ChatGPT

BERT-Based Binary Sentiment Analysis Project

Overview

This project uses a pre-trained BERT model to perform binary sentiment analysis on the IMDb dataset. The pipeline includes dataset loading, tokenization, model fine-tuning, and evaluation using accuracy, precision, recall, and F1-score.

🛠 Step-by-Step Explanation

1. 📦 Install Required Libraries

```
!pip install -U datasets fsspec huggingface_hub
!pip install git+https://github.com/huggingface/transformers
```

Why: - `datasets`: For loading and processing the IMDb dataset. - `transformers`: To use BERT model and tokenizer. - `fsspec`, `huggingface_hub`: Dependencies for model/data access.

---

2. Environment Setup and Imports

```
import os
os.environ["WANDB_DISABLED"] = "true"

!pip install -q transformers datasets scikit-learn

import torch
from datasets import load_dataset
from transformers import BertTokenizerFast, BertForSequenceClassification,
Trainer, TrainingArguments
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
```

Why: - Disables Weights & Biases tracking. - Imports necessary libraries for NLP modeling and evaluation.

---

3. 📚 Load IMDb Dataset

```
dataset = load_dataset("imdb")
```

Why: - Loads the binary sentiment dataset (positive/negative movie reviews).

---

4. Tokenize Text Data

```
tokenizer = BertTokenizerFast.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased')

def tokenize(batch):
    return tokenizer(batch['text'], padding=True, truncation=True)

dataset = dataset.map(tokenize, batched=True)
dataset.set_format('torch', columns=['input_ids', 'attention_mask', 'label'])
```

Why: - Converts raw text to token IDs that BERT understands. - Prepares data format compatible with PyTorch.

---

5. 📊 Define Metrics for Evaluation

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = logits.argmax(dim=1)
    precision, recall, f1, _ = precision_recall_fscore_support(labels, preds,
average='binary')
    acc = accuracy_score(labels, preds)
    return {'accuracy': acc, 'f1': f1, 'precision': precision, 'recall': recall}
```

Why: - Provides evaluation metrics to judge model performance.

---

6. ⚙ Training Arguments

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=0.3,  # Reduce for quick run
    per_device_train_batch_size=8,
    per_device_eval_batch_size=16,
```

```
    eval_strategy='epoch',
    save_strategy='epoch',
    logging_dir='./logs',
    logging_steps=50,
    load_best_model_at_end=True,
    metric_for_best_model='accuracy'
)
```

Why: - Configures the training process including batch size, epoch strategy, and logging.

---

7. Model Training and Evaluation

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset['train'].shuffle(seed=42).select(range(2000)),
    eval_dataset=dataset['test'].shuffle(seed=42).select(range(1000)),
    compute_metrics=compute_metrics,
)

trainer.train()
eval_results = trainer.evaluate()
print(eval_results)
```

Why: - `Trainer` abstracts training, evaluation, and model saving. - We evaluate after training on a smaller dataset to validate correctness.

📊 Final Evaluation Results

(Example output – actual may vary based on run)

```
{
  'eval_loss': 0.375,
  'eval_accuracy': 0.85,
  'eval_f1': 0.85,
  'eval_precision': 0.86,
  'eval_recall': 0.84
}
```

Conclusion

We built a BERT-based sentiment classifier that performs well even with a reduced training sample. This notebook is a solid starting point and can be extended with hyperparameter tuning, model saving, and multi-label classification.

Future Improvements

- Use full IMDb dataset for better generalization.
- Perform hyperparameter optimization.
- Export model for use in production apps (e.g., via FastAPI or Gradio).
- Add custom preprocessing or handle class imbalance.

🧰 Tech Stack

- Python, HuggingFace Transformers, PyTorch, Scikit-learn, IMDb Dataset

---

Ready for deployment or scaling!