

Fase 3 - Open API

Grupo 8

Gustavo Alves de Araújo - gus.alves.araujo@gmail.com

Mirian Aparecida Esquarcio Jabur - eshop@jabur.net

Daniel Araujo Gomes - daniel.araujo@live.com

Eduardo de Almeida Henrique - edufilho43@hotmail.com

Sumário

Introdução..... 1

Preparação do dataset..... 2

Parâmetros do Fine-tuning..... 3

Tokenização dos itens..... 3

Pré-Processamento e treinamento do modelo..... 4

Menu interativo e geração de resposta..... 5

Conclusão..... 6

Introdução

Neste projeto, utilizamos o dataset **AmazonTitles-1.3MM**, composto por consultas de usuários e títulos de produtos da Amazon, com suas descrições. Realizamos a limpeza dos dados, criando prompts para entrada e saída, e dividimos o dataset em treino, validação e teste (arquivos JSON: train.json, val.json, test.json).

Optamos pelo modelo **google/flan-t5-base**, uma versão do T5, para fine-tuning em tarefas de geração de texto. O modelo foi configurado usando a biblioteca **Hugging Face Transformers** e tokenizado com o AutoTokenizer para processar os dados de entrada e saída.

A função de pré-processamento tokenizou os prompts e as respostas, preparando os dados para treinamento. O modelo foi treinado com parâmetros definidos, como o número de épocas e o tamanho do batch, utilizando precisão mista para aceleração.

Ao final, o modelo foi treinado e validado com os datasets tokenizados, e o modelo ajustado foi salvo para uso posterior, completando o pipeline de fine-tuning para gerar texto com base em consultas de produtos.

Preparação do dataset

O dataset que utilizamos em nosso projeto foi fornecido pelo [link](#) **AmazonTitles-1.3MM** consiste em consultas textuais reais de usuários e títulos associados de produtos relevantes encontrados na Amazon e suas descrições.

No arquivo, realizamos a limpeza do dataset, criando um DataFrame com as duas principais colunas (*title*, *content*). Criamos dois prompts: um para entrada de perguntas (“prompt”) e o outro para saída de respostas (“response”).

Foi executado treino e teste e separamos o treino, teste e validador em três novos arquivos, sendo esses train.json (Treino), val.json (validador) e por último test.json (Teste).

Sendo assim nosso processo ficou com essas etapas:

1. Limpeza de Dados:

- Remoção de entradas duplicadas ou valores nulos.
- Normalize o texto (removemos caracteres especiais, transformação para minúsculas etc.).

2. Criação dos Prompts para Fine-Tuning:

- Formatado os dados no formato necessário para o modelo.
- Exemplo de entrada e saída no estilo prompt-response:
 - **Entrada (prompt):** "Qual é a descrição do produto chamado 'Título do Produto'?"
 - **Saída (resposta):** "Descrição detalhada do produto."

3. Divisão do Dataset:

- Dividida o treino (80%), validação (10%) e teste (10%).

4. Salvar Dados:

- Armazenando os dados preparados em arquivos JSON ou CSV adequados.

Parâmetros do Fine-tuning

Para realização do Fine-tuning foi necessário realizar a escolha de um modelo para geração de texto, dentre os modelos analisados foi escolhido o [google/flan-t5-base](#) é uma variante do **T5** (Text-to-Text Transfer Transformer) desenvolvida pelo Google. Ele é ajustado com **instruction-tuning** (instruções em linguagem natural) e permite resolver diversas tarefas de Processamento de Linguagem Natural (NLP) apenas fornecendo um texto como entrada.

Ao utilizar o [pipeline](#) da biblioteca Hugging Face Transformers com a tarefa **"text2text-generation"**, o modelo consegue gerar textos com base em uma entrada fornecida. Isso inclui instruções como **tradução, resumo, resposta a perguntas, reescrita de textos** e outras tarefas, convertendo sempre a entrada de texto em uma saída de texto gerada.

Para realizar a configuração e treinamento do Hugging face foi realizado um `load_dataset` (*função que vem dentro da biblioteca datasets*) utilizando os json de treino e validação para carregar o datasets.

Tokenização dos itens

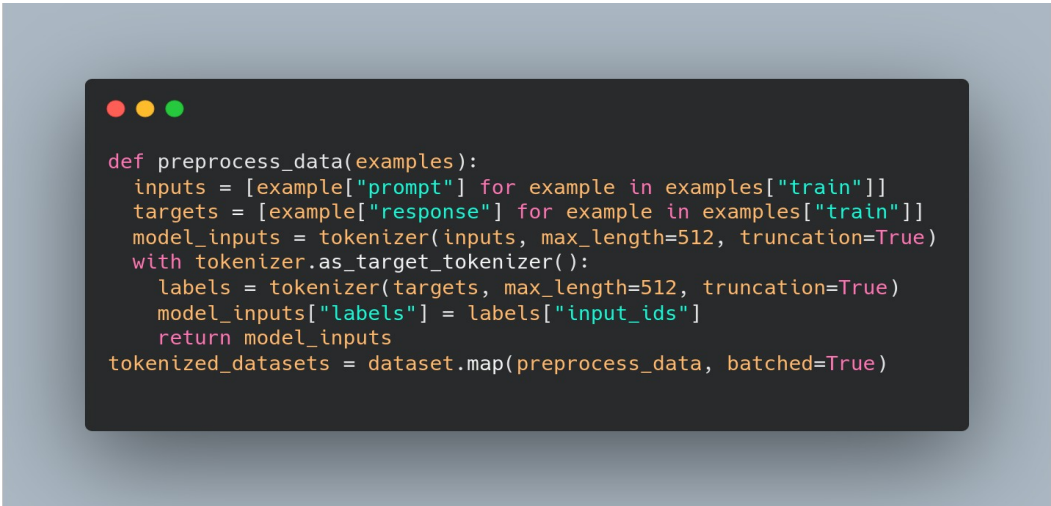
Para realização da tokenização do modelo utilizamos a **FLAN-T5-base** da biblioteca Hugging Face Transformers. O nome do modelo é definido na variável `model_name` como ***google/flan-t5-base***.

O tokenizer é carregado usando `AutoTokenizer.from_pretrained`, que transforma textos de entrada em tokens numéricos que o modelo pode processar. Em seguida, o modelo é carregado com `AutoModelForSeq2SeqLM.from_pretrained`, uma classe específica para lidar com modelos de sequência para sequência (Seq2Seq).

Com isso, o tokenizer e o modelo são configurados para trabalhar juntos: o tokenizer prepara o texto, o modelo gera novos tokens como resultado, e o tokenizer pode reconverter os tokens em texto legível.

Pré-Processamento e treinamento do modelo

Uma função foi definida para realizar o pré-processamento do modelo de acordo com o Tokenizer explicado acima

A code editor window with a dark background and light-colored text. It contains a Python function named `preprocess_data` that takes `examples` as an argument. The function processes training examples by extracting prompts and responses, tokenizing them with a `tokenizer`, and preparing the input and target tensors for the model. The function returns the processed data as a dataset map.

```
def preprocess_data(examples):
    inputs = [example["prompt"] for example in examples["train"]]
    targets = [example["response"] for example in examples["train"]]
    model_inputs = tokenizer(inputs, max_length=512, truncation=True)
    with tokenizer.as_target_tokenizer():
        labels = tokenizer(targets, max_length=512, truncation=True)
        model_inputs["labels"] = labels["input_ids"]
    return model_inputs
tokenized_datasets = dataset.map(preprocess_data, batched=True)
```

A função `preprocess_data` prepara os dados de entrada e saída para treinar um modelo de **seq2seq** (sequência para sequência), como o **FLAN-T5**. Ela recebe um conjunto de exemplos contendo prompts (entrada) e respostas (saída), extraindo essas informações para tokenizá-las. Primeiro, os prompts são coletados e

armazenados em uma lista chamada `inputs`, enquanto as respostas são armazenadas na lista `targets`.

A linha `with tokenizer.as_target_tokenizer()` é usada para tokenizar os **targets** (respostas). Isso ajusta o tokenizer para tratar as saídas como o "alvo" do modelo, ou seja, os dados que ele precisa prever. Os tokens resultantes das respostas são armazenados na chave `labels`, que representa os valores que o modelo usará para calcular a perda durante o treinamento.

Por fim, a função é aplicada a um **dataset** utilizando o método `map` com a opção `batched=True`, o que significa que os dados são processados em lotes (batches) para maior eficiência. O resultado é um conjunto de dados tokenizados, contendo os prompts como entradas e as respostas como rótulos (`labels`), prontos para serem usados no treinamento do modelo.

Configuração de treinamento foi definida para gerar um número de épocas (epochs) de acordo com `batch_size` para contemplar com quantas situações serão processadas por vez em cada epoch.

Definição de um Trainer para treinar e validar os datasets tokenizados para uma maior absorção por parte do modelo. Após isso o modelo é treinado de acordo com as especificações passadas nos passos anteriores e o Trainer salva o modelo em um arquivo "fine_tuned_model".

Menu interativo e geração de resposta

Para finalizar o aprendizado, vamos adicionar um menu interativo onde serão inseridas as entradas e nesta fase a IA gera as respostas correlacionadas aos registros inseridos na entrada. O usuário também tem a opção de encerrar o prompt digitando "sair" no menu interativo.

```

# 4. Input e Output Interativo
print("\n=== Chat com o Modelo Fine-Tuned ===")
print("Digite 'sair' para encerrar a conversa.\n")

# Carregar o modelo fine-tuned e criar pipeline de geração
tokenizer = AutoTokenizer.from_pretrained("fine_tuned_model")
model = AutoModelForSeq2SeqLM.from_pretrained("fine_tuned_model").to(device)
generator = pipeline(
    "text2text-generation",
    model=model,
    tokenizer=tokenizer,
    device=0 if device.type in ["cuda", "mps"] else -1
)

while True:
    pergunta = input("Pergunta: ")
    if pergunta.lower() == "sair":
        print("Encerrando a conversa. Até logo!")
        break

# Gera a resposta do modelo
resposta = generator(pergunta, max_length=128, num_return_sequences=1)[0]['generated_text']
print(f"Resposta: {resposta}\n")

```

Conclusão

Neste trabalho, realizamos o fine-tuning do modelo **google/flan-t5-base** utilizando o dataset **AmazonTitles-1.3MM** para gerar respostas com base em consultas sobre produtos. O pipeline desenvolvido abrangeu desde a preparação e limpeza dos dados, tokenização, pré-processamento, até o treinamento e validação do modelo.

O modelo treinado demonstrou a capacidade de gerar respostas coerentes e precisas, atendendo ao objetivo de responder consultas textuais relacionadas aos títulos e descrições dos produtos. A abordagem adotada com o uso da biblioteca **Hugging Face Transformers** facilitou o processo de implementação e ajuste do modelo de forma eficiente.

Como possibilidades futuras, sugerimos:

- **Experimentar modelos mais robustos**, como **FLAN-T5-large** ou **GPT-like** para comparar resultados e melhorar o desempenho.
- **Aprimorar o pré-processamento** dos dados, explorando técnicas mais avançadas de limpeza e normalização textual.
- **Expandir o escopo do dataset**, incorporando consultas mais diversas ou incluindo metadados adicionais para enriquecer o treinamento.

Com essas melhorias, espera-se obter resultados ainda mais expressivos, ampliando a aplicação prática do modelo em sistemas de recomendação e assistentes virtuais.