Lab. 14: Inferência com Hugging Face Pipelines

Aléssio Miranda Júnior

Outubro - 2026/2

Lab. 14: Inferência com Hugging Face Pipelines

⋆ Objetivo do Laboratório

No Lab 13, carregar um modelo Transformer e extrair atenções exigiu dezenas de linhas de código. Nesta atividade final do Módulo 2, você descobrirá o poder da abstração `pipeline()` do Hugging Face — que reduz tarefas complexas de NLP a **três linhas de código**. Você implementará dois pipelines distintos (Análise de Sentimento e NER), aplicará em um corpus de avaliações de produtos, e validará tudo pelo CI/CD do GitLab.

1 Parte 1: O Problema N — Estilo Maratona

Problema N: O Monitor de Reputação da BrandPulse

Contexto: A *BrandPulse*, uma plataforma de monitoramento de marca, recebeu um contrato de um e-commerce para classificar automaticamente 1.000 avaliações de clientes por dia. O time de produto precisa de dois módulos:

1. **Sentimento:** Cada avaliação recebe uma nota de 1–5 estrelas (negativo a positivo)
2. **Entidades:** Extrair automaticamente nomes de produtos, marcas e locais mencionados

O prazo é apertado — o CTO quer a PoC pronta em 50 minutos usando modelos pré-treinados do Hugging Face Hub. Nenhum treinamento custom é permitido.

Modelo de Entrada: Texto livre (avaliações de clientes em português).
Modelo de Saída:

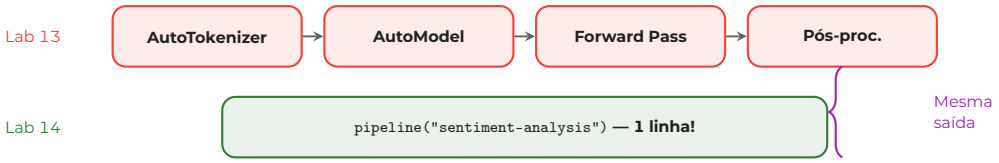
1. Sentimento (label + score) para cada avaliação
2. Lista de entidades nomeadas (NER) com tipo e confiança

Critérios de Aprovação:

1. Pipeline de sentimento produz label e score $\in [0, 1]$
2. Pipeline de NER detecta pelo menos 1 entidade em texto com nomes próprios
3. Sentimento de texto positivo $\neq$ sentimento de texto negativo
4. Ambos os pipelines funcionam sem GPU

2 Parte 2: Entendendo o Salto — Do Manual ao Automático

Para entender o valor da abstração `pipeline()`, compare o que você fez no Lab 13 (carregar tokenizer, modelo, fazer forward pass, pós-processar logits) com o que faremos hoje:



No Lab 13, você controlou cada passo manualmente — o que é ótimo para entender a mecânica. Mas na produção, precisamos de velocidade de desenvolvimento. A abstração `pipeline()` encapsula **tudo**: download do modelo, tokenização, inferência e pós-processamento.

• Teoria: O que o `pipeline()` Faz por Baixo

Quando você escreve `pipeline("sentiment-analysis")`, o Hugging Face:

1. **Baixa** o modelo e o tokenizer do Hub (cache local em `~/.cache/huggingface/`)
2. **Tokeniza** o texto de entrada (converte strings em IDs de tokens)
3. **Executa** o forward pass no modelo (CPU ou GPU, automaticamente)
4. **Pós-processa** os logits de saída em labels legíveis ("1 star", "5 stars", etc.)

Tudo isso em uma única chamada de função. O programador não precisa se preocupar com shapes de tensores, padding ou conversão de logits.

3 Parte 3: Instalação (3 min)

Se você já tem `transformers` e `torch` do Lab 13, está pronto. Caso contrário:

```
pip install transformers torch
```

O primeiro uso de cada modelo baixará os pesos do Hub (~600MB para o modelo de sentimento). Nas execuções seguintes, os pesos são carregados do cache local.

4 Parte 4: Pipeline de Análise de Sentimento (12 min)

A. Instanciando o Pipeline

Crie o arquivo `src/analise_sentimento.py`. O pipeline é instanciado com uma única linha:

```
from transformers import pipeline

# 1. Pipeline de Analise de Sentimentos (multilingual)
```

```
print("Carregando modelo de sentimento...")
analisador = pipeline(
    "sentiment-analysis",
    model="nlptown/bert-base-multilingual-uncased-sentiment"
)
```

O modelo `nlptown/bert-base-multilingual-uncased-sentiment` é um BERT multilingual fine-tunado em avaliações de produtos. Ele suporta 6 idiomas (incluindo português) e classifica textos em 5 categorias: "1 star" (muito negativo) a "5 stars" (muito positivo).

B. Analisando um Corpus de Avaliações

Agora vamos aplicar o modelo em lote sobre 5 avaliações fictícias de clientes:

```
# 2. Corpus de avaliacoes de clientes
avaliacoes = [
    "Achei o produto horrivel, a bateria viciou em 1 semana!",
    "Entrega rapida, produto excelente, recomendo!",
    "Produto ok, nada demais. Cumpre o que promete.",
    "Pessima experiencia. Atendimento grosseiro e demorado.",
    "Superou minhas expectativas! Qualidade incrivel.",
]

# 3. Analise em lote (batch)
print("\n=== ANALISE DE SENTIMENTO ===")
resultados_sentimento = analisador(avaliacoes)

for avaliacao, resultado in zip(avaliacoes, resultados_sentimento):
    estrelas = int(resultado['label'].split()[0])
    emoji = "*" * estrelas
    print(f"\n  Texto: {avaliacao[:60]}...")
    print(f"  Label: {resultado['label']} | "
          f"Score: {resultado['score']:.2%} | {emoji}")
```

Observe a saída: o campo `label` contém a classificação ("1 star" a "5 stars") e o `score` indica a confiança do modelo (0.0 a 1.0). A avaliação "produto horrível" deve receber 1–2 stars; "produto excelente" deve receber 4–5 stars.

△ Importante: Modelo Multilingual vs. Monolingual

O modelo `nlptown` suporta 6 idiomas incluindo português, mas não é treinado especificamente em PT-BR. Para resultados mais precisos em português, modelos da comunidade como `citizenlab/twitter-xlm-roberta-base-sentiment-finetunned` podem ser mais adequados — porém são maiores e mais lentos para baixar.

5 Parte 5: Pipeline de NER — Entidades Nomeadas (12 min)

A. O que é NER?

NER (*Named Entity Recognition*) é a tarefa de identificar e classificar automaticamente menções a entidades no texto: pessoas, organizações, locais, datas, etc. É essencial em aplicações como extração de informação, compliance e análise de mídia.

Tag	Significado	Exemplo
PER	Pessoa	Elon Musk, Lula
ORG	Organização	SpaceX, Google, Apple
LOC	Localização	São Paulo, Califórnia

B. Implementando o Pipeline de NER

```
# 4. Pipeline de NER (Named Entity Recognition)
print("\nCarregando modelo de NER...")
ner = pipeline(
    "ner",
    model="Davlan/bert-base-multilingual-cased-ner-hrl",
    aggregation_strategy="simple"
)
```

O parâmetro `aggregation_strategy="simple"` faz com que subpalavras sejam agrupadas: ao invés de detectar “Elon” e “Musk” separadamente, o modelo retorna “Elon Musk” como uma única entidade.

C. Testando com Textos Ricos em Entidades

```
# 5. Textos com entidades nomeadas
textos_ner = [
    "Elon Musk fundou a SpaceX em Hawthorne, California.",
    "O presidente Lula visitou a sede do Google em Sao Paulo.",
    "A Apple lancou o iPhone 15 em Cupertino em setembro.",
]

print("\n=== ENTIDADES NOMEADAS (NER) ===")
resultados_ner = []
for texto in textos_ner:
    entidades = ner(texto)
    resultados_ner.append(entidades)
    print(f"\n  Texto: {texto}")
    for ent in entidades:
        print(f"    [{ent['entity_group']:>5}] "
```

```
                f"{ent['word']:>20} "
                f"(confianca: {ent['score']:.2%})")
```

Observe os resultados: “Elon Musk” deve ser classificado como PER (pessoa), “SpaceX” como ORG (organização) e “Hawthorne” como LOC (localização). A confiança costuma ficar acima de 90% para entidades bem conhecidas.

6 Parte 6: Referência — O Ecossistema de Tasks (5 min)

O Hugging Face oferece dezenas de tarefas pré-configuradas via `pipeline()`. Leia a tabela abaixo para entender a amplitude do ecossistema:

Task	Uso	Código
`sentiment-analysis`	Classificar emoção	`pipeline("sentiment-analysis")`
`ner`	Extrair entidades	`pipeline("ner")`
`translation`	Traduzir idiomas	`pipeline("translation_en_to_pt")`
`summarization`	Resumir textos	`pipeline("summarization")`
`zero-shot-classification`	Classificar sem treino	`pipeline("zero-shot-classification")`
`fill-mask`	Completar frases	`pipeline("fill-mask")`

Cada uma dessas tasks pode ser instanciada em 1 linha e executada em 1 linha adicional. A barreira de entrada para NLP profissional nunca foi tão baixa.

▷ Exemplo: title=Desafio Extra

Escolha uma task adicional da tabela (ex: `fill-mask`) e teste com uma frase em português. Adicione os resultados ao seu script como um bloco extra. Isso demonstra versatilidade ao CTO da BrandPulse.

7 Parte 7: Garantia de Qualidade — Testes CI/CD (8 min)

Os testes validam as 4 propriedades exigidas pelo Problema N. Crie o arquivo `tests/test_sentimento.py`:

```
from transformers import pipeline

analisador = pipeline(
    "sentiment-analysis",
    model="nlptown/bert-base-multilingual-uncased-sentiment"
)
ner = pipeline(
    "ner",
```

```
        model="Davlan/bert-base-multilingual-cased-ner-hrl",
        aggregation_strategy="simple"
    )

    def test_sentimento_retorna_label_e_score():
        """Pipeline deve retornar label e score."""
        r = analisador("Produto excelente!")
        assert 'label' in r[0] and 'score' in r[0]
        assert 0 <= r[0]['score'] <= 1

    def test_ner_detecta_entidade():
        """NER deve encontrar pelo menos 1 entidade."""
        r = ner("Elon Musk fundou a SpaceX.")
        assert len(r) >= 1, "Nenhuma entidade detectada"

    def test_sentimento_positivo_vs_negativo():
        """Texto positivo deve ter sentimento diferente do negativo."""
        r_pos = analisador("Adorei o produto, incrivel!")[0]
        r_neg = analisador("Horrivel, pessimo, nunca mais!")[0]
        assert r_pos['label'] != r_neg['label'], \
            f"Mesmo label para pos e neg: {r_pos['label']}"

    def test_pipeline_funciona_sem_gpu():
        """Pipeline deve funcionar em CPU."""
        r = analisador("Texto de teste simples.")
        assert r is not None
```

Cada teste mapeia para um critério de aprovação: output estruturado, detecção de entidades, diferenciação de sentimento e compatibilidade CPU. Execute `pytest tests/test_sentimento.py -v` e confirme os **4 passed**.

8 Parte 8: Commit e Push

```
git add src/analise_sentimento.py tests/test_sentimento.py
git commit -m "Lab 14: Hugging Face Pipelines (Sentimento + NER) + CI/CD"
git push origin main
```

Conclusão: O que Você Construiu no Módulo 2

✔ Takeaways

- Você usou a abstração `pipeline()` para executar **Análise de Sentimento** em lote sobre avaliações de clientes — 3 linhas de código para um classificador que funciona em 6 idiomas.
- Implementou **Named Entity Recognition (NER)** para extrair automaticamente pessoas, organizações e locais de textos, com confiança $> 90\%$.
- Conheceu o ecossistema **Hugging Face Hub** — o "GitHub da IA" com 1M+ modelos prontos para uso imediato.
- Os 4 testes CI/CD garantem: output estruturado, detecção de entidades, diferenciação de sentimento e compatibilidade CPU.

◇ **Informação**

Fechamento do Módulo 2 — "A Revolução da Linguagem e dos Vetores". Em 6 aulas, você percorreu a jornada completa do NLP moderno: de Embeddings artesanais (Lab I) até Hugging Face Pipelines (Lab N), passando por busca semântica com cosseno manual, bancos vetoriais com HNSW, chunking de PDFs para RAG, e Self-Attention do Transformer.

No Módulo 3 — "IA Generativa e RAG", você consumirá **LLMs via API** (OpenAI, Gemini), dominará **Prompt Engineering**, e construirá um **sistema RAG corporativo completo** que combina tudo que você aprendeu: embeddings + banco vetorial + chunking + LLM = o produto que a indústria mais demanda hoje.