

BERT 移动端文本识别系统 (BERT Text Classification & NER for Mobile)

本项目提供了一套完整的解决方案，用于训练 BERT 模型识别文本中的“日程/闹钟”意图，并提取“时间、地点、内容”等实体信息。最终模型通过 ONNX 导出，可直接部署于 Android/iOS 移动端实现离线推理。

项目结构

```
BERT_Project/
├── data/                                # 训练数据存放目录
│   ├── classifier_train.jsonl          # 意图分类训练数据
│   └── ner_train.jsonl                 # 实体抽取训练数据 (BIOES/JSON格式)
├── models/                             # 模型输出目录 (训练后的模型和导出文件)
│   ├── classifier/                     # 训练好的分类器 PyTorch 模型
│   ├── ner/                           # 训练好的 NER PyTorch 模型
│   ├── classifier.onnx                 # [产物] 移动端可用分类模型 (FP32, ~390MB)
│   ├── ner.onnx                       # [产物] 移动端可用 NER 模型 (FP32, ~390MB)
│   ├── classifier.quant.onnx           # [推荐] 量化后的分类模型 (Int8, ~100MB)
│   └── ner.quant.onnx                 # [推荐] 量化后的 NER 模型 (Int8, ~100MB)
├── train_classifier.py                  # 分类模型训练脚本
├── train_ner.py                        # NER 模型训练脚本
├── export_onnx.py                      # 分类模型 ONNX 导出脚本
├── export_ner_onnx.py                  # NER 模型 ONNX 导出脚本
├── clean_and_quantize.py               # [新增] 模型量化脚本 (压缩模型体积)
├── infer_pipeline.py                   # 完整链路测试脚本 (串联两级模型)
├── verify_onnx.py                     # 单模型验证脚本
├── Dockerfile                         # 开发环境镜像定义
├── docker-compose.yml                  # 容器编排配置
└── requirements.txt                    # Python 依赖
```

快速开始 (推荐使用 Docker)

本项目已封装 Docker 环境，无需在本地安装复杂的 PyTorch/CUDA 环境。

1. 启动开发环境

在项目根目录下运行：

```
docker compose up -d --build
```

2. 进入开发容器

```
docker compose exec bert-trainer bash
```

注意：后续所有命令均默认在容器内的 `/app` 目录下执行。

🔧 详细训练与部署流程

第一步：准备数据 (数据增强)

为提升模型精度，我们提供了数据生成脚本，可以自动生成大量带标注的训练数据。

1. 生成训练数据 (10,000条)：

```
python generate_data.py
```

该脚本会在 `data/` 目录下生成 `classifier_train_large.jsonl` 和 `ner_train_large.jsonl`。

2. 数据格式说明：

- 分类数据 (`data/classifier_train_large.jsonl`):

```
{"text": "明天下午3点在会议室开会", "label": 0}
```

(Label定义: 0=日程, 1=闹钟, 2=其他)

- NER数据 (`data/ner_train_large.jsonl`):

```
{"tokens": ["明", "天", "开", "会"], "ner_tags": [1, 2, 0, 0]}
```

(Tags: O=0, B-TIME=1, I-TIME=2, B-LOC=3, I-LOC=4, B-CONTENT=5, I-CONTENT=6)

第二步：训练意图分类模型 (Classifier)

该模型用于判断用户输入的句子是“日程”、“闹钟”还是“其他”。

```
python train_classifier.py \  
  --train_file data/classifier_train_large.jsonl \  
  --output_dir models/classifier \  
  --epochs 3
```

输出：模型文件保存在 `models/classifier/`

第三步：训练实体抽取模型 (NER)

该模型用于从句子中提取具体的时间、地点和事件内容。

```
python train_ner.py \  
  --train_file data/ner_train_large.jsonl \  
  --output_dir models/ner \  
  --epochs 5
```

输出：模型文件保存在 *models/ner/*

第四步：导出为移动端模型 (ONNX)

PyTorch 模型不能直接在手机上运行，我们需要将其导出为通用的 ONNX 格式。

1. 导出分类模型

```
python export_onnx.py \  
  --model_dir models/classifier \  
  --output models/classifier.onnx
```

2. 导出 NER 模型

```
python export_ner_onnx.py \  
  --model_dir models/ner \  
  --output models/ner.onnx
```

产物： *models/classifier.onnx* 和 *models/ner.onnx* (FP32, 约 390MB)

第五步：模型量化 (Optimization) [推荐]

原始导出的 ONNX 模型体积较大（约 390MB），建议进行 Int8 量化，将其压缩至 100MB 左右，更适合移动端下载和加载。

1. 量化分类模型

```
python clean_and_quantize.py models/classifier.onnx  
models/classifier.quant.onnx
```

2. 量化 NER 模型

```
python clean_and_quantize.py models/ner.onnx models/ner.quant.onnx
```

产物： *models/classifier.quant.onnx* 和 *models/ner.quant.onnx* (Int8, 约 98MB)

第六步：全链路验证 (Simulation)

在交付给移动端开发之前，我们可以通过 `infer_pipeline.py` 脚本来验证两个 ONNX 模型配合工作的效果。它模拟了“先分类，再抽取”的业务逻辑。

```
python infer_pipeline.py --text "明天下午3点在会议室开会"
```

预期输出 JSON:

```
{
  "type": "日程",
  "raw_text": "明天下午3点在会议室开会",
  "time": "明天 下午 3 点",
  "location": "会议室",
  "content": "开会"
}
```

移动端集成指南

对于 Android (Java/Kotlin) 或 iOS (Swift/ObjC) 开发人员：

1. 获取模型文件：

将 `models/classifier.quant.onnx` 和 `models/ner.quant.onnx` 放入 App 的 Assets 目录。

2. 集成 ONNX Runtime 库：

- Android: `implementation 'com.microsoft.onnxruntime:onnxruntime-android:1.15.0'`
- iOS: `pod 'OnnxRuntime-c'`

3. 推理逻辑伪代码：

```
// 1. 预处理 (Tokenizer)
// 需要在端上实现一个简单的 WordPiece Tokenizer (或使用现成库)
long[] inputIds = tokenizer.encode(userInput);

// 2. 运行分类模型
OrtSession clsSession = env.createSession("classifier.quant.onnx");
float[] categoryLogits = clsSession.run(inputIds);
int category = argmax(categoryLogits);

if (category == SCHEDULE || category == ALARM) {
    // 3. 运行 NER 模型
    OrtSession nerSession = env.createSession("ner.quant.onnx");
    float[][] tokenScores = nerSession.run(inputIds);
}
```

```
int[] tags = argmax(tokenScores); // B-TIME, I-LOC...

// 4. 解析结果
Result result = parseBioTags(userInput, tags);
return result.toJson();
}
```

本地开发 (不使用 Docker)

如果您坚持使用本地 Python 环境：

1. 创建虚拟环境 (Python 3.10+):

```
python -m venv venv
source venv/bin/activate
```

2. 安装依赖:

```
pip install -r requirements.txt
```

3. 运行上述所有 python 命令即可。