Processamento de Dados e Treinamento de Modelos para Tradução Automática entre Português e Tupi Antigo

Calebe Rezende

Setembro 2024

Resumo

Este trabalho tem como objetivo desenvolver um modelo de tradução automática para o Tupi Antigo, utilizando técnicas de processamento de linguagem natural. A pesquisa se concentra na coleta e organização de um corpus linguístico e na implementação de um pipeline que inclui tokenização e treinamento de um modelo baseado na arquitetura de transformadores, com foco na preservação e revitalização linguística.

Sumário

1 Introdução

Este projeto tem como objetivo desenvolver recursos linguísticos modernos para línguas indígenas sem falantes nativos vivos, com foco específico no desenvolvimento de chatbots para o Tupi Antigo (também conhecido como Língua Brasílica).

O Tupi Antigo foi a língua predominante ao longo do litoral brasileiro durante os séculos XVI, XVII e parte do XVIII, sendo amplamente empregada pelos jesuítas na evangelização das populações indígenas. Contudo, em meados do século XVIII, o uso da língua foi oficialmente proibido por determinação do Marquês de Pombal, pouco antes da expulsão dos jesuítas das colônias portuguesas. Estima-se que, ao final do século XIX, não havia mais falantes nativos do Tupi Antigo. Atualmente, o que restou dessa língua são poucos textos, em sua maioria escritos por jesuítas nos primeiros séculos de colonização. Esse acervo limitado, aliado à falta de digitalização dos documentos existentes, apresenta

um grande desafio para o desenvolvimento de ferramentas computacionais. Como parte do processo de preparação de dados, é necessária a digitalização por meio de tecnologia OCR (Reconhecimento Óptico de Caracteres) para possibilitar a comparação e análise dos textos disponíveis.

Para o desenvolvimento de tradutores automáticos, propomos uma metodologia baseada na existência de línguas proximais, para as quais há uma quantidade mais significativa de textos e recursos disponíveis. A metodologia das Línguas Proximais segue os seguintes passos:

Identificar uma língua próxima do ponto de vista linguístico, para a qual exista um maior número de textos e ferramentas; Utilizar tradutores já existentes entre o português e essa língua proximal; No passo final, realizar a tradução entre a língua proximal e o Tupi Antigo. Atualmente, investigamos duas línguas proximais com maior disponibilidade de textos e recursos:

Guarani Paraguaio, uma língua oficial do Paraguai, conhecida por ser derivada do Tupi Antigo. É a língua indígena americana mais falada, utilizada tanto no Paraguai quanto em comunidades vizinhas. Existem tradutores disponíveis entre o espanhol e o Guarani Paraguaio, que podem ser aproveitados para nossos objetivos.

Nheengatu, uma língua falada na região do Rio Negro, entre os povos Baniwa, Baré e Warekena. Há um projeto em andamento que visa o desenvolvimento de um tradutor entre o português e o Nheengatu.

Até o momento, construímos um conjunto de dados de textos em Tupi Antigo com suas respectivas traduções para o português. Os próximos passos envolvem a obtenção e o teste dos tradutores para Nheengatu e Guarani Paraguaio.

2 Objetivos

O principal objetivo desta pesquisa é desenvolver um modelo de tradução automática...

3 Revisão da Literatura

3.1 A Importância do Tupi Antigo

O Tupi Antigo foi a língua franca no Brasil colonial e influenciou profundamente a cultura...

3.2 Modelos de Tradução Automática

Nos últimos anos, os modelos de tradução automática baseados em redes neurais...

4 Metodologia

Introdução

O arquivo descreve um processo de **treinamento de um modelo de tradução automática utilizando o modelo T5**, especificamente para o caso de tradução entre o

português e o Tupi Antigo. Abaixo está uma descrição técnica e acadêmica dos processos envolvidos.

1. Carregamento e Tokenização dos Dados

O primeiro passo envolve o carregamento de um arquivo JSON contendo dados formatados de Tupi Antigo e suas respectivas traduções para o português. O arquivo é carregado usando a biblioteca `json` e passa por uma verificação para garantir que os dados estejam corretamente formatados. A partir desses dados, utiliza-se o `T5Tokenizer` para tokenizar as entradas textuais. A tokenização transforma as sentenças em sequências de números que representam tokens, que podem ser processados pelo modelo de aprendizado profundo.

```
tokenizer = T5Tokenizer.from_pretrained("t5-small")
model = T5ForConditionalGeneration.from_pretrained("t5-small")
formatted_data = json.load(f)
```

2. Criação de um Dataset Customizado

Após a tokenização, um dataset customizado é criado para ser utilizado pelo `Trainer` da biblioteca `transformers`. Esse dataset é uma classe personalizada (`CustomDataset`) que armazena os dados tokenizados e formata as amostras de acordo com a necessidade do modelo, fornecendo os `input_ids`, `attention_mask`, e os rótulos (`labels`), que são os textos-alvo para a tradução.

```
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, inputs, targets):
        self.inputs = inputs
        self.targets = targets
    def __len__(self):
        return len(self.inputs["input_ids"])
    def __getitem__(self, idx):
        return {
            "input_ids": self.inputs["input_ids"][idx],
            "attention_mask": self.inputs["attention_mask"][idx],
            "labels": self.targets["input_ids"][idx]
        }
```

3. Definição dos Argumentos de Treinamento

O treinamento do modelo é configurado com uma série de parâmetros e estratégias específicas através da classe `TrainingArguments`. Entre os parâmetros mais importantes estão:

- **Taxa de aprendizado** (`learning_rate`): Define a taxa de atualização dos pesos durante o processo de otimização. - **Tamanho do batch** (`per_device_train_batch_size`): Número de amostras processadas por vez durante o treinamento. - **Número de épocas**

(`num_train_epochs`): Quantidade de vezes que o conjunto de dados completo será passado pelo modelo. - **Decay de pesos (`weight_decay`)**: Técnica de regularização usada para evitar overfitting.

```
training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=8,
    num_train_epochs=3,
    weight_decay=0.01,
    save_steps=10_000,
    save_total_limit=2,
)
```

4. Treinamento com Gradient Accumulation e Precision Mista

Para otimizar o uso de recursos computacionais, o código utiliza **acumulação de gradientes** e **precisão mista**. A acumulação de gradientes permite que o modelo seja treinado em batches menores, acumulando gradientes ao longo de vários passos antes de fazer uma atualização nos pesos, economizando memória. A precisão mista (`autocast`) melhora o desempenho do treinamento ao utilizar menos precisão (16 bits) para algumas operações, sem comprometer a qualidade do treinamento.

```
with autocast():
    outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels
    loss = outputs.loss / accumulation_steps
scaler.scale(loss).backward()
```

5. Utilização do Trainer para Fine-Tuning

O processo de treinamento é gerenciado pelo `Trainer`, uma classe da biblioteca `transformers` que facilita a execução do fine-tuning em modelos pré-treinados. Ele cuida da otimização, agendamento de aprendizado e checkpoints de salvamento automático. O modelo é treinado em várias épocas, e os pesos são ajustados com base nos dados fornecidos.

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
)
trainer.train()
```

6. Salvamento do Modelo e Tokenizador

Após o treinamento, o modelo e o tokenizador são salvos no Google Drive para uso futuro. O salvamento é feito utilizando o método `save_pretrained()`, que armazena os pesos treinados do modelo e os vocabulários gerados pelo tokenizador.

```
save_directory = '/content/drive/MyDrive/Tupi Antigo/t5_fine_tuned_model/'
model.save_pretrained(save_directory)
tokenizer.save_pretrained(save_directory)
```

7. Conclusão

Este processo representa uma abordagem completa para o treinamento de um modelo de tradução automática utilizando o T5 pré-treinado, ajustado para um novo par de línguas com poucos recursos disponíveis, como o Tupi Antigo e o português. A estratégia de utilização de técnicas de otimização, como acumulação de gradientes e precisão mista, permite treinar o modelo de forma eficiente, mesmo em ambientes com recursos computacionais limitados.