尚硅谷大模型技术之AI智评

（作者：尚硅谷研究院）

版本：V0.9.0

1. 项目架构
   1. 项目简介

本项目基于BERT预训练模型根据文本分类任务微调，用于评论情感分析，将评论分类为“好评”与“差评”两类。通过Transformers与PyTorch进行模型的搭建与训练，并使用TensorBoard可视化模型参数、计算图、损失与准确率等信息。通过ONNX框架部署模型，并通过Starlette框架提供Web界面。

* 1. 核心功能

评论分类

* 1. 应用场景

投诉预警：发现差评关键词（如“扣费异常”）触发内部审核。

商品评价管理：自动筛选差评，优先处理质量问题或投诉。

虚假评论检测：识别异常好评（如刷单）或恶意差评。

评论过滤：过滤无意义好评，保留实质性反馈。

社区管理：自动隐藏辱骂性差评或广告性质好评。

内容推荐：根据用户对电影/音乐的好评偏好调整推荐算法。

* 1. 项目结构

本项目主要由6个模块组成：

* 数据预处理模块（preprocess）
* 模型搭建模块（models）
* 模型训练与测试模块（train）
* 可视化模块（visual）
* 模型推理模块（inference）
* 模型部署模块（deploy）
  1. 技术栈

深度学习框架：PyTorch

Web框架：Starlette

数据处理：pandas，datasets，Numpy

可视化：TensorBoard

预训练模型使用：Transformers

模型部署：ONNX

* 1. 项目准备
     1. 环境准备
        + 1. 创建conda虚拟环境

终端输入如下命令，创建项目的虚拟环境，并指定Python版本：

conda create -n sentiment-analysis python=3.12

激活该虚拟环境：

conda activate sentiment-analysis

更换国内源：

pip config set global.index-url <https://pypi.tuna.tsinghua.edu.cn/simple>

根据CUDA版本选择PyTorch版本并安装：

pip3 install torch torchvision torchaudio --index-url <https://download.pytorch.org/whl/cu126>

安装其他所需的库：

pip install jupyter pandas matplotlib tqdm transformers datasets tensorboard starlette uvicorn onnx onnxruntime

* + 1. 模型准备

BERT预训练模型：<https://huggingface.co/google-bert/bert-base-chinese>。

RoBERTa预训练模型：<https://huggingface.co/uer/roberta-base-finetuned-dianping-chinese>。

* + 1. 项目目录结构

sentiment\_analysis                                /项目根目录

    ├─data                                        /原始数据

    │      online\_shopping\_10\_cats.csv            /购物评论数据集

    │

    ├─common                                      /公共模块

    ├─preprocess                                  /数据预处理模块

    ├─models                                      /模型模块

    │  ├─bert-base-chinese                        /BERT预训练模型

    │  └─roberta-base-finetuned-dianping-chinese  /RoBERTa预训练模型

    ├─train                                       /训练模块

    ├─visual                                      /可视化模块

    ├─inference                                   /推理模块

    ├─deploy                                      /部署模块

    └─templates                                   /网页模板

1. 公共模块
   1. 模块结构

common

    ├─\_\_init\_\_.py

    ├─config.py     /配置文件

    └─seed.py       /固定随机数种子

* 1. config.py

config.py文件中存放项目所有的配置信息：

"""配置文件"""

import torch

# 随机数种子

SEED = 1234

# ======================== 数据集配置 ========================

# 数据集路径

DATA\_PATH = "data/online\_shopping\_10\_cats.csv"

# 测试集比例

TEST\_RATIO = 0.05

# 训练集比例

TRAIN\_RATIO = 0.75

# 数据集大小上限

TOTAL\_EXAMPLES = 100000

# ======================== 模型配置 ========================

# BERT预训练模型路径

TRANSFORMER\_PATH = "models/bert-base-chinese"

# RoBERTa预训练模型路径

ROBERTA\_PATH = "models/roberta-base-finetuned-dianping-chinese"

# 模型输出维度

OUTPUT\_DIM = 2

# 是否冻结预训练模型权重

FREEZE = False

# ======================== 训练配置 ========================

# 批次大小

BATCH\_SIZE = 2

# 学习率

LEARNING\_RATE = 1e-5

# 训练轮数

EPOCHS = 2

# 设备

DEVICE = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

# 模型保存路径

MODEL\_PATH = "models/bert-base-chinese-finetuned-sentiment-analysis.pt"

# TensorBoard日志保存路径

LOG\_PATH = "logs/log\_0"

# ======================== 部署配置 ========================

# onnx模型保存路径

DEPLOY\_MODEL\_PATH = "deploy/bert-base-chinese-finetuned-sentiment-analysis.onnx"

* 1. seed.py

seed.py中提供一个函数用于固定所有的随机数种子，确保结果可复现：

"""设置随机数种子"""

import torch

import random

import numpy as np

from common import config

def set\_seed():

    """设置随机数种子"""

    seed = config.SEED

    # 设置torch随机种子

    torch.manual\_seed(seed)

    torch.cuda.manual\_seed(seed)

    torch.cuda.manual\_seed\_all(seed)

    # 设置numpy随机种子，pandas也会使用这个种子

    np.random.seed(seed)

    # 设置python随机种子

    random.seed(seed)

    # 禁用cudnn的自动优化，确保算法一致

    torch.backends.cudnn.benchmark = False

    # 启用cudnn的确定性模式，确保结果可重复

    torch.backends.cudnn.deterministic = True

* 1. \_\_init\_\_.py

在\_\_init\_\_.py中从当前包下的模块中导入函数，外部可以更方便的调用函数。

"""

公共包

配置文件:

- config

函数:

- set\_seed: 设置随机数种子

"""

from .seed import set\_seed

1. 数据预处理模块
   1. 模块结构

preprocess

    ├─\_\_init\_\_.py

    ├─get\_dataset.py            /加载数据集

    ├─get\_tokenizer.py          /加载tokenizer

    ├─get\_tokenized\_dataset.py  /获取tokenizer处理后的数据集

    ├─get\_dataloader.py         /获取DataLoader

└─preprocess.py             /执行数据预处理所有流程

* 1. get\_dataset.py

原始数据集如下：

文本

描述已自动生成

首先将review列重命名为text，之后只保留text列和lable列，并去除缺失值。

其次根据配置文件中设置的样本数上限获取指定数量的数据作为数据集，并划分训练集和测试集。

"""加载数据集"""

import datasets

import pandas as pd

def get\_dataset(data\_path, test\_ratio, total\_examples):

    """

    加载数据集

    参数:

    - data\_path: 数据集路径

    - test\_ratio: 测试集比例

    - total\_examples: 数据集大小上限

    返回值:

    - train\_dataset: 训练集

    - test\_dataset: 测试集

    """

    # 加载数据集

    df = pd.read\_csv(data\_path)

    # 重命名review列名为text

    df = df.rename(columns={"review": "text"})

    # 保留text和label列

    df = df[["text"] + ["label"]]

    # 去除缺失值

    df = df.dropna()

    # 取数据集总数据量和total\_examples中的较小值作为数据集大小

    num\_example = min(len(df), total\_examples)

    # 随机采样数据集大小的数据

    df = df.sample(n=num\_example)

    # 训练集大小

    num\_train = int(num\_example \* (1 - test\_ratio))

    # 加载训练集

    train\_dataset = datasets.Dataset.from\_pandas(df=df[:num\_train])

    # 加载测试集

    test\_dataset = datasets.Dataset.from\_pandas(df=df[num\_train:])

    return train\_dataset, test\_dataset

* 1. get\_tokenizer.py

根据配置文件中设置的预训练模型加载相应的tokenizer。

"""加载tokenizer"""

import transformers

def get\_tokenizer(transformer\_path):

    """

    通过指定路径加载tokenzier

    参数:

    - transformer\_path: 预训练模型路径

    返回值:

    - tokenizer: tokenzier对象

    """

    return transformers.AutoTokenizer.from\_pretrained(transformer\_path)

* 1. get\_tokenized\_dataset.py
  2. get\_dataloader.py
  3. preprocess.py
  4. \_\_init\_\_.py

1. 模型搭建模块
2. 训练模块
3. 推理模块
4. 可视化模块
5. Web模块
6. 模型部署