5: vacina UBS | 6: matrizes | 7: cachorro parque
25 # 8: investidores dolar | 9: IA industria
26 ground_truth_testes = [
27     {
28         "query": "economia e dinheiro",
29         "relevantes": [1, 8] # juros e investidores dolar
30     },
31     {
32         "query": "problemas de saude",
33         "relevantes": [4, 5] # anomalia cardiaca e vacina UBS
34     },
35     {
36         "query": "animais de estimacao",
37         "relevantes": [0, 7] # felino sofa e cachorro parque
38     },
39     {
40         "query": "matematica e computacao",
41         "relevantes": [6, 9] # multiplicacao matrizes e IA industria
42     }
43 ]

```

Agora, crie uma função de avaliação comparativa que calcule a média de Precision@2 e Recall@2 sobre todas as queries para os dois buscadores:

```

1 def avaliar_buscadores(banco, ground_truth, k=2):
2     """Compara quantitativamente Busca Lexica vs Busca Semantica."""
3     prec_lex, rec_lex = [], []
4     prec_sem, rec_sem = [], []
5
6     for item in ground_truth:
7         q = item["query"]
8         rel = item["relevantes"]
9
10        # Baseline de sobreposicao lexica (nao e BM25)
11        res_lex = buscar_lexicamente(q, banco)
12        # Identificar os indices retornados no banco original
13        indices_lex = [banco.index(doc) for _, doc in res_lex]
14        prec_lex.append(calcular_precision_at_k(rel, indices_lex, k))
15        rec_lex.append(calcular_recall_at_k(rel, indices_lex, k))
16
17        # Busca Semantica
18        res_sem = buscar_semanticamente(q, banco, top_k=k)
19        indices_sem = [idx for _, idx, _ in res_sem]
20        prec_sem.append(calcular_precision_at_k(rel, indices_sem, k))
21        rec_sem.append(calcular_recall_at_k(rel, indices_sem, k))

```

3 PARTE 3: IDENTIFICADORES EXATOS COMO CASO DE TESTE (7 MIN) 5

```

22
23     return {
24         "lexica": {"precision": np.mean(prec_lex), "recall": np.mean(
25             rec_lex)},
26         "semantica": {"precision": np.mean(prec_sem), "recall": np.
27             mean(rec_sem)}
28     }

```

Execute essa avaliação no bloco principal:

```

1  if __name__ == "__main__":
2      resultado = avaliar_buscadores(banco_documentos,
3          ground_truth_testes, k=2)
4      print("\n" + "="*50)
5      print("AVALIACAO FORMAL DE RECUPERACAO (K=2)")
6      print("="*50)
7      print(f"Busca Lexica -> Precision@2: {resultado['lexica']['precision']:.2f} | Recall@2: {resultado['lexica']['recall']:.2f}")
8      print(f"Busca Semantica -> Precision@2: {resultado['semantica']['precision']:.2f} | Recall@2: {resultado['semantica']['recall']:.2f}")

```

Execute o benchmark e registre os resultados observados. Eles podem variar conforme a versão do modelo, o corpus e a tokenização; o objetivo é fundamentar uma decisão técnica com dados, não confirmar um vencedor antecipadamente.

3 Parte 3: Identificadores Exatos como Caso de Teste (7 min)

Embeddings não são infalíveis. Quando nos deparamos com **identificadores exatos**, números de protocolo, códigos de peças ou siglas industriais (ex: ERR-504, PORT-8080, ISO-9001), a busca densa pode perder o sinal literal. O efeito depende do modelo, da tokenização e do corpus; por isso, vamos medi-lo.

Vamos provar essa falha. Adicione ao `banco_documentos` um documento com código técnico:

```

1  # Adicionando documento tecnico com identificador estrito ao banco
2  doc_tecnico = "Incidente de infraestrutura: o erro ERR-504 exige
3      reinicializacao forçada do pod."
4  banco_documentos.append(doc_tecnico)
5  # Indice do doc_tecnico = 10

```

Agora teste a query: "Como diagnosticar o erro ERR-504?":

```

1  query_codigo = "Como diagnosticar o erro ERR-504?"
2
3  print("\n--- Teste com Codigo Exato: ERR-504 ---")

```

```

4 print("Top-1 Lexica:")
5 res_l = buscar_lexicamente(query_codigo, banco_documentos)
6 print(f"  Matches: {res_l[0][0]} -> {res_l[0][1]}")
7
8 print("Top-1 Semantica:")
9 res_s = buscar_semanticamente(query_codigo, banco_documentos, top_k
    =1)
10 print(f"  Score: {res_s[0][0]:.4f} -> {res_s[0][2]}")

```

△ Importante: O que observar?

Compare os rankings. A sobreposição léxica preserva o token ERR-504; a busca densa pode recuperar contexto técnico relacionado ou perder o identificador. Não conclua a partir de uma única query: registre o resultado e use consultas adicionais no benchmark.

4 Parte 4: Implementando a Busca Híbrida (8 min)

Muitos sistemas combinam sinais léxicos e densos por **Busca Híbrida (Hybrid Search)**. A fusão pode preservar termos literais e paráfrases, mas deve ser comparada com as abordagens isoladas no domínio real.

Uma forma didática de combinar é a soma ponderada de scores transformados para escalas comparáveis:

$$\text{Score}_{\text{Híbrido}} = \alpha \cdot \text{Score}_{\text{Semântico}} + (1 - \alpha) \cdot \text{Score}_{\text{Léxico}} \quad (2)$$

Implemente o buscador híbrido em `src/buscador_cosseno.py`:

```

1 # 10. Motor de Busca Hibrida (Semantica + Lexica)
2 def buscar_hibrido(query_texto, banco, alpha=0.5, top_k=3):
3     """
4     Combina busca semantica e lexica com ponderacao alpha.
5     alpha = 1.0 -> 100% semantica
6     alpha = 0.0 -> 100% lexica
7     """
8     vetor_query = nlp(query_texto).vector
9     palavras_query = set(query_texto.lower().split())
10
11     scores_semanticos = []
12     scores_lexicos = []
13
14     for doc in banco:
15         # Score semantico (cosseno)
16         v_doc = nlp(doc).vector
17         s_sem = cosine_similarity(vetor_query, v_doc)
18         scores_semanticos.append(s_sem)

```

```

19
20     # Score lexico (intersecao)
21     p_doc = set(doc.lower().split())
22     s_lex = len(palavras_query & p_doc)
23     scores_lexicos.append(float(s_lex))
24
25     # Normalizacao Min-Max dos scores lexicos para [0, 1]
26     max_lex = max(scores_lexicos) if max(scores_lexicos) > 0 else 1.0
27     scores_lex_norm = [s / max_lex for s in scores_lexicos]
28
29     # Combinacao ponderada
30     resultados = []
31     for idx, doc in enumerate(banco):
32         score_hibrido = (alpha * scores_semanticos[idx]) + ((1 -
33             alpha) * scores_lex_norm[idx])
34         resultados.append((score_hibrido, idx, doc))
35
36     resultados.sort(key=lambda x: x[0], reverse=True)
37     return resultados[:top_k]

```

Teste a query "Como diagnosticar o erro ERR-504?" e compare os três rankings. O valor de α deve ser escolhido no conjunto de validação, não ajustado para favorecer esta única consulta.

5 Parte 5: RAG Responsável — Abstenção e Citações (7 min)

Um dos maiores erros em arquiteturas RAG corporativas é a ausência de um **mecanismo de recusa ou abstenção**. Se o usuário perguntar: “Qual a receita do pudim de leite condensado?”, um buscador ingênuo forçará o retorno de 3 documentos (provavelmente a frase de chocolate e açúcar), o LLM tentará inventar uma resposta a partir desse contexto inadequado, e o sistema alucinará.

Um sistema responsável deve:

- 1 Definir uma **política de abstenção calibrada**: score, cobertura e domínio podem sinalizar que o sistema deve se abster.
- 2 Devolver **metadados de proveniência** (ID, versão, score e trecho) para permitir citação auditável.

```

1 # 11. Recuperador com Mecanismo de Abstencao e Metadados Auditaveis.
2 # Calibre threshold no conjunto de validacao do modelo e corpus em
   uso.
3 def recuperar_para_rag(query_texto, banco, threshold, top_k=2,
4     buscador=buscar_semanticamente):
5     """

```

```

6     Recupera contexto para RAG. Se nenhum documento atingir o
       threshold minimo,
7     o sistema se abstem de responder para evitar alucinacoes no LLM.
8     """
9     candidatos = buscador(query_texto, banco, top_k=top_k)
10
11     if not candidatos:
12         return {
13             "status": "ABSTENCAO",
14             "motivo": "Nenhuma evidencia foi recuperada.",
15             "documentos": []
16         }
17
18     # Score baixo e apenas um dos sinais da politica de abstencao.
19     if candidatos[0][0] < threshold:
20         return {
21             "status": "ABSTENCAO",
22             "motivo": f"Score maximo ({candidatos[0][0]:.3f})
                       inferior ao threshold ({threshold}).",
23             "documentos": []
24         }
25
26     # Documentos aprovados com metadados de citacao
27     evidencias = []
28     for score, idx, texto in candidatos:
29         if score >= threshold:
30             evidencias.append({
31                 "doc_id": f"DOC-{idx:03d}",
32                 "score": round(score, 4),
33                 "source_version": "corpus-lab-1",
34                 "trecho": texto
35             })
36
37     return {
38         "status": "SUCESSO",
39         "motivo": "Evidencias confiaveis encontradas no corpus.",
40         "documentos": evidencias
41     }

```

Teste com duas queries no bloco de execução:

```

1  if __name__ == "__main__":
2      print("\n--- Teste de Abstencao e Citacoes para RAG ---")
3      q_valida = "investimentos no mercado de acoes e cambio"
4      q_invalida = "qual e a receita tradicional de bolo de fuba?"
5
6      print("1. Query Valida:", q_valida)
7      print("    Resultado:", recuperar_para_rag(q_valida,
8                                                  banco_documentos))
9
10     print("\n2. Query Fora de Escopo:", q_invalida)

```

```
10 print("    Resultado:", recuperar_para_rag(q_invalida,
    banco_documentos))
```

6 Parte 6: Novos Testes Automatizados no Pytest (3 min)

Abra o arquivo `tests/test_buscador.py` e adicione os 3 testes que validam a Parte 2:

```
1 from src.buscador_cosseno import (
2     calcular_precision_at_k, calcular_recall_at_k,
3     buscar_hibrido, recuperar_para_rag
4 )
5
6 def test_metricas_ir_em_ranking_conhecido():
7     relevantes = [1, 2]
8     recuperados = [1, 3]
9     assert calcular_precision_at_k(relevantes, recuperados, k=2) ==
10         0.5
11     assert calcular_recall_at_k(relevantes, recuperados, k=2) == 0.5
12
13 def test_busca_hibrida_recupera_codigo_exato():
14     """Garante que a busca hibrida recupera o codigo tecnico ERR-504
15     em primeiro lugar."""
16     resultados = buscar_hibrido("Como resolver o erro ERR-504?",
17         banco_documentos, alpha=0.5, top_k=1)
18     doc_top1 = resultados[0][2]
19     assert "ERR-504" in doc_top1
20
21 def test_mecanismo_abstencao_sem_candidatos():
22     def buscador_vazio(query, banco, top_k):
23         return []
24
25     res = recuperar_para_rag(
26         "Qual o melhor fertilizante para plantar couve?",
27         banco_documentos,
28         threshold=0.55, buscador=buscador_vazio)
29     assert res["status"] == "ABSTENCAO"
30     assert len(res["documentos"]) == 0
```

Execute a suíte de testes no terminal:

```
1 pytest tests/test_buscador.py -v
```

Todos os **6 testes** (3 da Parte 1 e 3 da Parte 2) devem passar com selo verde!

7 Parte 7: Commit e Push (2 min)

```
1 git add src/buscador_cosseno.py tests/test_buscador.py
2 git commit -m "Lab 10 (Parte 2): Metricas IR, busca hibrida e
   politica de abstencao"
3 git push origin main
```

Conclusão: Da PoC Ingênua à Engenharia de IR

✓ Takeaways

- Você aprendeu que **uma demo bonita não é métrica**: sistemas de busca corporativos são avaliados com **Precision@K** e **Recall@K** sobre consultas rotuladas.
- Investigou como termos exatos, números e códigos de erro (ERR-504) se comportam nos sinais denso e léxico.
- Implementou a **Busca Híbrida (Hybrid Search)** e registrou que seus ganhos dependem de validação.
- Desenvolveu a **Política de Abstenção**: um sistema RAG deve saber se calar quando não possui evidências suficientes no corpus.

◇ Informação

Gancho para a Próxima Aula (Aula 11 - ChromaDB): Seu buscador agora produz rankings que podem ser medidos, comparados e recusados quando falta evidência. Mas observe como a busca é executada: um laço `for` que calcula cossenos vetor a vetor na CPU ($O(N \times D)$). Na Aula 11, investigaremos índices vetoriais aproximados e seus trade-offs de recall, latência e memória.