# 大数据分析实践实验报告

## 实验五 MRPC—BERT 微调

### 一、实验目标

本次实验以 MRPC 数据集为研究对象，旨在基于 BERT-base 预训练模型构建句子对释义分类模型，通过微调预训练模型实现对两个句子是否为释义关系的精准判断，验证预训练模型在语义匹配任务中的适配性，同时评估模型在准确率、F1 分数等核心指标上的表现，探索适用于句子对分类任务的微调策略

### 二、实验环境

python3，jupyter notebook

### 三、实验过程

MRPC（Microsoft Research Paraphrase Corpus）任务的目标是判断两个句子是否互为释义（即表达相同的语义），属于句子对分类任务（二分类：是 / 否）
本实验的核心思路是基于 Hugging Face 生态的预训练模型微调范式，围绕“数据加载 - 预处理 - 模型构建 - 训练评估 - 推理部署”的完整 NLP 任务流程展开

### 实验思路：

```
import torch
from datasets import load_dataset
from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    TrainingArguments,
    Trainer,
    DataCollatorWithPadding
)
import evaluate
import numpy as np

# 设置随机种子以确保结果可复现
torch.manual_seed(42)
np.random.seed(42)

# 1. 加载MRPC数据集
print("加载MRPC数据集...")
dataset = load_dataset( path: "glue",  name: "mrpc")

# 2. 加载预训练模型和分词器
model_name = "bert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name, num_labels=2)
```

首先，代码先完成环境准备，导入 transformers、datasets 等核心库并设置随机种子确保结果可复现，以保证实验稳定性。接着通过 load_dataset 加载 GLUE 基准中的 MRPC 数据集，其中包含标注好的句子对及是否为释义的标签，直接对接任务需求。随后选用 bert-base-uncased 预训练模型，搭配对应的 AutoTokenizer 分词器，前者通过 AutoModelForSequenceClassification 加载并适配二分类任务（设置 num_labels=2），后者负责将原始文本转换为模型可识别的 token 序列

```
# 3. 数据预处理函数
def preprocess_function(examples):
    return tokenizer(examples["sentence1"], examples["sentence2"], truncation=True)

print("预处理数据...")
tokenized_dataset = dataset.map(preprocess_function, batched=True)

# 4. 数据收集器（用于动态填充）
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# 5. 加载评估指标（MRPC使用准确率和F1分数）
accuracy = evaluate.load("accuracy")
f1 = evaluate.load("f1")

def compute_metrics(eval_pred):
    predictions, labels = eval_pred
    predictions = np.argmax(predictions, axis=1)
    acc = accuracy.compute(predictions=predictions, references=labels)
    f1_score = f1.compute(predictions=predictions, references=labels)
    return {**acc, **f1_score}
```

数据预处理环节，自定义 preprocess_function 对句子对进行分词、截断处理，利用 dataset.map 实现批量预处理，再通过 DataCollatorWithPadding 完成动态填充，保证同一批次数据长度一致，既满足模型输入要求又提升训练效率。评估指标方面，结合 MRPC 任务特点，同时加载准确率和 F1 分数，通过 compute_metrics 函数将模型输出的 logits 转换为预测类别后计算指标，兼顾分类正确性和类别平衡下的性能表现

```python
# 6. 设置训练参数 - 修改了evaluation_strategy为eval_strategy
training_args = TrainingArguments(
    output_dir="./mrpc-bert-model",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=3,
    weight_decay=0.01,
    eval_strategy="epoch",  # 注意这里改为eval_strategy
    save_strategy="epoch",
    load_best_model_at_end=True,
    push_to_hub=False,
)

# 7. 初始化Trainer
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)

# 8. 训练模型
print("开始训练模型...")
trainer.train()

# 9. 评估模型
print("评估模型...")
eval_results = trainer.evaluate()
print(f"评估结果: {eval_results}")

# 10. 在测试集上评估
print("在测试集上评估...")
test_results = trainer.evaluate(tokenized_dataset["test"])
print(f"测试集结果: {test_results}")

# 11. 保存模型
print("保存模型...")
trainer.save_model("./mrpc-bert-final")
```

训练配置通过 TrainingArguments 设定关键参数：输出目录、学习率（2e-5 的小学习率避免破坏预训练语义）、批次大小、训练轮数，以及 eval_strategy="epoch"（每轮结束后验证）、load_best_model_at_end=True（保留最优模型）等策略，平衡训练效果与泛化能力。初始化 Trainer 时整合模型、数据、指标等组件，调用 train () 完成微调，evaluate () 分别在验证集和测试集评估性能，最后通过 save_model 保存模型

```python
# 12. 模型推理示例
print("\n模型推理示例:")
def predict_paraphrase(sentence1, sentence2):
    inputs = tokenizer(sentence1, sentence2, return_tensors="pt", truncation=True)
    with torch.no_grad():
        logits = model(**inputs).logits
    predicted_class = torch.argmax(logits, dim=1).item()
    return "是释义" if predicted_class == 1 else "不是释义"

# 测试几个例子
examples = [
    ("The cat sat on the mat.", "A cat was sitting on the mat."),
    ("I love programming.", "Programming is my passion."),
    ("The weather is nice today.", "It's raining heavily outside.")
]

for s1, s2 in examples:
    result = predict_paraphrase(s1, s2)
    print(f"句子1: {s1}")
    print(f"句子2: {s2}")
    print(f"预测结果: {result}\n")
```

推理部分设计 predict_paraphrase 函数，接收句子对后经分词编码、模型前向传播（关闭梯度计算提升效率），取 logits 最大值对应的类别作为结果，直观展示模型的实际应用效果

## 实验结果：

```
加载MRPC数据集...
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
预处理数据...
Map: 100%|██████████| 1725/1725 [00:00<00:00, 12178.54 examples/s]
```

成功加载数据集，并成功完成数据预处理

```
开始训练模型...
  0%|          | 0/690 [00:00<?, ?it/s]D:\pythonProject\venv\Lib\site-packages\torch\utils\data\dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinn
  won't be used.
  warnings.warn(warn_msg)
 33%|███       | 230/690 [10:16<15:12,  1.98s/it]
  0%|          | 0/26 [00:00<?, ?it/s]
  8%|▊         | 2/26 [00:00<00:07,  3.25it/s]
 12%|█▏        | 3/26 [00:01<00:09,  2.35it/s]
 15%|█▌        | 4/26 [00:01<00:10,  2.01it/s]
 19%|█▉        | 5/26 [00:02<00:11,  1.81it/s]
 23%|██▎       | 6/26 [00:03<00:10,  1.84it/s]
 27%|██▋       | 7/26 [00:03<00:10,  1.74it/s]
 31%|███       | 8/26 [00:04<00:11,  1.63it/s]
 35%|███▍      | 9/26 [00:05<00:10,  1.58it/s]
 38%|███▊      | 10/26 [00:05<00:10,  1.54it/s]
 42%|████▏     | 11/26 [00:06<00:09,  1.55it/s]
 46%|████▌     | 12/26 [00:06<00:08,  1.58it/s]
 50%|█████     | 13/26 [00:07<00:07,  1.66it/s]
 54%|█████▍    | 14/26 [00:08<00:07,  1.66it/s]
 58%|█████▊    | 15/26 [00:08<00:06,  1.58it/s]
 62%|██████▏   | 16/26 [00:09<00:06,  1.64it/s]
 65%|██████▌   | 17/26 [00:09<00:05,  1.65it/s]
 69%|██████▉   | 18/26 [00:10<00:04,  1.67it/s]
 73%|███████▎  | 19/26 [00:11<00:04,  1.69it/s]
 77%|███████▋  | 20/26 [00:11<00:03,  1.70it/s]
 81%|████████  | 21/26 [00:12<00:02,  1.67it/s]
 85%|████████▍ | 22/26 [00:12<00:02,  1.70it/s]
 88%|████████▊ | 23/26 [00:13<00:01,  1.65it/s]
 92%|█████████▏| 24/26 [00:14<00:01,  1.64it/s]
 96%|█████████▌| 25/26 [00:14<00:00,  1.66it/s]

{'eval_loss': 0.4063064754009247, 'eval_accuracy': 0.8186274509803921, 'eval_f1': 0.875838926174497, 'eval_runtime': 15.7235, 'eval_samples_per_second': 25.948, 'eval_steps_per_second': 1.654, 'epoch': 1.0}
```

第一轮训练

```
 67%|██████    | 460/690 [20:38<07:36,  1.99s/it]
  0%|          | 0/26 [00:00<?, ?it/s]
  8%|▊         | 2/26 [00:00<00:07,  3.38it/s]
 12%|█▏        | 3/26 [00:01<00:09,  2.38it/s]
 15%|█▌        | 4/26 [00:01<00:11,  2.00it/s]
 19%|█▉        | 5/26 [00:02<00:11,  1.81it/s]
 23%|██▎       | 6/26 [00:02<00:10,  1.84it/s]
 27%|██▋       | 7/26 [00:03<00:10,  1.73it/s]
 31%|███       | 8/26 [00:04<00:11,  1.63it/s]
 35%|███▍      | 9/26 [00:04<00:10,  1.60it/s]
 38%|███▊      | 10/26 [00:05<00:10,  1.60it/s]
 42%|████▏     | 11/26 [00:06<00:09,  1.62it/s]
 46%|████▌     | 12/26 [00:06<00:08,  1.63it/s]
 50%|█████     | 13/26 [00:07<00:07,  1.70it/s]
 54%|█████▍    | 14/26 [00:07<00:07,  1.69it/s]
 58%|█████▊    | 15/26 [00:08<00:06,  1.60it/s]
 62%|██████▏   | 16/26 [00:09<00:05,  1.68it/s]
 65%|██████▌   | 17/26 [00:09<00:05,  1.70it/s]
 69%|██████▉   | 18/26 [00:10<00:04,  1.73it/s]
 73%|███████▎  | 19/26 [00:10<00:04,  1.72it/s]
 77%|███████▋  | 20/26 [00:11<00:03,  1.72it/s]
 81%|████████  | 21/26 [00:12<00:02,  1.69it/s]
 85%|████████▍ | 22/26 [00:12<00:02,  1.71it/s]
 88%|████████▊ | 23/26 [00:13<00:01,  1.68it/s]
 92%|█████████▏| 24/26 [00:13<00:01,  1.65it/s]
 96%|█████████▌| 25/26 [00:14<00:00,  1.69it/s]

{'eval_loss': 0.3754688799381256, 'eval_accuracy': 0.8455882352941176, 'eval_f1': 0.8926746166950597, 'eval_runtime': 15.4257, 'eval_samples_per_second': 26.449, 'eval_steps_per_second': 1.685, 'epoch': 2.0}
```

第二轮训练

```
 72%|███████   | 500/690 [22:41<08:30,  2.69s/it]{'loss': 0.4069, 'grad_norm': 3.769829750061035, 'learning_rate': 5.536231884057971e-06, 'epoch': 2.17}
100%|██████████| 690/690 [31:02<00:00,  2.11s/it]
  0%|          | 0/26 [00:00<?, ?it/s]
  8%|▊         | 2/26 [00:00<00:07,  3.10it/s]
 12%|█▏        | 3/26 [00:01<00:10,  2.16it/s]
 15%|█▌        | 4/26 [00:01<00:11,  1.89it/s]
 19%|█▉        | 5/26 [00:02<00:12,  1.69it/s]
 23%|██▎       | 6/26 [00:03<00:11,  1.67it/s]
 27%|██▋       | 7/26 [00:04<00:13,  1.38it/s]
 31%|███       | 8/26 [00:04<00:12,  1.39it/s]
 35%|███▍      | 9/26 [00:05<00:12,  1.41it/s]
 38%|███▊      | 10/26 [00:06<00:11,  1.44it/s]
 42%|████▏     | 11/26 [00:06<00:09,  1.58it/s]
 46%|████▌     | 12/26 [00:07<00:09,  1.55it/s]
 50%|█████     | 13/26 [00:08<00:07,  1.63it/s]
 54%|█████▍    | 14/26 [00:08<00:07,  1.64it/s]
 58%|█████▊    | 15/26 [00:09<00:07,  1.55it/s]
 62%|██████▏   | 16/26 [00:09<00:06,  1.64it/s]
 65%|██████▌   | 17/26 [00:10<00:05,  1.66it/s]
 69%|██████▉   | 18/26 [00:11<00:04,  1.67it/s]
 73%|███████▎  | 19/26 [00:11<00:04,  1.68it/s]
 77%|███████▋  | 20/26 [00:12<00:03,  1.68it/s]
 81%|████████  | 21/26 [00:12<00:03,  1.66it/s]
 85%|████████▍ | 22/26 [00:13<00:02,  1.68it/s]
 88%|████████▊ | 23/26 [00:14<00:01,  1.63it/s]
 92%|█████████▏| 24/26 [00:14<00:01,  1.60it/s]
 96%|█████████▌| 25/26 [00:15<00:00,  1.63it/s]

{'eval_loss': 0.4392760694026947, 'eval_accuracy': 0.8504901960784313, 'eval_f1': 0.8950086058519794, 'eval_runtime': 16.3567, 'eval_samples_per_second': 24.944, 'eval_steps_per_second': 1.59, 'epoch': 3.0}
```

第三轮训练

```
评估模型...
D:\pythonProject\venv\Lib\site-packages\torch\utils\data\dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
  warnings.warn(warn_msg)
100%|██████████| 26/26 [00:15<00:00,  1.72it/s]
评估结果: {'eval_loss': 0.3754688799381256, 'eval_accuracy': 0.8455882352941176, 'eval_f1': 0.8926746166950597, 'eval_runtime': 15.8046, 'eval_samples_per_second': 25.815, 'eval_steps_per_second': 1.645, 'epoch': 3.0}
在测试集上评估...
D:\pythonProject\venv\Lib\site-packages\torch\utils\data\dataloader.py:665: UserWarning: 'pin_memory' argument is set as true but no accelerator is found, then device pinned memory won't be used.
  warnings.warn(warn_msg)
100%|██████████| 108/108 [01:06<00:00,  1.62it/s]
测试集结果: {'eval_loss': 0.38656508922576904, 'eval_accuracy': 0.8376811594202899, 'eval_f1': 0.884809942004971, 'eval_runtime': 67.4458, 'eval_samples_per_second': 25.576, 'eval_steps_per_second': 1.601, 'epoch': 3.0}
保存模型...
```

分别在验证集与测试集上评估模型性能

从测试集评估结果来看，模型整体表现符合预期且达到 BERT-base 在该任务上的典型基线水平：eval_loss 为 0.3866，数值较低说明模型预测与真实标签贴合度高，无明显过拟合或欠拟合；83.77% 的准确率属于中等偏上，虽略低于 BERT-base 的最优基线（84%-86%）但处于合理区间，反映模型对句子对释义关系的整体判断能力良好；88.40% 的 F1 分数表现优秀，说明模型在精确率和召回率间实现了较好平衡，对正类（释义对）的识别效果突出。效率层面，67.4458 秒完成约 1725 个样本的评估、每秒处理 25.576 个样本，推理速度能够满足常规场景需求

模型推理示例：

```
句子1: The cat sat on the mat.
句子2: A cat was sitting on the mat.
预测结果: 是释义

句子1: I love programming.
句子2: Programming is my passion.
预测结果: 是释义

句子1: The weather is nice today.
句子2: It's raining heavily outside.
预测结果: 不是释义
```

观察可知，模型的预测结果正确，能正确判断两个句子是否互为释义（即表达相同的语义）

## 四、实验总结

本次实验，我们成功基于 BERT-base 模型完成了 MRPC 句子对释义分类任务的模型训练与评估，模型在测试集上取得了 83.77% 的准确率和 88.40% 的 F1 分数，接近 BERT-base 在该任务上的基线水平，损失值低且无明显过拟合 / 欠拟合现象，推理效率也满足常规应用需求，从而验证了预训练模型微调方案在语义匹配任务中的有效性。整体而言，模型达到了实验预期目标，可用于实际的释义判断场景；若需进一步提升性能，或许可尝试更换更大规模的预训练模型、优化超参数或引入数据增强策略