NLP TB_3 | Legislação da UFAM

Ágata Clícia Santos Brazão
Ricardo Nogueira Miranda Filho

Download e Pré-Processamento dos Dados

Ao verificar o formato dos arquivos e as condições que se encontravam, foi possível observar que os documentos tratam-se de PDFs, alguns escaneados e outros digitais. Para capturar o texto dos PDFs foi utilizado a biblioteca e o texto foi registrado em arquivos .txt.

Para extrair as informações desejadas dos documentos, foi utilizado a API da OpenAI, com o seguinte prompt:

```
for parte in partes_texto:
    prompt = (
        "Você é um assistente que ajuda a pré-processar dados legislativos. "
        "Por favor, extraia e organize as informações legislativas do seguinte texto. "
        "Certifique-se de remover conteúdo irrelevante, normalizar o texto e estruturá-lo "
        "em seções claras. Inclua apenas informações importantes e relevantes sobre as leis, "
        "resoluções e normas mencionadas no texto.\n\n"
        f"{parte}"
    )
```

As configurações utilizadas para o modelo responsável por fazer o pré-processamento dos dados levou em conta o tamanho das partes de texto e a necessidade do modelo se ater às informações do texto, justificando a baixa temperatura.

```
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "Você é um assistente especializado em pré-processamento de dados legislativos."},
        {"role": "user", "content": prompt}
    ],
    max_tokens=2000,
    temperature=0.2
)
```

Devido a limitações no uso da API, fez-se necessário quebrar alguns textos para não ultrapassar o máximo de tokens permitidos por operações. Para isso foi criada a seguinte função:

```
[5]  def dividir_texto(texto, max_tokens=4000):
         partes = []
         palavras = texto.split()
         parte_atual = []

         for palavra in palavras:
             parte_atual.append(palavra)
             if len(" ".join(parte_atual)) > max_tokens:
                 partes.append(" ".join(parte_atual))
                 parte_atual = []

         if parte_atual:
             partes.append(" ".join(parte_atual))

         return partes
```

Dividindo os textos quando necessário em partes de no máximo 4000 tokens, embora o limite permitido pela API seja um pouco acima de 5000. Ao fim, o pré-processamento dos textos era realizado e então o texto pré-processado é inserido em uma lista para ao fim ser concatenado.

```
for texto, perguntas_necessarias in zip(textos_legislacao, proporcao_perguntas):
    texto_preprocessado = preprocessar_texto(texto)
    textos_preprocessados.append(texto_preprocessado)
    for _ in range(perguntas_necessarias):
        pergunta_resposta = gerar_pergunta_resposta(texto_preprocessado)
        base_de_dados.append(pergunta_resposta)
```

Geração de Base de Dados Sintética de Instruções

A medida que um documento é pré-processado, o conteúdo textual é passado como parâmetro para a função **gerar_pergunta_resposta**, que assim como a função de pré-processamento utiliza o modelo gpt-3.5-turbo. As configurações e o prompt seguem:


```
for texto, perguntas_necessarias in zip(textos_legislacao, proporcao_perguntas):
    texto_preprocessado = preprocessar_texto(texto)
    textos_preprocessados.append(texto_preprocessado)
    for _ in range(perguntas_necessarias):
        pergunta_resposta = gerar_pergunta_resposta(texto_preprocessado)
        base_de_dados.append(pergunta_resposta)
```

```
try:
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[
            {"role": "system", "content": "Você é um assistente que gera perguntas e respostas baseadas em texto legislativo."},
            {"role": "user", "content": prompt}
        ],
        max_tokens=400,
        temperature=0.7
    )
```

```
prompt = (
    "Baseado na seguinte legislação:\n\n"
    f"{texto_legislacao}\n\n"
    "Gere uma pergunta e uma resposta que sejam coerentes e relevantes. "
    "Certifique-se de que a pergunta seja clara e a resposta forneça uma explicação precisa e completa. "
    "Use apenas um '\\n' para separar a pergunta e a resposta na sua resposta, e não inclua nenhum texto adicional. "
    "A estrutura deve ser:\n"
    "Pergunta\nResposta"
)
```

O comando de separar a pergunta e a resposta por apenas um \n e não ter mais nenhum texto adicional na resposta é importante, para evitar a necessidade de realizar ajustes no conteúdo gerado ao final. A temperatura desta vez é 0.7, pois o modelo tem de ser mais criativo na elaboração das perguntas e respostas.

Para evitar perguntas repetidas, foi estabelecido que quanto maior o texto, mais perguntas ele vai gerar (decisão tomada após percebermos um grande volume de perguntas repetidas ao lidar com textos menores). Foi calculado o tamanho total de caracteres presentes no conjunto de PDFs, após isso verificou-se a quantidade de caracteres em cada PDF, a razão entre o total e a quantidade do PDF é responsável por indicar quantos pares de pergunta-resposta são gerados a partir dele.

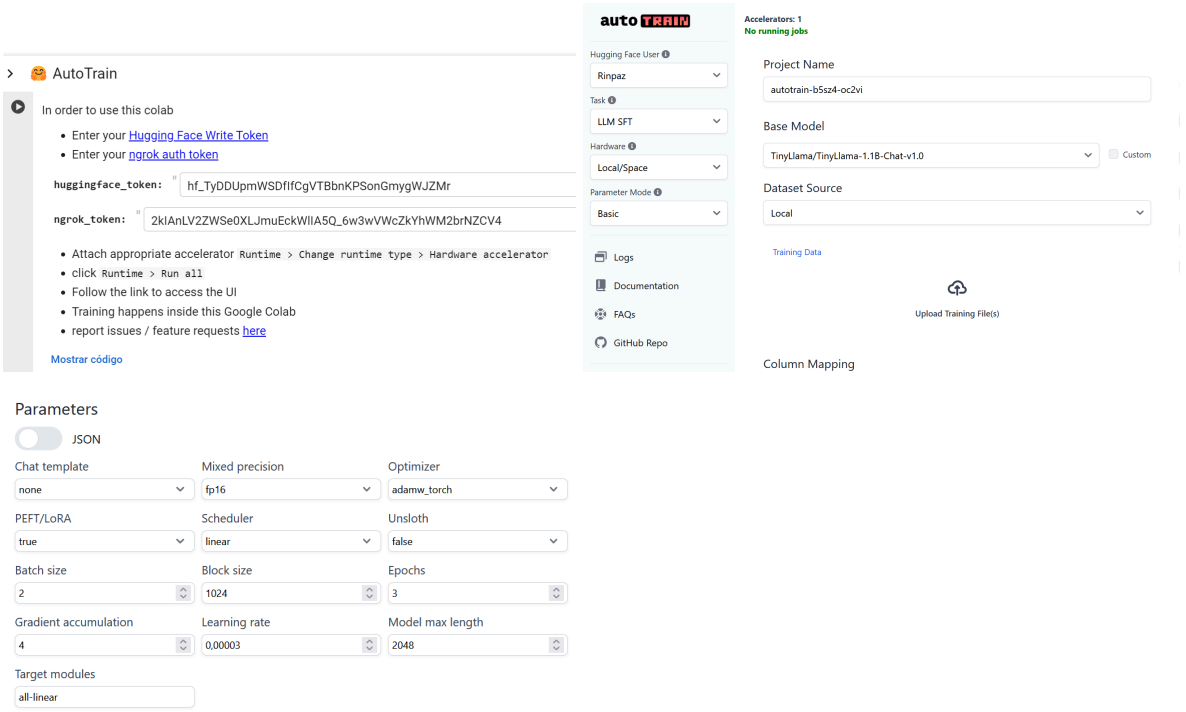
```
[29] base_de_dados = []
     total_perguntas = 1000
     textos_preprocessados = []
     texto_tamanho = [len(texto) for texto in textos_legislacao]

[30] soma_tamanhos = sum(texto_tamanho)
     proporcao_perguntas = [int((tamanho / soma_tamanhos) * total_perguntas) for tamanho in texto_tamanho]
```

Ao fim, depois de várias tentativas, os créditos da API acabaram (devido ao mau uso). Com isso, acabamos ficando com o conjunto de instruções gerados inicialmente apenas.

Treinamento do Modelo de Linguagem com LoRA

O modelo escolhido para ser treinado foi o TinyLlama . Para realizar o treino do modelo foi empregada a ferramenta AutoTrain, que foi executada via Google Colab.



Implementação de RAG (Retrieval-Augmented Generation)

Avaliação