

新闻话题分类

一、实验名称：新闻话题分类

二、实验目的

使用大量的新闻文本进行训练，得到新闻的主题分类模型。

三、实验原理

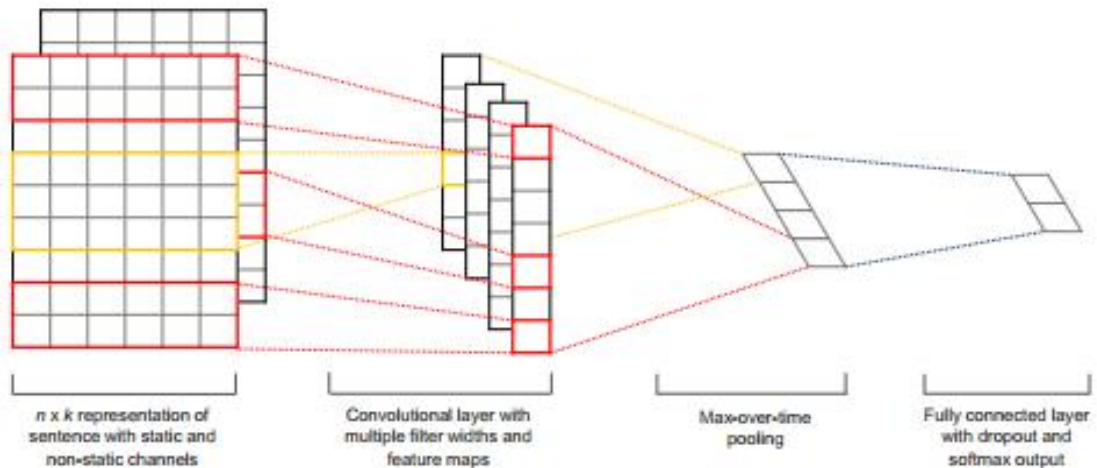
本次实验中应用了两种模型，以下按顺序说明它们的原理。另外 jieba 作为汉语文本预处理的重要工具，这里也对其进行说明。

1、Jieba 的使用

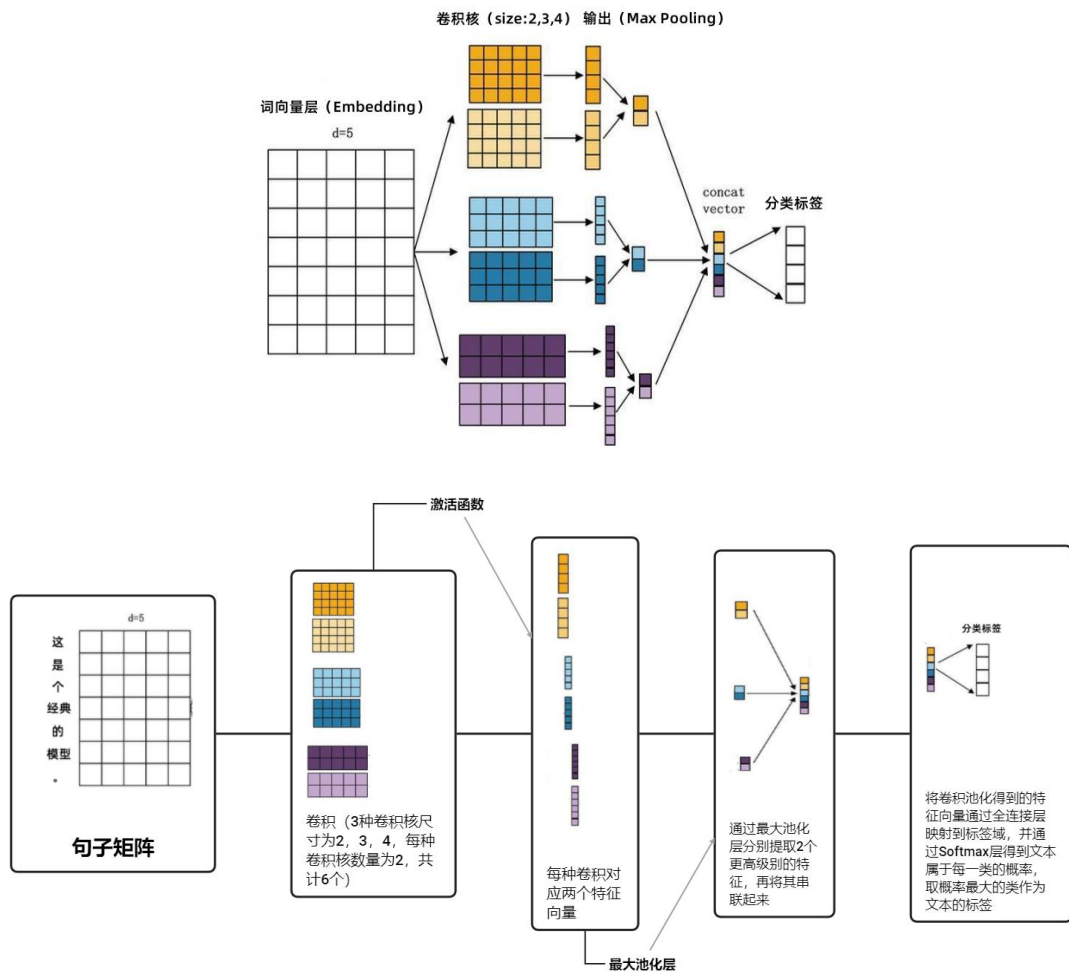
jieba 是 Python 第三方中文分词库，用于中文文本的分词处理，是 nlp 的重要工具。在分词时，有三种模式：精确模式不存在冗余数据，适合文本分析；全模式扫描所有可能的词语，速度快，但可能有歧义；搜索引擎模式对长词再次切分，提高召回率，适用于搜索引擎分词。本模型中采用第一个模式，具体的分词原理这里不做引用了。

2、TextCNN

TextCNN 将卷积神经网络应用到文本分类中，提取文本特征。



TextCNN 包含 Embedding 层、Convolution 层、Pooling 层和 Fully Connected，依次起到将输入的自然语言文本编码成分布式表示，提取 n-gram 特征，池化，映射到标签域的作用。



3、Transformer 文本主题识别

Transformer 模型的核心部分是自注意力机制, 捕捉词与词之间的依赖关系, 理解上下文信息。由于其本身不包含序列的顺序信息, 采用位置编码确保序列。在文本主题识别的过程中, 将待识别的文本作为输入序列, 通过分词、词嵌入等预处理步骤, 将文本转换为模型可以处理的数值表示, 进而获得 Transformer 编码。然后用各种方法进行主题识别。作业中采用了镜像网站训练好的模型, 这里不对其原理进行赘述了。

四、实验过程与成果

数据使用:

数据集: `cnews.train.txt` → `cnews.train.xlsx`

1、观察数据:

数据集给出的是新闻主题和文本, 文本内容有几百字, 较长, 考虑使用专门处理文字的方法。本题目是一道自然语言处理 (nlp) 的题目。

2、Jieba TextCNN

首先使用 TextCNN。为了给出便于模型训练的数据, 在预处理时采用了 jieba 包, 以下是预处理文件运行的情况:

```
C:\Users\zhs20\.conda\envs\jqxx\python.exe "C:\Users\zhs20\Desktop\【机器学习期末大作业】2212266+张恒硕\news_classification\jieba_TextCNN\process.py"
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\zhs20\AppData\Local\Temp\jieba.cache
Loading model cost 0.339 seconds.
Prefix dict has been built successfully.

进程已结束, 退出代码为 0
```

处理后的数据集中包含标签序号列和词向量列，还被分成了训练集、验证集、测试集三个文件，以下展示文件的部分内容：

	A	B	C	D	E	F
1	category	indices				
2		2	2001, 261, 2001, 2001, 2001, 166, 1722, 2001, 84, 508, 51			
3		4	2001, 1057, 2001, 2001, 2001, 2001, 2001, 2001, 3, 2, 12,			
4		5	2001, 2001, 2001, 236, 2001, 2001, 368, 2001, 2001, 1414			
5		2	1281, 2001, 2001, 450, 48, 2001, 421, 143, 2001, 2001, 16			
6		1	2001, 2001, 236, 2001, 184, 2001, 2001, 2001, 2001, 6, 20			
7		1	1284, 2001, 2001, 2001, 2001, 194, 226, 2001, 2001, 2001			
8		7	2001, 2001, 2001, 2001, 62, 2001, 2001, 2001, 483, 546, 3			
9		2	2001, 2001, 2001, 2001, 26, 2001, 268, 1462, 20, 2001, 70			

利用训练集和验证集训练模型，训练过程如下：

```
C:\Users\zhs20\.conda\envs\jxxx\python.exe "C:\Users\zhs20\Desktop\【机器学习期末大作业】2212266+张恒硕\news_classification\jieba_TextCNN\train.py"
Epoch 1/10: 100%|██████████| 500/500 [00:23<00:00, 20.94it/s, loss=0.679]
Epoch 1/10, Train Loss:0.6786909350156785, Val Loss: 0.2559353157877922, Val Acc: 0.92475
Epoch 2/10: 100%|██████████| 500/500 [00:22<00:00, 22.00it/s, loss=0.197]
Epoch 2/10, Train Loss:0.1974925444945693, Val Loss: 0.18304773035645486, Val Acc: 0.944625
```

```
Epoch 7/10: 100%|██████████| 500/500 [00:24<00:00, 20.08it/s, loss=0.00481]
Epoch 7/10, Train Loss:0.004809622956207022, Val Loss: 0.18822967141866684, Val Acc: 0.9515
Epoch 8/10: 100%|██████████| 500/500 [00:24<00:00, 20.13it/s, loss=0.00285]
Epoch 8/10, Train Loss:0.0028492391869658603, Val Loss: 0.20143956138193608, Val Acc: 0.94975
Epoch 9/10: 100%|██████████| 500/500 [00:24<00:00, 20.10it/s, loss=0.00464]
Epoch 9/10, Train Loss:0.004642979440919589, Val Loss: 0.21408233185112477, Val Acc: 0.947625
Epoch 10/10: 100%|██████████| 500/500 [00:24<00:00, 20.11it/s, loss=0.00866]
Epoch 10/10, Train Loss:0.008656072619138285, Val Loss: 0.27874381124973296, Val Acc: 0.9345
```

以下表格选取了第 7 个 epoch 的训练结果：

Train Loss	Val Loss	Val Acc
0.004809622956207022	0.18822967141866684	0.9515

使用测试集测试模型的准确度，结果如下：

```
C:\Users\zhs20\.conda\envs\jxxx\python.exe "C:\Users\zhs20\Desktop\【机器学习期末大作业】2212266+张恒硕\news_classification\jieba_TextCNN\test.py"
Test Accuracy: 0.9286
进程已结束，退出代码为 0
```

（多次训练时，最高达到过 0.96）

可以发现，模型的结果十分好，基本可以准确的识别新闻的主题。在耗时上，训练过程耗时几分钟，测试过程耗时几秒，都在可以接受的范围内。这说明，本模型适用于大量文本的新闻主题识别任务。

3、Transformer

其次，考虑直接使用 Transformer 训练模型，使用镜像网站 <https://hf-mirror.com/> 下载的 bert-base-chinese。

然而，这种方法对算力的要求过高，即使裁剪文本也难以运行。这里仅展示其运行的状态，无法给出最终的结果：

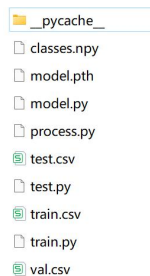
```
C:\Users\zhs20\.conda\envs\jqxx\python.exe "C:\Users\zhs20\Desktop\【机器学习期末大作
C:\Users\zhs20\.conda\envs\jqxx\lib\site-packages\transformers\utils\generic.py:2
    torch.utils._pytree._register_pytree_node(
C:\Users\zhs20\.conda\envs\jqxx\lib\site-packages\transformers\utils\generic.py:2
    torch.utils._pytree._register_pytree_node(
Some weights of BertForSequenceClassification were not initialized from the model
You should probably TRAIN this model on a down-stream task to be able to use it f
C:\Users\zhs20\.conda\envs\jqxx\lib\site-packages\accelerate\accelerator.py:444:
dataloader_config = DataLoaderConfiguration(dispatch_batches=None)
    warnings.warn(
0%|          | 0/7500 [00:00<?, ?it/s]Traceback (most recent call last):
```

这里警告了一些版本更新上的问题。

五、代码与解析

代码有两套，具体见附件，以下对其进行逐个解析

1、Jieba TextCNN：对各文件的作用进行总述，具体各部分代码的作用见注释。



process.py 该文件定义了对文本的预处理方法。处理文本列时，通过使用 jieba，最终得到去掉虚词、数字的汉语词汇表的索引列表。而处理标签列时，使用数字序号进行编码。在划分为训练集、验证集、测试集后，输出。同时输出 **classes.npy** 作为类别索引。

```
import re
import jieba
import numpy as np
import pandas as pd
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# 分词与筛选
def process(in_text):
    source_list = list(jieba.cut(in_text, cut_all=False))
    # 设定虚词为停用词
    stopwords = {'的', '了', '在', '是', '我', '有', '和', '也', '就', '都', '不', '你', '他', '她', '它', '我们', '你们', '他们', '自己', '这', '那', '上', '下', '要', '说', '着', '么', '吧', '会', '过', '去', '还', '很', '太', '都', '又', '就', '也', '还', '了', '啊', '吗', '吧', '呢', '着', '啦', '的',
```

```

        '得', '地', '个', '把', '对', '等', '但', '而', '和', '还',
'或', '可', '却', '如果', '虽然',
        '但是', '因为', '所以', '因此', '于是', '因此', '这样', '那样',
'比如', '例如', '同时', '然后', '然后',
        '还有', '另外', '此外', '同样', '既然', '除了', '只有', '只要',
'不仅', '不管'}

# 去掉虚词、数字，只保留剩下的汉字
filtered_list = [word for word in source_list if
                  word not in stopwords and not word.isdigit() and not
re.match(r'^[\u4e00-\u9fa5]', word)]

return filtered_list

# 构建词汇表
def build_vocab(tokens):
    # 统计词汇出现频次
    word_counts = Counter(tokens)
    # 选取频次最高的部分词汇
    vocab_size = 2000
    vocab = [word for word, count in word_counts.most_common(vocab_size)]
    # 在词汇表列表末尾添加两个特殊标记
    vocab.append('<PAD>')
    vocab.append('<UNK>')
    # 返回词汇表和索引
    return vocab, {word: idx for idx, word in enumerate(vocab)}

# 填充到相同长度
def build_indices(tokens, word_to_idx, max_seq_length):
    indices = [word_to_idx.get(token, word_to_idx['<UNK>']) for token in
tokens]
    indices += [word_to_idx['<PAD>']] * (max_seq_length - len(indices))
    indices = indices[:max_seq_length]
    return np.array(indices)

# 将原文件转换为xlsx格式后导入，并标注列名
df = pd.read_excel('../cnews.train.xlsx', header=None,
names=['category', 'content'], engine='openpyxl')
# 应用上述函数处理文本列，最后获得字符串
df['tokens'] = df['content'].apply(process)
all_tokens = [token for news_tokens in df['tokens'] for token in
news_tokens]
vocab, vocabulary = build_vocab(all_tokens)

```



```

df['indices'] = df['tokens'].apply(lambda tokens: build_indices(tokens,
vocabulary, 200))
df['indices'] = df['indices'].apply(lambda x: ','.join(map(str, x)))
# 类别列转换为数值编码
label_encoder = LabelEncoder()
df['category'] = label_encoder.fit_transform(df['category'])
# 保留有用列，划分训练集和测试集
df = df[['category', 'indices']]
train_df, test_df = train_test_split(df, test_size=0.2, random_state=42)
train_df, val_df = train_test_split(train_df, test_size=0.2,
random_state=42)
# 保存数据
np.save('classes.npy', label_encoder.classes_)
train_df.to_csv('train.csv', index=False)
val_df.to_csv('val.csv', index=False)
test_df.to_csv('test.csv', index=False)

```

`model.py` 定义了嵌入层、卷积层、池化层和分类层。

```

import torch
import torch.nn as nn
import torch.nn.functional as f

class Pool(nn.Module):
    def __init__(self):
        super(Pool, self).__init__()

    def forward(self, x):
        return f.max_pool1d(x, kernel_size=x.size(2)).squeeze(-1)

class TextCNN(nn.Module):
    def __init__(self, num_classes, num_embeddings, embedding_dim=128,
kernel_sizes=[3, 4, 5], num_channels=[100, 100, 100]):
        # 调用父类的初始化方法
        super(TextCNN, self).__init__()
        # 类别数量
        self.num_classes = num_classes
        # 嵌入层
        self.embedding = nn.Embedding(num_embeddings, embedding_dim)
        # 卷积层（一维卷积、批量归一化层、ReLU 激活函数）
        self.cnn_layers = nn.ModuleList(
            [nn.Sequential(nn.Conv1d(in_channels=embedding_dim if
self.embedding is not None else 1,
out_channels=c, kernel_size=k),

```

```

nn.BatchNorm1d(c), nn.ReLU(inplace=True))
        for c, k in zip(num_channels, kernel_sizes)]
    # 池化层
    self.pool = Pool()
    # 分类层 (Dropout 层、全连接层)
    self.classify = nn.Sequential(nn.Dropout(p=0.2),
nn.Linear(sum(num_channels), self.num_classes))

    # 前向传播函数
    def forward(self, input):
        input = self.embedding(input)
        input = input.permute(0, 2, 1)
        y = [self.pool(layer(input)).squeeze(-1) for layer in
self.cnn_layers]
        y = torch.cat(y, dim=1)
        return self.classify(y)

```

`train.py` 使用自定义的 `TextDataset` 函数，完成数据载入后，指定模型的参数，进行训练与验证。

```

import torch
import torch.nn as nn
from torch import optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
import pandas as pd
from tqdm import tqdm
from model import TextCNN

# 加载、处理数据集
class TextDataset(Dataset):
    def __init__(self, df):
        self.labels = df['category'].values
        self.texts = df['indices'].apply(lambda x: np.array(list(map(int,
x.replace('\n', '').split(',')))).values

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):
        return torch.tensor(self.texts[idx], dtype=torch.long),
torch.tensor(self.labels[idx], dtype=torch.long)

# 训练模型函数

```

```

def train(model, criterion, optimizer, train_loader, val_loader, epochs,
device):
    model.to(device)
    model.train()
    best_val_acc = 0
    for epoch in range(epochs):
        running_loss = 0.0
        train_tqdm=tqdm(train_loader, desc=f"Epoch {epoch + 1}/{epochs}")
        for inputs, labels in train_tqdm:
            inputs, labels = inputs.to(device), labels.to(device)
            # 清除之前的梯度
            optimizer.zero_grad()
            # 前向传播, 得到预测输出
            outputs = model(inputs)
            # 损失
            loss = criterion(outputs, labels)
            # 反向传播损失, 计算梯度并累计
            loss.backward()
            running_loss += loss.item()
            # 使用优化器更新模型权重
            optimizer.step()
            train_tqdm.set_postfix(loss=running_loss / len(train_loader))
        # 验证
        val_loss, val_acc=evaluate(model, criterion, val_loader, device)
        print(f"Epoch {epoch + 1}/{epochs}, "
              f"Train Loss:{running_loss / len(train_loader)}, Val Loss:
{val_loss}, Val Acc: {val_acc}")
        if val_acc > best_val_acc:
            best_val_acc = val_acc
            torch.save(model.state_dict(), 'model.pth')

# 评估函数
def evaluate(model, criterion, data_loader, device):
    model.eval()
    total_loss = 0.0
    correct = 0
    with torch.no_grad():
        for inputs, labels in data_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            total_loss += loss.item()
            _, predicted = torch.max(outputs, 1)

```



```

        correct += (predicted == labels).sum().item()
    return total_loss / len(data_loader), correct /
len(data_loader.dataset)

train_df = pd.read_csv('train.csv')
val_df = pd.read_csv('val.csv')
# 加载、包装数据
train_dataset = TextDataset(train_df)
val_dataset = TextDataset(val_df)
train_loader = DataLoader(train_dataset, 64, shuffle=True)
val_loader = DataLoader(val_dataset, 64)
# 标签种类, 词汇表长度 (+2)
model = TextCNN(num_classes=len(train_df['category'].unique()),
num_embeddings=2002)
# 损失函数为交叉熵损失
criterion = nn.CrossEntropyLoss()
# Adam 优化器
optimizer = optim.Adam(model.parameters(), lr=0.001)
# 训练轮次
epochs = 10
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
train(model, criterion, optimizer, train_loader, val_loader, epochs,
device)

```

test.py 同样使用自定义的 **TextDataset** 函数, 用测试集测试模型的准确度。

```

import torch
from torch.utils.data import Dataset, DataLoader
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from model import TextCNN

# 加载、处理数据集
class TextDataset(Dataset):
    def __init__(self, df):
        self.labels = df['category'].values
        self.texts = df['indices'].apply(lambda x: np.array(list(map(int,
x.replace('\n', ' ').split(','))))).values

    def __len__(self):
        return len(self.labels)

    def __getitem__(self, idx):

```

```

        return torch.tensor(self.texts[idx], dtype=torch.long),
torch.tensor(self.labels[idx], dtype=torch.long)




# 导入测试集和相关文件
test_df = pd.read_csv('test.csv')
label_encoder = LabelEncoder()
label_encoder.classes_ = np.load('classes.npy', allow_pickle=True)
test_dataset = TextDataset(test_df)
test_loader = DataLoader(test_dataset, 64)
model = TextCNN(num_classes=len(label_encoder.classes_),
num_embeddings=2002)
model.load_state_dict(torch.load('model.pth'))
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# 测试
model.eval()
results = []
labels = []
with torch.no_grad():
    for texts, true_labels in test_loader:
        texts = texts.to(device)
        outputs = model(texts)
        _, predicted = torch.max(outputs, 1)
        predicted_classes = predicted.cpu().numpy()
        results.extend(predicted_classes)
        true_labels_cpu = true_labels.cpu().numpy()
        labels.extend(true_labels_cpu)

# 测试正确率
results = np.array(results)
labels = np.array(labels)
accuracy = np.mean(results == labels)
print(f"Test Accuracy: {accuracy:.4f}")

```

2、Transformer: transformer.py, 具体内容见注释
还有 bert_localpath 中下载的资源。

 config.json
 pytorch_model.bin
 vocab.txt

```

import pandas as pd
import torch
from torch.utils.data import DataLoader, Dataset
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

```

```

from transformers import BertTokenizer, BertForSequenceClassification,
Trainer, TrainingArguments

class MyDataset(Dataset):
    def __init__(self, texts, labels, attention_masks):
        self.texts = texts
        self.labels = labels
        self.attention_masks = attention_masks

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        return self.texts[idx], self.attention_masks[idx],
self.labels[idx]

# 数据加载和预处理
df = pd.read_excel('../cnews.train.xlsx', header=None,
names=['category', 'content'], engine='openpyxl')
texts = df['content'].tolist()
label_encoder = LabelEncoder()
df['category_encoded'] = label_encoder.fit_transform(df['category'])
# 加载模型和分词器
model = BertForSequenceClassification.from_pretrained('bert_localpath',
num_labels=len(set(df['category_encoded'])))
tokenizer = BertTokenizer.from_pretrained('bert_localpath')
# 编码文本数据
encoded_texts = tokenizer(texts, truncation=True, padding=True,
return_tensors='pt', max_length=100)
encoded_input_ids = encoded_texts['input_ids']
encoded_attention_masks = encoded_texts['attention_mask']
# 划分数据集并加载
train_df, val_df = train_test_split(df, test_size=0.2, random_state=42)
train_labels =
torch.tensor(label_encoder.transform(train_df['category'].tolist()),
dtype=torch.long)
val_labels =
torch.tensor(label_encoder.transform(val_df['category'].tolist()),
dtype=torch.long)
train_texts = encoded_input_ids[train_df.index]
train_attention_masks = encoded_attention_masks[train_df.index]
val_texts = encoded_input_ids[val_df.index]

```

```

val_attention_masks = encoded_attention_masks[val_df.index]
train_dataset = MyDataset(train_texts, train_labels,
train_attention_masks)
val_dataset = MyDataset(val_texts, val_labels, val_attention_masks)
train_dataloader = DataLoader(train_dataset, batch_size=16,
shuffle=True)
val_dataloader = DataLoader(val_dataset, batch_size=16, shuffle=False)
# 设定模型参数
training_args = TrainingArguments(output_dir='results',
num_train_epochs=3,
                                per_device_train_batch_size=16,
per_device_eval_batch_size=64,
                                warmup_steps=500, weight_decay=0.01,
logging_steps=10, evaluation_strategy="epoch")
# 训练模型
trainer = Trainer(model=model, args=training_args,
train_dataset=train_dataset, eval_dataset=val_dataset)
trainer.train()
trainer.save_model('results/best_model')

```

六、分析

本问题给出了大量的文本内容，这对处理提出了很高的要求。直接使用 Transformer 时，即使大幅裁剪文本内容，设备仍无法得到结果。而通过使用 jieba 进行分词，得到的词向量适用于 textCNN 模型，得到了良好的结果。然而 Transformer 并不接受这种分词的词向量，可见，在建立不同的模型时，可以采用的预处理的方法是不同的。