Sentiment Analysis Pipeline with Hugging Face

Koushik Reddy

1. Loading the Data and Model

The IMDb dataset is loaded using the `datasets` library, providing a standard train/test split for binary sentiment classification. The pre-trained BERT model `bert-base-uncased` is used for its strong performance on natural language tasks. The corresponding tokenizer is loaded based on the model name to ensure compatibility.

2. Tokenizing the Data

Tokenization is done using the BERT tokenizer with truncation and padding. To determine an appropriate truncation length, I plotted a histogram of input token lengths and chose a `max_length` of 256. This value captures most of the reviews' context while balancing computational efficiency.

3. Fine-Tuning the Model

I fine-tuned the model using the Hugging Face `Trainer` API for ease of use and built-in integration with PyTorch. The training parameters are:

- `output_dir="checkpoints"`: Directory for saving checkpoints.
- `per_device_train_batch_size=4` and `per_device_eval_batch_size=4`: Small batch sizes to fit GPU memory.
- `gradient_accumulation_steps=16`: Accumulates gradients to simulate a larger batch.
- `num_train_epochs=3`: Sufficient epochs for fine-tuning, as the mode is large
- `eval_strategy="epoch"` and `save_strategy="epoch"`: Evaluate and save after each epoch.
- `learning_rate=2e-5`: Typical for BERT fine-tuning.
- `load_best_model_at_end=True`: Keep the checkpoint with the best evaluation metric.

4. Evaluating the Model

After training, the model is evaluated using `trainer.evaluate()`. The `compute_metrics` function returns both accuracy and F1 score. The final performance on the validation set is:

- **Accuracy**: 91.544%
- **F1 Score**: 91.543%

These scores indicate that the fine-tuned BERT model performs well on the sentiment classification task.

5. Saving and Loading the Model

Once the best checkpoint is selected, the model and tokenizer are saved:

```
trainer.model.save_pretrained("./best_model")
tokenizer.save_pretrained("./best_model")
```

They can be reloaded later for inference without retraining.

6. Making Predictions

For inference on new text, the saved model and tokenizer can be loaded and used with a custom prediction function:

```
def predict_sentiment(model, tokenizer, text):
    model_inputs = tokenizer(text, return_tensors='pt')
    pred = torch.argmax(model(**model_inputs).logits)
    return ['NEGATIVE', 'POSITIVE'][pred]
```

This function tokenizes the input text, feeds it through the model, and returns either `NEGATIVE` or `POSITIVE` sentiment based on the prediction.