Comparação de Técnicas de Classificação de Texto em LLMs: Zero-Shot, Fine-Tuning e RAG

Comparativo de Estratégias de LLMs para Classificação em Domínios Técnicos

Disciplina : Modelos de Linguagem e Generativos
Curso: Mestrado Profissional em Computação Aplicada
Professor Rogério - Dezembro 2025

UNIVERSIDADE PRESBITERIANA MACKENZIE
FACULDADE DE COMPUTAÇÃO E INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

Desafio

Vanilla LLM 1. Implemente uma solução de Text Zero-Shot Classification com dados de um dos sites https://www.gutenberg.org/, http://arxiv.org/, https://www.imdb.com/ ou bulário eletrônico (https://consultas.anvisa.gov.br/#/bulario/). A solução deve mostrar e comparar os resultados (métricas) do modelo original e de uma solução com fine-tuning ou RAG (Retrieval-Augmented Generation).

Link colab:
https://colab.research.google.com/drive/1Mn1hrKbGQkq_jUVwTSVa39ZpG_bi8fj2?usp=sharing#scrollTo=NQYlOG69-FqO

O Problema

O Desafio da Classificação Técnica

Caso de Uso:

- Classificar resumos de artigos científicos do arXiv http://arxiv.org/ .
- Classes Alvo: Inteligência Artificial (AI “Categoria **`cs.AI`**”) vs. Machine Learning (ML “Categoria **`cs.LG`**”).

Pontos-chave:

- AI e ML são subdomínios altamente correlacionados.
- Grande sobreposição de vocabulário (ex: "Redes Neurais", "Otimização").
- O conhecimento genérico dos LLMs é suficiente?

Dataset e Metodologia

Dados e Preparação

- Fonte: API do arXiv (cs.AI e cs.LG).
- Volume Total: 1.000 Artigos (Título + Abstract).
- Balanceamento: 500 AI / 500 ML.

Métricas:

- F1 Macro Score: Métrica principal, crucial para avaliar o desempenho balanceado em ambas as classes (AI e ML).
- Accuracy (Acurácia).

Dataset e Metodologia

```python
# ============================================================
# 2. DOWNLOAD DOS ARTIGOS DO ARXIV (cs.AI E cs.LG)
# ============================================================
# Vamos buscar 500 artigos de cs.AI e 500 artigos de cs.LG

def buscar_artigos_arxiv(categoria, max_results=500):
    search = arxiv.Search(
        query=f"cat:{categoria}",
        max_results=max_results,
        sort_by=arxiv.SortCriterion.SubmittedDate
    )
    registros = []
    for result in search.results():
        registros.append({
            "id": result.entry_id,
            "title": result.title.strip().replace("\n", " "),
            "abstract": result.summary.strip().replace("\n", " "),
            "categories": " ".join(result.categories),
            "published": result.published
        })
    return pd.DataFrame(registros)

df_ai = buscar_artigos_arxiv("cs.AI", max_results=500)
df_ml = buscar_artigos_arxiv("cs.LG", max_results=500)

print("cs.AI:", df_ai.shape)
print("cs.LG:", df_ml.shape)
```

```
/tmp/ipython-input-2357164003.py:13: DeprecationWarning: The 'Search.results' method is deprecated, use 'Client.results' instead
  for result in search.results():
cs.AI: (500, 5)
cs.LG: (500, 5)
```

As 3 Estratégias Avaliadas

1. Zero-Shot: Classificação por Inferência de Linguagem Natural (NLI). Custo de Treino: Custo Zero.

Classificação Base (Zero-Shot)

Detalhes: Modelo: facebook/bart-large-mnli. Mecanismo: O modelo usa seu conhecimento pré-treinado em NLI para inferir a relação entre o resumo e os rótulos 'AI' ou 'ML'.

Resultado: Acurácia: ≈ 49.5% F1 Macro Score: ≈ 0.48 Conclusão: O modelo genérico falha na distinção. O desempenho é próximo ao acaso.

Zero-Shot:

```python
# ============================================================
# 5. ZERO-SHOT CLASSIFICATION (BART-MNLI)
# ============================================================
# Aqui usamos um modelo NLI (BART-MNLI) para Zero-Shot.

device = 0 if torch.cuda.is_available() else -1

zero_shot_classifier = pipeline(
    "zero-shot-classification",
    model="facebook/bart-large-mnli",
    device=device
)

candidate_labels = ["AI", "ML"]

# Para não ficar muito pesado, podemos limitar o número de exemplos de teste.
# Se quiser rodar em todos, troque n amostras por len(test_df).
N_ZERO_SHOT = min(300, len(test_df))

test_sample = test_df.sample(N_ZERO_SHOT, random_state=42).copy()
test_texts = (test_sample["title"] + " - " + test_sample["abstract"]).tolist()

preds_zeroshot = []
for text in tqdm(test_texts, desc="Zero-Shot (BART-MNLI)"):
    result = zero_shot_classifier(
        text,
        candidate_labels=candidate_labels,
        multi_label=False
    )
    preds_zeroshot.append(result["labels"][0])

test_sample["pred_zero_shot"] = preds_zeroshot

y_true = test_sample["label_text"].tolist()
y_pred_zero_shot = test_sample["pred_zero_shot"].tolist()

print("=== RELATÓRIO ZERO-SHOT (BART-MNLI) ===")
print(classification_report(y_true, y_pred_zero_shot, digits=4))

print("MATRIZ DE CONFUSÃO (Zero-Shot)")
print(confusion_matrix(y_true, y_pred_zero_shot, labels=["AI", "ML"]))

from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

labels = ["AI", "ML"]

cm = confusion_matrix(y_true, y_pred_zero_shot, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)

plt.figure(figsize=(5, 4))
disp.plot(cmap="Blues")
plt.title("Matriz de Confusão (Zero-Shot)")
plt.show()
```

As 3 Estratégias Avaliadas

2. Fine-Tuning: Ajuste supervisionado de pesos do modelo aos dados específicos. Custo de Treino: Alto (Exige GPU e tempo).

Detalhes: Modelo Base: SciBERT (allenai/scibert_scivocab_uncased). Vantagem SciBERT: Pré-treinado em literatura científica, otimizando o vocabulário. Treinamento: Supervisionado (800 exemplos) por 3 épocas.

Resultado: Acurácia: **62.0%** F1 Macro Score: **0.615** Conclusão: O ajuste supervisionado ao domínio é a rota para o melhor desempenho, mas exige alto investimento em recursos.

Fine-Tuning:

```python
# ============================================================
# 6. FINE-TUNING (SciBERT) PARA CLASSIFICAÇÃO AI vs ML
# ============================================================

model_name = "allenai/scibert_scivocab_uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)

def tokenize_function(examples):
    return tokenizer(
        examples["text"],
        truncation=True,
        padding="max_length",
        max_length=256
    )

# Criar datasets Hugging Face
train_dataset = Dataset.from_pandas(
    train_df[["title", "abstract", "label"]].rename(columns={"label": "labels"})
)
test_dataset = Dataset.from_pandas(
    test_df[["title", "abstract", "label"]].rename(columns={"label": "labels"})
)

# Concatenar título + abstract em um único campo de texto
def concat_title_abstract(example):
    example["text"] = example["title"] + " - " + example["abstract"]
    return example

train_dataset = train_dataset.map(concat_title_abstract)
test_dataset = test_dataset.map(concat_title_abstract)

train_dataset = train_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)

train_dataset = train_dataset.remove_columns(["title", "abstract", "text", "__index_level_0__"])
test_dataset = test_dataset.remove_columns(["title", "abstract", "text", "__index_level_0__"])

train_dataset.set_format("torch")
test_dataset.set_format("torch")

num_labels = 2

model = AutoModelForSequenceClassification.from_pretrained(
    model_name,
    num_labels=num_labels,
    id2label=id2label,
    label2id=label2id
)
```

As 3 Estratégias Avaliadas

3. RAG Retrieval-Augmented Classification (Solução Híbrida) **:** Aumento do prompt com contexto de busca semântica (FAISS). Custo de Treino: Baixo/Médio (Somente custo de indexação).

Detalhes da Implementação: **Modelos:** Combina Embeddings (`all-mpnet-base-v2`) + **Zero-Shot Classifier** (`BART-MNLI`). **Mecanismo:** Utiliza **FAISS** (índice vetorial) para buscar os **K=5** vizinhos mais similares do corpus completo. **Processo:** O texto original é **aumentado** com o contexto (e rótulos) dos vizinhos e enviado ao BART para inferência. **Objetivo:** Melhorar a decisão do Zero-Shot fornecendo **conhecimento de domínio** via contexto, sem Fine-Tuning.

Resultado: Acurácia: **57.5%** F1 Macro Score: **0.563** Conclusão: Demonstra o potencial do **RAG** como um "atalho inteligente", superando significativamente o Zero-Shot puro ao introduzir contexto especializado.

RAG:

```
# ============================================================
# 10. RAG SIMPLES PARA CLASSIFICAÇÃO (EMBEDDINGS + FAISS + ZERO-SHOT COM CONTEXTO)
# ============================================================
# Ideia: usar embeddings + FAISS para recuperar artigos similares
# para o texto de teste, e alimentar um modelo zero-shot com
# "texto + contexto dos vizinhos".

# 10.1. Criar embeddings do CORPUS COMPLETO (título + abstract)

embedder = SentenceTransformer("all-mpnet-base-v2")

corpus_texts = (df["title"] + " - " + df["abstract"]).tolist()
corpus_embeddings = embedder.encode(corpus_texts, show_progress_bar=True, convert_to_numpy=True)

# 10.2. Criar índice FAISS

d = corpus_embeddings.shape[1]
index = faiss.IndexFlatL2(d)
index.add(corpus_embeddings)

print("Tamanho do índice FAISS:", index.ntotal)
```

```
Batches: 100% ████████████████████ 32/32 [00:24<00:00,  2.01it/s]
Tamanho do índice FAISS: 1000
```

RAG:

```
# ============================================================
# 11. CLASSIFICAÇÃO COM RAG
# ============================================================

# Usaremos o mesmo zero_shot_classifier (BART-MNLI),
# mas agora com CONTEXTO dos k vizinhos mais similares.

K = 5  # quantidade de vizinhos para contexto
N_RAG = min(200, len(test_df))  # limitar para não ficar tão pesado

test_sample_rag = test_df.sample(N_RAG, random_state=123).copy()
texts_rag = (test_sample_rag["title"] + " - " + test_sample_rag["abstract"]).tolist()

preds_rag = []

for text in tqdm(texts_rag, desc="RAG Classification"):
    # Embedding do texto de teste
    query_emb = embedder.encode([text], convert_to_numpy=True)

    # Busca no índice FAISS
    D, I = index.search(query_emb, K)
    idx_neighbors = I[0]

    # Constrói contexto a partir dos vizinhos
    context_parts = []
    for idx in idx_neighbors:
        ctx_title = df.iloc[idx]["title"]
        ctx_abst = df.iloc[idx]["abstract"]
        ctx_label = df.iloc[idx]["label_text"]
        context_parts.append(f"[{ctx_label}] {ctx_title} - {ctx_abst[:300]}...")

    contexto = "\n\n".join(context_parts)

    rag_input = (
        "TEXTO ALVO:\n" + text + "\n\n"
        "ARTIGOS RELACIONADOS:\n" + contexto + "\n\n"
        "Com base no texto alvo e nos artigos relacionados, "
        "classifique o texto como 'AI' ou 'ML'."
    )

    result = zero_shot_classifier(
        rag_input,
        candidate_labels=candidate_labels,
        multi_label=False
    )
    preds_rag.append(result["labels"][0])

test_sample_rag["pred_rag"] = preds_rag

y_true_rag = test_sample_rag["label_text"].tolist()
y_pred_rag = test_sample_rag["pred_rag"].tolist()

print("=== RELATÓRIO RAG (Zero-Shot + FAISS + Contexto) ===")
print(classification_report(y_true_rag, y_pred_rag, digits=4))

print("MATRIZ DE CONFUSÃO (RAG)")
print(confusion_matrix(y_true_rag, y_pred_rag, labels=["AI", "ML"]))


from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

labels = ["AI", "ML"]

cm = confusion_matrix(y_true_rag, y_pred_rag, labels=labels)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)

plt.figure(figsize=(5, 4))
disp.plot(cmap="Blues")
plt.title("Matriz de Confusão (RAG)")
plt.show()
```

Comparativo de Performance (Resultados Finais)

F1 Macro Score

Fine-Tuning (SciBERT): **62.0%** de Acurácia / **0.615** F1 Macro

RAG (BART + Contexto): 57.5% de Acurácia / 0.563 F1 Macro

Zero-Shot (BART Puro): 49.5% de Acurácia / 0.482 F1 Macro

O Salto do RAG: O RAG aumentou o F1 Score em **8.1 pontos percentuais** em relação ao Zero-Shot puro.

O Trade-off

Escolhendo a Estratégia Certa

Para Performance Máxima: Fine-Tuning (SciBERT). Se você tem GPU / Dados Rotulados. Para Eficiência / Contexto: RAG (Embeddings + Zero-Shot). Se você não tem dados suficientes / Precisa de agilidade. Para Baseline / Tarefa Genérica: Zero-Shot. Se custo zero é a prioridade.

Aplicações no Mundo Real

Onde o Experimento se Aplica?

Setor Financeiro (RAG): Uso: Classificação de documentos legais, relatórios de risco ou notícias de mercado. Benefício: RAG garante contexto e precisão com corpus regulatório interno sem Fine-Tuning constante.

Setor de Saúde/Farmacêutico (Fine-Tuning/RAG): Uso: Classificação rápida de ensaios clínicos, patentes ou artigos de pesquisa. Benefício: Fine-Tuning para o vocabulário médico/científico aumenta a precisão.

Manufatura/Engenharia (RAG): Uso: Organização e catalogação de tickets de suporte ou relatórios de falha de equipamentos. Benefício: RAG classifica novos problemas com base no histórico e vocabulário técnico de falhas passadas.

Alunos envolvidos

Jefferson Eduardo da Silva - RA 10751463

Newton Alves Peixoto Neto - RA 10751466

Gustavo Lima de Carvalho - RA 10751465

Luiz Carlos Grandisoli - RA 10756587

Matheus Inoue - RA 10751469

Paula Rodrigues dos Santos - RA 10751464

Mais detalhes aqui…

Dúvidas estamos a disposição.

Obrigado!