Documentação: Aplicação do Modelo BERT para Detecção de Bots no Twitter

1. Introdução

Este documento descreve o processo de implementação de um modelo BERT para a detecção de bots no Twitter. O objetivo é utilizar tweets como entradas e classificar automaticamente se um determinado tweet foi postado por um bot ou por um humano. O processo é dividido em etapas que incluem carregamento de dados, preparação dos mesmos, treinamento e avaliação do modelo.

2. Carregamento dos Dados

Nesta etapa, o dataset contendo os tweets e suas classificações (bot ou não) é carregado diretamente do Google Drive para o ambiente de execução.

```python
from google.colab import drive
import pandas as pd

# Monta o Google Drive para carregar o arquivo
drive.mount('/content/drive')

# Carrega o arquivo CSV
df = pd.read_csv('/content/drive/MyDrive/Atividades M11 DADOS/bot_detection_data.csv')

# Verifica as primeiras linhas do dataset
df.head()
```

3. Preparação dos Dados

Os dados precisam ser tokenizados para serem processados pelo modelo BERT. Usamos o tokenizer pré-treinado do BERT para converter os tweets em tokens e, em seguida, ajustamos os rótulos.

```python
[ ] from sklearn.model_selection import train_test_split

# Divide os dados em 80% treino e 20% teste
train_inputs, test_inputs, train_labels, test_labels = train_test_split(input_ids, labels, test_size=0.2, random_state=42)
train_masks, test_masks, _, _ = train_test_split(attention_masks, labels, test_size=0.2, random_state=42)
```

4. Divisão dos Dados em Treinamento e Teste

Os dados são divididos em conjuntos de treinamento e teste, mantendo 80% dos dados para treino e 20% para teste.

```
[ ] from torch.utils.data import DataLoader, TensorDataset, RandomSampler, SequentialSampler

    # Cria datasets para treino e teste
    train_data = TensorDataset(train_inputs, train_masks, train_labels)
    test_data = TensorDataset(test_inputs, test_masks, test_labels)

    # DataLoader para pegar os dados em lotes
    train_dataloader = DataLoader(train_data, sampler=RandomSampler(train_data), batch_size=32)
    test_dataloader = DataLoader(test_data, sampler=SequentialSampler(test_data), batch_size=32)
```

5. Criação de DataLoaders

DataLoaders são configurados para carregar os dados em pequenos lotes durante o treinamento e a avaliação.

```
from transformers import BertForSequenceClassification

# Carrega o modelo BERT com uma camada de classificação para 2 classes
model = BertForSequenceClassification.from_pretrained(
    'bert-base-uncased',
    num_labels=2,  # Tarefa binária (bot ou não bot)
    output_attentions=False,
    output_hidden_states=False
)

# Move o modelo para a GPU, se disponível
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

6. Configuração do Modelo e Otimizador

O modelo BERT é carregado com uma camada extra para a tarefa de classificação binária. O otimizador AdamW e a função de perda CrossEntropyLoss são configurados, juntamente com um scheduler para ajustar a taxa de aprendizado.

```
[ ] from transformers import AdamW

    # Configura o otimizador AdamW
    optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:591: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation tc
  warnings.warn(
```

7. Treinamento do Modelo

O treinamento é feito utilizando uma época para fins experimentais. Parte das camadas do BERT é congelada para reduzir o tempo de treinamento, enquanto a precisão mista otimiza o uso da GPU. Apenas uma época é utilizada para economizar tempo, dado o alto custo computacional do BERT (cada época está demorando em média duas horas para ser rodado). Parte das camadas é congelada para acelerar o processo, e precision mista otimiza o uso da GPU.

```python
import torch
from torch.nn import CrossEntropyLoss
from torch.optim import AdamW
from tqdm import tqdm
from transformers import get_linear_schedule_with_warmup

# Função de perda
loss_fn = CrossEntropyLoss()

# Número de épocas
epochs = 1

# Otimizador e scheduler
optimizer = AdamW(model.parameters(), lr=5e-5)
total_steps = len(train_dataloader) * epochs

scheduler = get_linear_schedule_with_warmup(optimizer, num_warmup_steps=0, num_training_steps=total_steps)

# Configuração de precision mista para acelerar
scaler = torch.cuda.amp.GradScaler()

# Congelando as primeiras camadas do BERT para acelerar
for param in model.bert.encoder.layer[:8].parameters():
    param.requires_grad = False

# Loop de treinamento
for epoch in range(epochs):
    model.train()
    total_loss = 0
```

```python
    for batch in tqdm(train_dataloader):
        b_input_ids, b_input_mask, b_labels = tuple(t.to(device) for t in batch)

        # Zera os gradientes
        optimizer.zero_grad()

        with torch.cuda.amp.autocast():
            outputs = model(b_input_ids, attention_mask=b_input_mask, labels=b_labels)
            loss = outputs.loss

        # Acumula a perda
        total_loss += loss.item()

        # Backpropagation com precision mista
        scaler.scale(loss).backward()

        # Atualiza os parâmetros
        scaler.step(optimizer)
        scaler.update()
        scheduler.step()

    print(f"Epoch {epoch+1}/{epochs} - Loss: {total_loss/len(train_dataloader)}")
```

```
<ipython-input-7-dc2036bb6ca3>:20: FutureWarning: `torch.cuda.amp.GradScaler(args...)` is deprecated. Please use `torch.amp.GradScaler('cuda', args...)` instead.
  scaler = torch.cuda.amp.GradScaler()
/usr/local/lib/python3.10/dist-packages/torch/amp/grad_scaler.py:132: UserWarning: torch.cuda.amp.GradScaler is enabled, but CUDA is not available.  Disabling.
  warnings.warn(
  0%|          | 0/1250 [00:00<?, ?it/s]<ipython-input-7-dc2036bb6ca3>:37: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please use `torch.amp.autocast('cuda', args...)` instead
  with torch.cuda.amp.autocast():
/usr/local/lib/python3.10/dist-packages/torch/amp/autocast_mode.py:265: UserWarning: User provided device_type of 'cuda', but CUDA is not available. Disabling
  warnings.warn(
100%|██████████| 1250/1250 [1:51:57<00:00,  5.37s/it]Epoch 1/1 - Loss: 0.6961234574317932
```

8. Avaliação do Modelo

O modelo é avaliado no conjunto de teste e diversas métricas de desempenho são calculadas. A baixa acurácia sugere a necessidade de ajustes, como aumentar o número de épocas ou ajustar a taxa de aprendizado. A acurácia de 49.56% mostra que o modelo está quase no nível de um chute aleatório. A precisão de 0.50 e o AUC-ROC de 0.50 indicam que o modelo não está diferenciando bem entre bots e não bots. Para melhorar, é recomendado aumentar o número de épocas, ajustar a taxa de aprendizado ou descongelar mais camadas do BERT para permitir maior aprendizado.

```
model.eval()  # Coloca o modelo no modo de avaliação
correct = 0
total = 0
all_labels = []
all_predictions = []

for batch in test_dataloader:
    b_input_ids, b_input_mask, b_labels = tuple(t.to(device) for t in batch)

    with torch.no_grad():
        outputs = model(b_input_ids, attention_mask=b_input_mask)

    logits = outputs.logits
    predictions = torch.argmax(logits, dim=-1)

    correct += (predictions == b_labels).sum().item()
    total += b_labels.size(0)

    # Armazena todos os rótulos e previsões para métricas posteriores
    all_labels.extend(b_labels.cpu().numpy())
    all_predictions.extend(predictions.cpu().numpy())

accuracy = correct / total
print(f"Acurácia no conjunto de teste: {accuracy * 100:.2f}%")

# Calcular outras métricas
precision = precision_score(all_labels, all_predictions, average='binary')
recall = recall_score(all_labels, all_predictions, average='binary')
f1 = f1_score(all_labels, all_predictions, average='binary')
conf_matrix = confusion_matrix(all_labels, all_predictions)
```

```
print(f"Precisão: {precision:.2f}")
print(f"Revocação: {recall:.2f}")
print(f"F1-Score: {f1:.2f}")
print("Matriz de Confusão:")
print(conf_matrix)
print(f"AUC-ROC: {auc_roc:.2f}")
```

```
Acurácia no conjunto de teste: 49.56%
Precisão: 0.50
Revocação: 0.59
F1-Score: 0.54
Matriz de Confusão:
[[1994 2974]
 [2070 2962]]
AUC-ROC: 0.50
```