

Fine-tuning de um foundation model utilizando o dataset "The AmazonTitles-1.3MM"

1. Introdução

O Processamento de Linguagem Natural (PLN) é uma área da inteligência artificial que se concentra em permitir que máquinas compreendam e interajam com a linguagem humana de forma natural. Com o crescimento exponencial de dados textuais provenientes de diversas fontes, como redes sociais, e-commerce e comunicações digitais, o PLN tornou-se fundamental para extrair informações relevantes e aprimorar a interação entre humanos e máquinas.

Modelos de linguagem avançados, baseados em arquiteturas de redes neurais profundas como os Transformers, revolucionaram o PLN. Esses modelos, quando pré-treinados em grandes volumes de dados textuais, podem ser adaptados para tarefas específicas por meio do fine-tuning, refinando seus parâmetros para melhorar o desempenho em contextos particulares.

2. Objetivo

Realizar o fine-tuning com o dataset “The AmazonTitles-1.3MM” em um modelo pré treinado para que sejam geradas respostas relevantes no contexto a partir do prompt formado pela pergunta do usuário sobre títulos de produtos da Amazon.

3. Preparação do dataset

O dataset **AmazonTitles-1.3MM** contém consultas reais de usuários, títulos de produtos e suas descrições na Amazon, totalizando aproximadamente 3.218.856 registros (soma de **trn.json** e **tst.json**).

Para este projeto, utilizamos os campos: **title** (título do produto) e **content** (descrição do produto).

O processo de preparação e formatação do dataset está dentro da função **process_and_format_dataset** apresentada logo no início do projeto e que receberá o arquivo **trn.json** e **tst.json** para executar as seguintes instruções:

- Carregar cada linha como um objeto JSON;
- Ignorar as linhas com erros de decodificação;
- Converter a lista de dicionários em um DataFrame do pandas;
- Remover registros com valores nulos dos campos title e content;
- Remover registros com strings vazias ou apenas espaços em branco dos campos title e content;
- Remover registros duplicatas com base nos campos title e content;
- Reiniciar o índice do DataFrame;
- Normalizar o texto removendo caracteres de controle e espaços extras;
- Criar as colunas **instruction**, **input** e **output** para o formato necessário;
- Pegar uma amostra do DataFrame se o **sample_size** estiver definido;
- Converter para uma lista de dicionários;
- Gerar um arquivo JSON com os resultados;

4. Resultados e análise da preparação do dataset

- **Processando o dataset de treinamento trn.json**
 - Total de registros no arquivo: 2.248.619
 - Registros com 'title' e 'content': 2.248.619
 - Registros após criar DataFrame: 2.248.619
 - Registros após remover nulos: 2.248.619
 - Registros após remover strings vazias: 1.390.403
 - Registros após remover duplicatas: 1.367.131
 - **Total de registros após processamento: 1.367.131**
- **Processando o dataset de teste (tst.json):**
 - Total de registros no arquivo: 970.237
 - Registros com 'title' e 'content': 970.237
 - Registros após criar DataFrame: 970.237
 - Registros após remover nulos: 970.237
 - Registros após remover strings vazias: 599.743
 - Registros após remover duplicatas: 593.744
 - **Total de registros após processamento: 593.744**
- **Remoção de valores nulos ou vazios:**
 - Eliminados registros com campos vazios em **title** ou **content**, reduzindo de **3.218.856** para **1.990.146** registros (soma de **trn.json** com **1.390.403** e **tst.json** com **599.743** registros).
- **Remoção de duplicatas:**
 - Eliminados registros duplicados para evitar redundâncias, resultando em **1.960.875** registros após a limpeza (soma de **trn.json** com **1.367.131** e **tst.json** com **593.744** registros).

5. Output do arquivo formatado

Após executadas todas as etapas de preparação e formatação do dataset a função **process_and_format_dataset** gera um novo arquivo .JSON no diretório definido pelo parâmetro **output_file_path** e com a quantidade de registros limitada se definida no parâmetro **sample_size**.

6. Escolha e configuração do Foundation Model

Geração de respostas baseadas em perguntas e limitações de hardware, como memória GPU e a conformidade com as restrições de uso do modelo são as principais premissas para a escolha de um foundation model adequado para o projeto. Entre os modelos mais populares podemos destacar os seguintes:

- **BERT** que é ótimo para compreensão de linguagem mas não é ideal para geração de textos;
- **Llama** adequado para geração de textos;
- **Mistral** adequado para geração de textos;
- **LlaMA** adequado para geração de textos.

O foundation model selecionado foi o **LlaMA** pela sua característica de geração de textos, baixa utilização de memória e desempenho 2x mais rápido.

7. Fine-tuning

Em tradução livre, ajuste fino, é o processo de continuar o treinamento de um modelo pré-treinado sobre um novo conjunto de dados com uma tarefa específica em mente. É o ajuste dos parâmetros do modelo para que ele se torne mais especializado em uma tarefa, mas sem perder as capacidades gerais aprendidas durante o treinamento inicial.

O fine-tuning será executado na instância gratuita **T4** do **Google Colab** com o a biblioteca **Unsloth** com o modelo open-source **Llama-3.1-8B**, um modelo quantizado para 4bits para reduzir o consumo de memória e permitir o treinamento do modelo em GPUs com menos vRAM ou gratuitas como a Tesla 4.

Também serão utilizadas bibliotecas adicionais como **torch**, **peft**, **pandas**, **xformers**, **triton**, **accelerate**, **bitsandbytes** e **dataset**.

8. Configuração do modelo

Para iniciar a configuração do modelo é preciso definir a quantidade de tokens, o tipo de dados, a quantização e o modelo na função **from_pretrained** da classe **FastLanguageModel**.

Parâmetro	Valor	Informações
max_seq_length	128	Comprimento máximo das sequências de entrada.
dtype	None	Definição de tipo de dados (também pode ser torch.float16 ou torch.float32).
load_in_4bit	True	Quantização do modelo em 4bits para economia de memória.
model_name	Unsloth/Meta-Llama-3.1-8B	Modelo quantizado para 4bits que permite o treinamento em GPUs com menos VRAM com a T4 .

Com o objetivo de reduzir a carga computacional e o uso de memória durante o treinamento, o **PEFT** é utilizado e o **LoRA** como espectro método para modificar apenas as camadas lineares do modelo, inserindo transformações de baixo posto que são treinadas enquanto outros pesos são mantidos congelados. Isso permite que o modelo seja adaptado para uma tarefa específica com um pequeno acréscimo nos parâmetros, mantendo a maioria do conhecimento pré-treinado intacto e reduzindo a quantidade de dados necessários para efetuar o ajuste.

Essa configuração é realizada na função **get_peft_model** da classe **FastLanguageModel**.

Parâmetro	Valor	Informações
r	16	Rank das matrizes LoRA podendo ser 8,16, 32, 64 e 128. Quanto menor o valor, menos custo computacional e menos poder de ajuste.
target_modules	["q_proj", "k_proj", "v_proj", "o_proj", "gate_proj", "up_proj", "down_proj",]	Modelos que serão adaptados
Lora_alpha	16	Controla a escla de atualização das matrizes LoRA
Lora_dropout	0	Ajuda a prevenir overfitting em tarefas específicas, onde 0 é otimizado e valores como 0.1 ou 0.2 podem ser úteis para baixo volume de dados.
Bias	"none"	Define como o viés será tratado nos módulos.
use_gradient_checkpointing	"unsloth"	Reduz o uso de memória VRAM ao recalculr alguns valores intermediários durante o backpropagation.
random_state	3407	Define uma semente aleatória para garantir reprodutibilidade.

O **Prompt** é um template consistente que ajuda o modelo a entender melhor a estrutura da tarefa que será solicitada. A estrutura é utilizada para tarefas de treinamento supervisionado ou geração de texto fornecendo uma instrução, um contexto oadicional e um espaço para a resposta.

Parâmetro	Valor	Informações
alpaca_prompt	"""Below is an instruction that describes a task, paired with an input that provides further context. Write a response that appropriately completes the request."""	Instrução do prompt, explica ao modelo o que esperar e como estruturar a resposta.
instruction	{ }	Instrução recebida do usuário.
input	{ }	Entrada adicional, se necessária.
response	{ }	Resposta fornecida pelo model.

A função **formatting_prompts_func** formata o texto que será exibido ao usuário utilizando o dicionário **examples** com a lista de instruções. O parâmetro **tokenizer.eos_token** deve ser adicionado ao final do prompt para indicar o fim da sequência do texto formatado.

9. Inferência

A inferência em um modelo de Linguagem de Grande Escala (LLM, *Large Language Model*) refere-se ao processo de preparar o modelo pré-treinado com as configurações customizadas para fornecer saídas baseadas em entradas fornecidas pelo usuário. Em outras palavras, é o ato de utilizar o modelo para executar tarefas específicas após ele já ter sido treinado.

10. Teste de prompt antes do fine-tuning com o dataset formatado

Para certificar a eficiência do fine-tuning que será realizado, vamos executar um teste com uma pergunta ao modelo configurado mas sem o treinamento com o dataset formatado gerando assim uma resposta genérica.

```
### Instruction:
Answer the question based on the product title.

### Input:
What is 'Girls Ballet Tutu Neon Pink'?

### Response:
This is a ballet tutu in neon pink.
```

11. Mapeamento do Dataset

Através da biblioteca **load_dataset** o caminho do dataset treinado é informado e os registros são carregados e formatados com a função **formatting_prompts_func** utilizando a propriedade **.map**.

12. Treinamento do modelo

Com a classe **TrainingArguments** da biblioteca **transformers** da Hugging Face serão configurados os hiperparâmetros e opções relacionadas ao treinamento que serão usadas pela classe **SFTTrainer** (Supervised Fine-Tuning Trainer) da biblioteca **trl** para realizar o fine-tuning supervisionado com o dataset formatado e as configurações já aplicadas ao modelo.

Parâmetros a classe **TrainingArguments**

Parâmetro	Valor	Informações
per_device_train_batch_size	1	Tamanho do batch por GPU, 1 para evitar falta de memória
gradient_accumulation_steps	2	Passos de acumulação de gradientes antes de atualizar os pesos do modelo.
#num_train_epochs	1	Épocas de treinamento, desabilitado nesse caso, pois o número de passos já está limitado pelo max_steps

max_steps	60	Número máximo de passos de treinamento a serem executados
learning_rate	2e-4	Taxa de aprendizado inicial do otimizador.
fp16	Not is_bfloat16_supported()	Meio-precisão (16 bits) para economia de memória.
bf16	Is_bfloat16_supported()	Formato bfloat16 (mais estável em algumas arquiteturas de hardware)
logging_steps	1	Frequência (em passos) com que as métricas de log são registradas.
optim	“adamw_8bit”	Versão de Adam otimizada para economizar memória usando pesos de 8bits.
weight_decay	0.01	Penalidade aplicada aos pesos para evitar overfitting.
lr_scheduler_type	Linear	Reduz a taxa de aprendizado de forma linear ao longo do treinamento
seed	3407	Controle de aleatoriedade garantindo a reprodutibilidade.
output_dir	“content/drive/MyDrive”	Diretório onde os resultados do treinamento e checkpoints serão armazenados.
report_to	“none”	Relatório e monitoramento do treinamento.
evaluation_strategy	“no”	Sem avaliação durante o treinamento.
save_strategy	“no”	Não salva checkpoints intermediários.

Parâmetros da classe **SFTTrainer**

Parâmetro	Valor	Informações
model	model	Modelo escolhido para executar o fine-tuning. Definido anteriormente em FastLanguageModel.from_pretrained()
tokenizer	tokenizer	Tokenizador do modelo que transforma texto em tensores numéricos. Definido anteriormente em FastLanguageModel.from_pretrained()
dataset_text_field	“text”	Nome do campo no dataset que contém o texto que será usado no treinamento
max_seq_length	128	Número máximo de passos de treinamento a serem executados
dataset_num_proc	2	Número de processos para carregar e processar os dados do dataset.
packing	False	Para fazer o treinamento 5x mais rápido com pequenas sequências
Args	args	Argumentos para o treinamento. Definido anteriormente em TrainingArguments()

13. Resultado do treinamento

Step	Training Loss	Step	Training Loss	Step	Traning Loss
1	3.275900	21	2.055200	41	1.074100
2	2.798100	22	1.820000	42	1.792600
3	2.599200	23	2.101500	43	1.667600
4	2.883600	24	1.621600	44	1.403900
5	2.855200	25	1.708600	45	1.628400
6	2.237300	26	1.526900	46	2.413900
7	2.091400	27	1.867900	47	1.220400
8	1.953700	28	1.540400	48	1.708000
9	2.020900	29	1.702100	49	1.832500
10	1.964400	30	1.619200	50	2.019200
11	1.937600	31	1.874000	51	0.871900
12	1.608800	32	1.511500	52	1.750000
13	1.524700	33	1.787100	53	2.017300
14	1.692300	34	1.497700	54	2.064300
15	1.977500	35	1.838700	55	1.861800
16	1.514700	36	1.911400	56	1.887800
17	1.863100	37	1.148400	57	1.741900
18	1.369300	38	1.405800	58	1.619800
19	1.021600	39	1.166900	59	1.542600
20	1.479700	40	1.433200	60	1.040100

```
TrainOutput(global_step=60, training_loss=1.7827455222606658, metrics={'train_runtime': 84.3205, 'train_samples_per_second': 1.423, 'train_steps_per_second': 0.712, 'total_flos': 643762909569024.0, 'train_loss': 1.7827455222606658, 'epoch': 0.0012})
```

14. Exportação do modelo

É importante exportar o modelo e o tokenizador treinados para que seja possível utilizá-lo futuramente sem a necessidade de realizar o fine-tuning novamente.

Para que seja possível salvar esses modelos a biblioteca **Huggin Face Transformers** possui a classe **.save_pretrained("/content/drive/MyDrive/trained_model")**.

15. Utilização do modelo com fine-tuning

Para utilizar o modelo treinado a qualquer momento basta chamar novamente a função **from_pretrained** da classe **FastLanguageModel** alterando somente o parâmetros **model_name** onde deve ser informado o caminho do modelo treinado e adicionado o parâmetro **device_map="auto"** para distribuição automática do modelo carregado em diferentes dispositivos de hardware, como GPUs ou CPUs otimizando assim o hardware disponível. Depois de configurados os parâmetros execute uma nova inferência no modelo com a função **for_inference(model)** também da classe **FastLanguageModel** e o modelo estará pronto para receber as perguntas do usuário.

16. Teste de prompt depois do fine-tuning

Para certificar a eficácia do fine-tuning executado, um teste com uma pergunta é realizado ao modelo configurado gerando assim uma resposta relevante ao contexto.

O tamanho da resposta é definido no parâmetro **max_new_tokens** da função **.generate()**.

```
### Instruction:
Answer the question based on the product title.

### Input:
What is 'Girls Ballet Tutu Neon Pink'?

### Response:
A tutu is a type of skirt, usually made of tulle, worn by ballet dancers. It is also used in other dance genres, such as jazz, modern, hip-hop, and contemporary.
```

```
### Instruction:
Answer the question based on the product title.

### Input:
Who was Ayrton Senna?

### Response:
Ayrton Senna da Silva (21 March 1960 - 1 May 1994) was a Brazilian racing driver and three-time Formula One World Champion. He is the third most successful F1 driver of all time in terms of race wins, behind Michael Schumacher and Alain Prost. Senna is considered by many to be one of the greatest drivers of all time. He won the title in 1988, 1990 and 1991 in a McLaren-Honda, and is the only driver to win the title in a car manufactured by a company based in Brazil. Senna's death in a car accident at the 1994 San Marino Grand Prix led to a change of the rules to improve safety. In 1999, he was posthumously elected to the FIA Hall of Fame
```

17. Links

- Youtube
<https://youtu.be/loluRca9OXQ>
- Projeto no GitHub
<https://github.com/victorluongo/fiap-techchallenge-fase3>
- Projeto no Google Colab
<https://colab.research.google.com/drive/1pRha6NOq7lnUdK4m2Mvznk5Qot7d4Lzz?usp=sharing>

18. Conclusão

- **Resultados Alcançados:** Desenvolvemos um modelo capaz de responder a perguntas sobre produtos da Amazon de forma relevante e informativa, com melhorias significativas na execução.

- **Limitações Enfrentadas:** Restrições computacionais impediram a experimentação com modelos maiores e ajustes mais agressivos.
- **Aprendizados:** Evidenciamos a importância do ajuste fino de hiperparâmetros, do uso eficiente dos recursos e da qualidade dos dados de treinamento.