新闻文本分类实验报告

|  |  |
| --- | --- |
| **课程名称：** | 机器学习 |
| **课程类别：** | 专业选修课 |
| **任课教师：** | 文勇 |
| **授课时间：** | 2024 年3月8日至 2024年7月5日 |
| **学　　号：** | 202312143002062 |
| **姓　　名：** | 顾佳凯 |
| **专业名称：** | 计算机科学与技术 |
| **所在学院：** | 人工智能学院 |

目录

[新闻文本分类实验报告 1](#_Toc170059482)

[1天池排名和分数截图 3](#_Toc170059483)

[2 赛题解析 4](#_Toc170059484)

[3 实验环境 5](#_Toc170059485)

[4 实验代码解析 7](#_Toc170059486)

[5 模型结构图 20](#_Toc170059487)

[6 损失函数描述 22](#_Toc170059488)

[7 实验总结 24](#_Toc170059489)

**1天池排名和分数截图**

在参与此次比赛的过程中，我先后提交了大约10次结果。截至2024年6月23日，取得的最佳成绩为0.9540，在排行榜上位列第145名。值得一提的是，这一版本的模型并未设置验证集。从5月底首次提交至今，历时近一个月的时间，期间在AutoDL GPU算力平台上的支出已接近500元人民币。

6月22日清晨，我再次启动了新一轮的训练，这一版本的模型加入了验证集，希望能够进一步提升模型的泛化能力。由于实验室GPU资源不到位，我只能选择租用AutoDL GPU算力平台进行训练。经过一天多的运行，5折交叉验证中的第一折尚未完成，刚充值的100元余额也即将耗尽。

经过慎重考虑，我决定暂停模型的训练，转而着手完善实验报告。在当前的条件下，受限于个人经济实力和设备性能，难以像拥有A100显卡的研究者那样对模型进行进一步调优。GPU算力平台的租用虽然提供了一定的便利，但高昂的费用也成为了进一步提升模型性能的阻碍。综合而言，对于个人参赛者而言，AI竞赛的门槛不可谓不高。在取得当前的成绩后，我已经感到心满意足，暂时不打算追求更高的名次了。



图一: 天池排名和分数截图

**2 赛题解析**

本次比赛的数据集由新闻文本组成，这些文本数据经过字符级别的匿名处理，以保护隐私。赛题提供了14个候选分类类别，分别是财经、彩票、房产、股票、家居、教育、科技、社会、时尚、时政、体育、星座、游戏和娱乐。

赛题数据包括三个部分：训练集、测试集A和测试集B。训练集包含20万条样本，测试集A和测试集B各包含5万条样本。为了防止选手人工标注测试集，赛题组织方对文本数据进行了字符级别的匿名处理。经过处理后的训练数据呈现出标签和文本两列的形式，标签与文本之间用制表符（\t）分隔。

在训练数据中，每个标签都对应一个数字，标签与数字之间的映射关系如下：

{'科技': 0, '股票': 1, '体育': 2, '娱乐': 3, '时政': 4, '社会': 5, '教育': 6, '财经': 7, '家居': 8, '游戏': 9, '房产': 10, '时尚': 11, '彩票': 12, '星座': 13}

赛题数据来源于互联网上的新闻，经过收集和匿名处理而得。因此，参赛者可以根据自己的专长，充分发挥创意，自行进行数据分析和特征工程。同时，赛题不限制使用任何外部数据和模型，给予了参赛者较大的自由度。

比赛采用宏观F1值（Macro F1 Score）作为评价标准，即所有类别的F1值的算术平均值。F1值是精确率（Precision）和召回率（Recall）的调和平均值，计算公式如下：

最终提交的结果文件需要与样例提交文件（sample\_submit.csv）的格式保持一致，并且文件后缀名必须为.csv。

为了帮助参赛者更好地理解和解决问题，赛题组织方在论坛中提供了往期参赛选手分享的比赛基线（Baseline）。参赛者可以通过学习这些基线，交流解题思路，从而获得更多的灵感和启发。

3 实验环境

在着手进行本次心跳信号分类预测实验之前,我对实验环境进行了慎重的选择和配置。综合考虑了计算资源、易用性和移植性等因素,我最终决定在AutoDL平台上开展实验。相比于阿里云天池实验室,AutoDL能够提供更加灵活和强大的GPU算力支持。我申请了一张A100-PCIE-40GB显卡,以期获得优异的训练和推理性能。

AutoDL平台提供了丰富的基础镜像,我选择了Pytorch/2.0.0/3.8(ubuntu20.04)/11.8作为实验的起点。在此基础上,我使用conda创建了一个名为ml的python 3.12.3虚拟环境,以实现实验环境的隔离和管理。这不仅可以避免不同项目之间的依赖冲突,还能够方便地复现和迁移实验结果。

在ml虚拟环境中,除了基础镜像自带的软件包外,我还安装了一些额外的依赖库,以满足实验的特定需求。其中,数据处理和可视化方面,我选用了pandas 2.2.2、numpy 1.26.4和matplotlib 3.9.0;深度学习框架采用了pytorch 2.3.0;预训练模型加载和微调则使用了transformers 4.41.1。

在构建实验环境的过程中,我特别注意到了实验代码的组织和运行方式。与常见的Jupyter Notebook(ipynb)形式不同,我选择了使用Python脚本文件(py)的方式来编写和执行实验代码。这种方式有几个显著的优点:

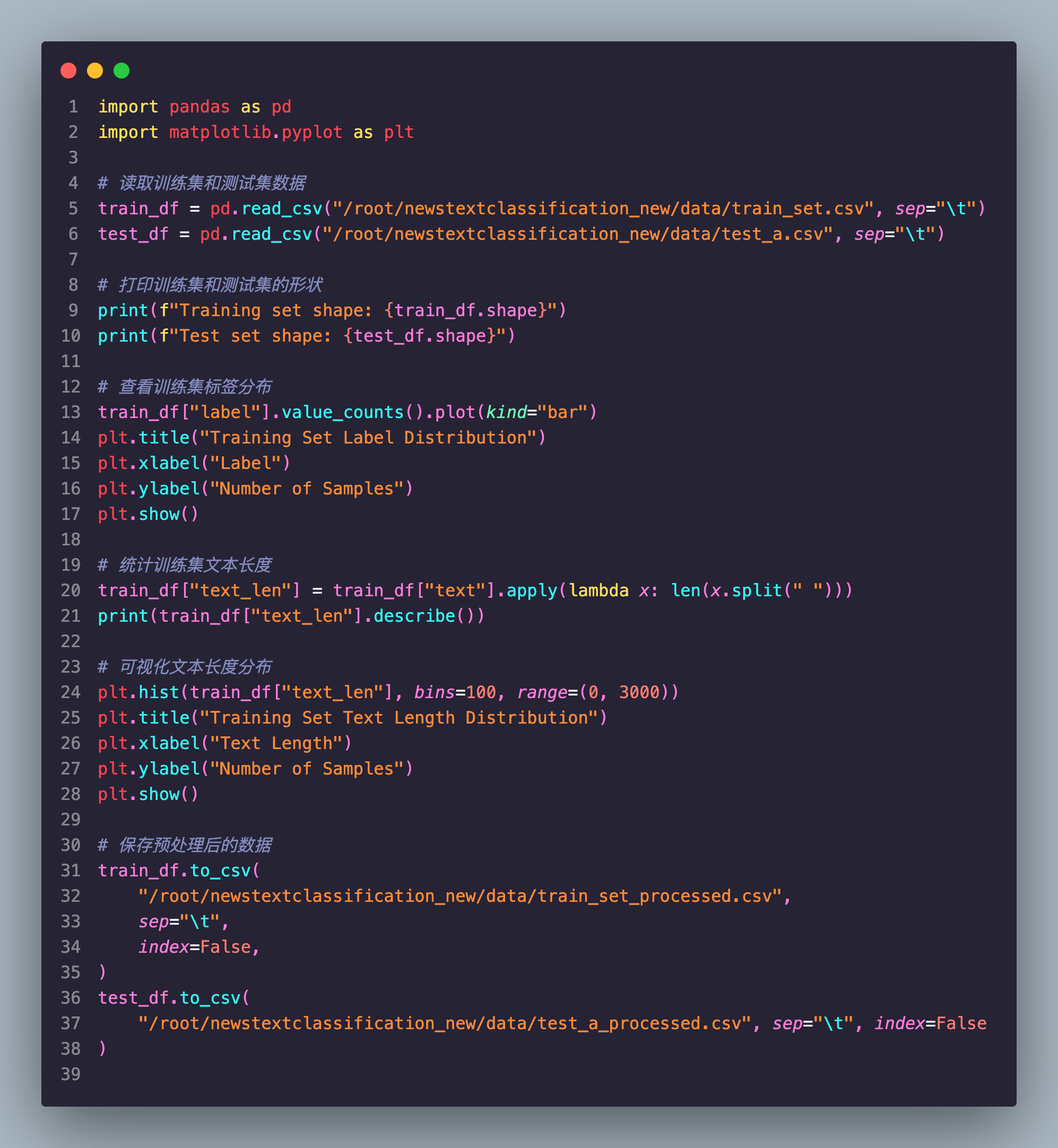
首先,py文件可以方便地在终端或命令行界面中运行,不需要依赖Jupyter Notebook服务。这使得实验代码的执行更加灵活和便捷。

其次,py文件更加适合在后台长时间运行,不会因为网络中断或终端断开而意外停止。在实际的实验过程中,训练深度学习模型往往需要几个小时甚至几天的时间。为了确保实验的连续性和稳定性,我使用了Linux系统自带的screen工具,将py文件放置在一个独立的screen会话中运行。这样,即使关闭了Jupyter Lab界面,实验也能够在后台继续进行,直到完成或手动终止。

最后,py文件的代码组织更加规范和清晰,便于模块化和重用。我将数据处理、模型定义、训练过程、评估指标等不同的功能模块分别封装在独立的函数或类中,并将它们组织在一个或多个py文件中。这种模块化的代码结构不仅提高了代码的可读性和可维护性,还方便了不同实验之间的代码复用和迁移。

总的来说,本次实验采用py文件+screen会话的方式,既满足了实验的性能和稳定性要求,又实现了代码的规范化和模块化组织,为后续的实验迭代和优化奠定了良好的基础。

4 实验代码解析



图二: 代码块一

这一阶段的主要目的是通过统计描述和可视化展示等方法，对数据集的基本特点和规律进行初步了解，为后续的数据预处理、特征工程和模型构建奠定基础。

首先，我们导入了两个核心的Python库：pandas和matplotlib.pyplot。pandas是一个强大的数据处理和分析工具，提供了高效的数据结构和操作函数，使我们能够方便地读取、清洗、转换和分析结构化数据。matplotlib.pyplot则是一个流行的数据可视化库，通过丰富的绘图函数和交互式界面，帮助我们直观地展示数据的分布、趋势和关联等特征。

接下来，我们使用pandas的read\_csv函数读取了训练集和测试集数据，并将它们分别存储在train\_df和test\_df两个DataFrame对象中。由于数据文件采用了制表符（\t）作为分隔符，因此我们通过设置sep参数来指定分隔符，确保数据能够被正确解析。

为了初步了解数据集的规模和特点，我们分别打印了训练集和测试集的形状（shape）。通过shape属性，我们可以快速获取数据集的样本数和特征数，这对于评估数据集的大小和复杂度具有重要意义。

在数据探索过程中，我们对训练集的标签分布进行了分析。通过使用value\_counts函数，我们统计了每个标签类别下的样本数量，并借助plot函数绘制柱状图，直观地展示不同类别的样本分布情况。通过观察标签分布图，我们可以判断数据集是否存在类别不平衡问题，并为后续的数据预处理和模型优化提供参考。（备注：由于是在命令行运行，柱状图并未展示。）

考虑到文本长度是文本分类任务中的一个重要特征，我们对训练集中的文本长度进行了统计和分析。首先，我们定义了一个匿名函数，通过split方法将文本按照空格分割，并计算分割后的词语数量，从而得到每个样本的文本长度。然后，我们使用describe函数生成了文本长度的统计摘要，包括最小值、最大值、均值、中位数等关键指标，以便全面了解文本长度的分布特点。

为了更直观地展示文本长度的分布情况，我们还绘制了文本长度的直方图。通过hist函数，我们将文本长度划分为100个区间，并统计每个区间内的样本数量。同时，我们设置了适当的图表标题、横纵坐标标签等元素，以清晰地呈现文本长度的分布特点。通过观察直方图，我们可以发现文本长度的集中区间、离散程度等特征，为后续的特征工程和模型设计提供重要参考。（备注：由于是在命令行运行，直方图并未展示。）

最后，为了便于后续的实验流程，我们将训练集和测试集数据保存为CSV文件。通过调用to\_csv函数，并指定文件路径、分隔符和索引等参数，我们可以将DataFrame对象持久化为文件，以便在其他代码模块中方便地读取和使用。需要注意的是，这里保存的数据与原始数据并无差异，因为我们尚未对数据进行实质性的预处理操作。



图三: 代码块二

这段代码定义了一个名为NewsDataset的数据集类,用于处理新闻文本分类任务的数据。NewsDataset类继承自PyTorch的Dataset类,Dataset类是一个抽象类,用于表示数据集。我们需要重写Dataset类的\_\_len\_\_和\_\_getitem\_\_方法,以定义数据集的长度和如何获取单个样本。

NewsDataset类的\_\_init\_\_方法是其初始化方法,接受data(数据)、tokenizer(分词器)、max\_len(最大序列长度)和is\_test(是否为测试集)作为参数。在初始化过程中,这些参数被赋值给类的实例属性,以便在其他方法中使用。\_\_len\_\_方法返回数据集的长度,即数据的行数。

\_\_getitem\_\_方法根据给定的索引(index)返回一个样本。当我们使用DataLoader加载数据时,DataLoader会内部调用\_\_getitem\_\_方法来获取指定索引的样本。在\_\_getitem\_\_方法中,首先通过iloc方法获取指定索引的文本数据。然后,使用tokenizer的encode\_plus方法对文本进行编码,将文本转换为BERT模型可以接受的输入格式。encode\_plus方法的参数包括:add\_special\_tokens(添加BERT所需的特殊标记,如[CLS]和[SEP])、max\_length(最大序列长度)、return\_token\_type\_ids(返回token类型ID)、padding(填充到最大长度)、truncation(截断到最大长度)、return\_attention\_mask(返回attention mask)和return\_tensors(返回PyTorch tensor)。这些参数的设置确保了所有输入序列的长度一致,并且生成了BERT模型所需的各种输入张量。

根据is\_test参数的值,\_\_getitem\_\_方法返回不同的样本格式。如果is\_test为True(测试集),则返回文本、编码后的输入ID和attention mask;如果is\_test为False(训练集或验证集),则除了返回文本、编码后的输入ID和attention mask外,还返回对应的标签(label)。标签是通过iloc方法获取的,并转换为PyTorch的long类型张量。

总的来说,这段代码实现了一个自定义的数据集类NewsDataset,它继承自PyTorch的Dataset类,并重写了\_\_len\_\_和\_\_getitem\_\_方法。通过使用BERT tokenizer对文本进行编码,NewsDataset类将原始文本数据转换为BERT模型所需的输入格式,并根据is\_test参数的值返回不同的样本格式。这为后续的数据加载和模型训练提供了便利。



图四: 代码块三

这段代码定义了一个名为BertLSTMForNewsCls的PyTorch模型类,用于基于BERT和LSTM的新闻文本分类任务。下面是对这段代码的详细解析:

BertLSTMForNewsCls类继承自nn.Module类,nn.Module是PyTorch中所有神经网络模块的基类。在\_\_init\_\_方法中,我们定义了模型的各个组件。首先,通过BertModel.from\_pretrained方法加载预训练的BERT模型,并将其赋值给self.bert属性。接下来,定义了一个双向LSTM层(nn.LSTM),用于对BERT的输出进行进一步的序列建模。LSTM层的输入维度为768(BERT的隐藏状态维度),隐藏状态维度为hidden\_dim,层数为num\_layers,并设置batch\_first=True表示以批次为第一维度。

在LSTM层之后,定义了一个多头注意力层(nn.MultiheadAttention),用于捕捉序列中的长距离依赖关系。多头注意力层的嵌入维度为hidden\_dim\*2(双向LSTM的隐藏状态维度),注意力头数为num\_attention\_heads,并设置batch\_first=True表示以批次为第一维度。接下来,定义了一个线性层(nn.Linear),用于计算注意力权重。为了防止过拟合,还定义了一个dropout层(nn.Dropout),用于随机丢弃一部分神经元。

最后,定义了一个分类器(nn.Sequential),由两个线性层和一个ReLU激活函数组成,用于将LSTM的输出映射到指定的类别数。分类器的输入维度为hidden\_dim\*2,中间隐藏层的维度为hidden\_dim,输出维度为num\_classes。

forward方法定义了模型的前向传播过程,接受输入ID(input\_ids)和attention mask(attention\_mask)作为输入。首先,将输入传递给BERT模型,获取BERT的输出。然后,提取BERT的序列输出(sequence\_output),其形状为(batch\_size, sequence\_length, hidden\_size)。接下来,将序列输出传递给LSTM层,获取LSTM的输出(lstm\_output),形状为(batch\_size, sequence\_length, hidden\_size \* 2)。

在获得LSTM的输出后,将其传递给多头注意力层,得到注意力输出(attn\_output)。然后,通过一个线性层和tanh激活函数计算注意力权重(attention\_weights),并进行归一化。接着,将注意力权重应用于注意力输出,得到加权输出(weighted\_output),并对加权输出进行求和,得到池化输出(pooled\_output)。最后,对池化输出应用dropout,然后传递给分类器,得到最终的分类结果(logits),并将其作为模型的输出返回。

总的来说,这段代码实现了一个基于BERT和LSTM的新闻文本分类模型。通过将BERT的输出传递给LSTM层进行序列建模,并使用多头注意力机制捕捉长距离依赖关系,该模型能够有效地对新闻文本进行分类。同时,通过引入dropout和两层全连接层作为分类器,模型具有一定的泛化能力和非线性表示能力。

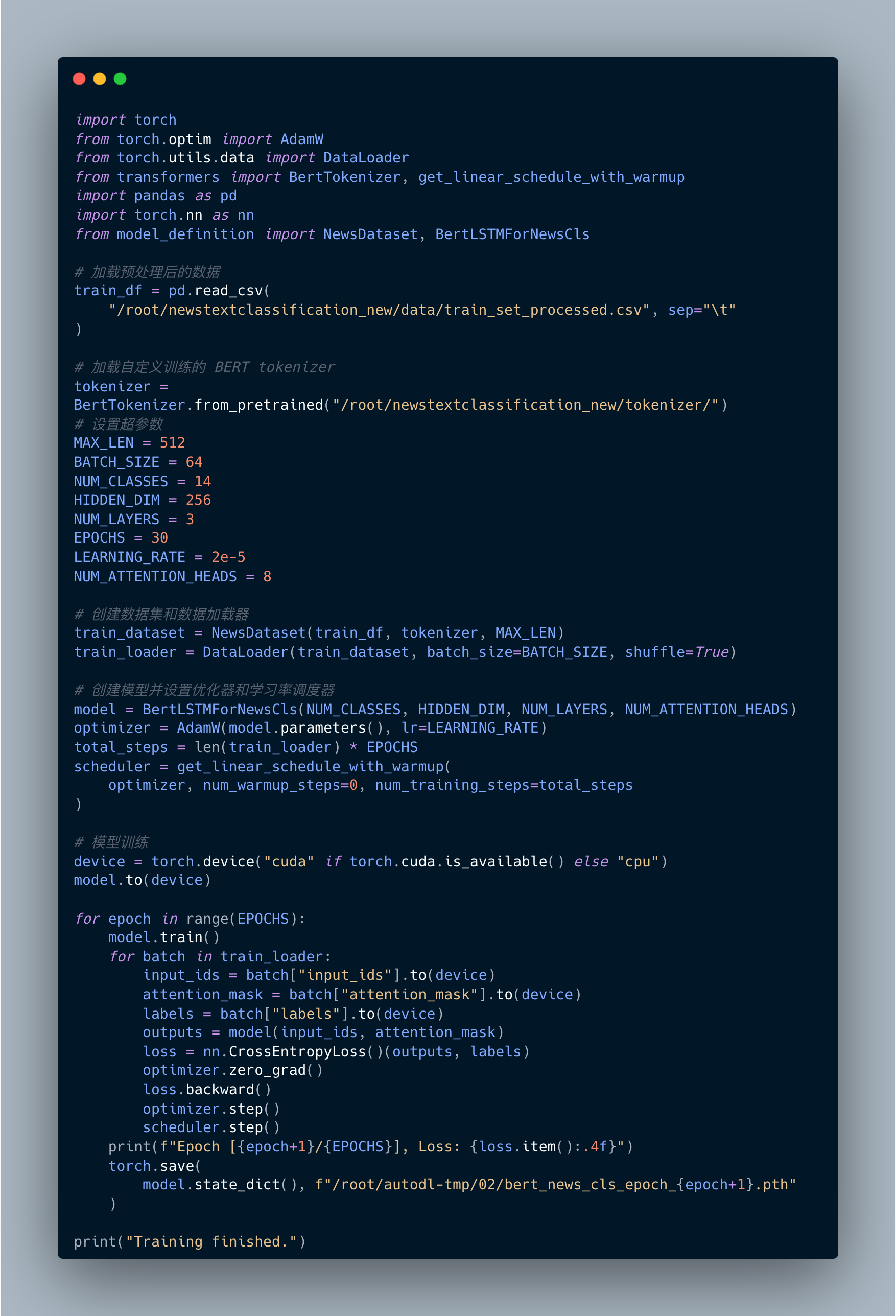


图五: 代码块四

这段代码主要目的是训练一个自定义的BERT tokenizer,用于将文本数据转换为模型可以处理的输入格式。首先,代码定义了数据集路径和tokenizer保存路径,然后读取训练集和测试集数据。接下来,将训练集和测试集的所有文本数据合并到一个列表中,用于训练tokenizer。为了方便tokenizer的训练,代码将合并后的文本数据保存到一个文件中,每行一个文本。

在准备好文本数据后,代码初始化了一个BertWordPieceTokenizer对象,并调用其train方法进行训练。train方法接受多个参数,包括用于训练的文本文件、词汇表大小、最小词频等。这里设置了词汇表大小为30522,最小词频为2,与预训练的BERT模型保持一致。训练完成后,代码将训练好的tokenizer保存到指定路径,以便后续的模型训练和预测使用。

通过训练自定义的BERT tokenizer,我们可以将文本数据转换为模型可以理解和处理的数字化表示,为后续的模型训练和预测做好准备。同时,使用自定义的tokenizer可以更好地适应特定领域的文本数据,提高模型的性能和泛化能力。



图六: 代码块五

这段代码是模型训练的主要部分,它加载预处理后的数据、自定义训练的BERT tokenizer,然后设置了一些超参数,如最大序列长度、批次大小、类别数、隐藏层维度、LSTM层数、训练轮数、学习率和注意力头数等。接下来,代码创建了NewsDataset对象和DataLoader对象,用于加载和批次化训练数据。

在模型创建部分,代码实例化了BertLSTMForNewsCls模型,并设置了AdamW优化器和学习率调度器。优化器用于更新模型的参数,学习率调度器用于在训练过程中动态调整学习率,以提高训练效果和收敛速度。

在模型训练部分,代码首先将模型移动到适当的设备上(GPU或CPU),然后进行多个epoch的训练。在每个epoch中,代码遍历数据加载器中的每个批次,将输入数据移动到设备上,并将其传递给模型进行前向传播。然后,代码计算损失函数(交叉熵损失),执行反向传播,更新模型参数,并更新学习率。在每个epoch结束时,代码打印当前的epoch数和损失值,并将当前epoch的模型状态字典保存到文件中,以便后续的模型评估和预测使用。

通过这段代码,我们可以使用自定义的数据集、tokenizer和模型架构,对新闻文本分类任务进行训练。通过设置合适的超参数和训练策略,我们可以获得一个性能良好的模型,用于后续的预测和评估任务。



图七: 代码块六

这段代码是模型预测的主要部分,它加载了训练好的模型和自定义的BERT tokenizer,然后读取待预测的数据,并使用训练好的模型对新数据进行预测。

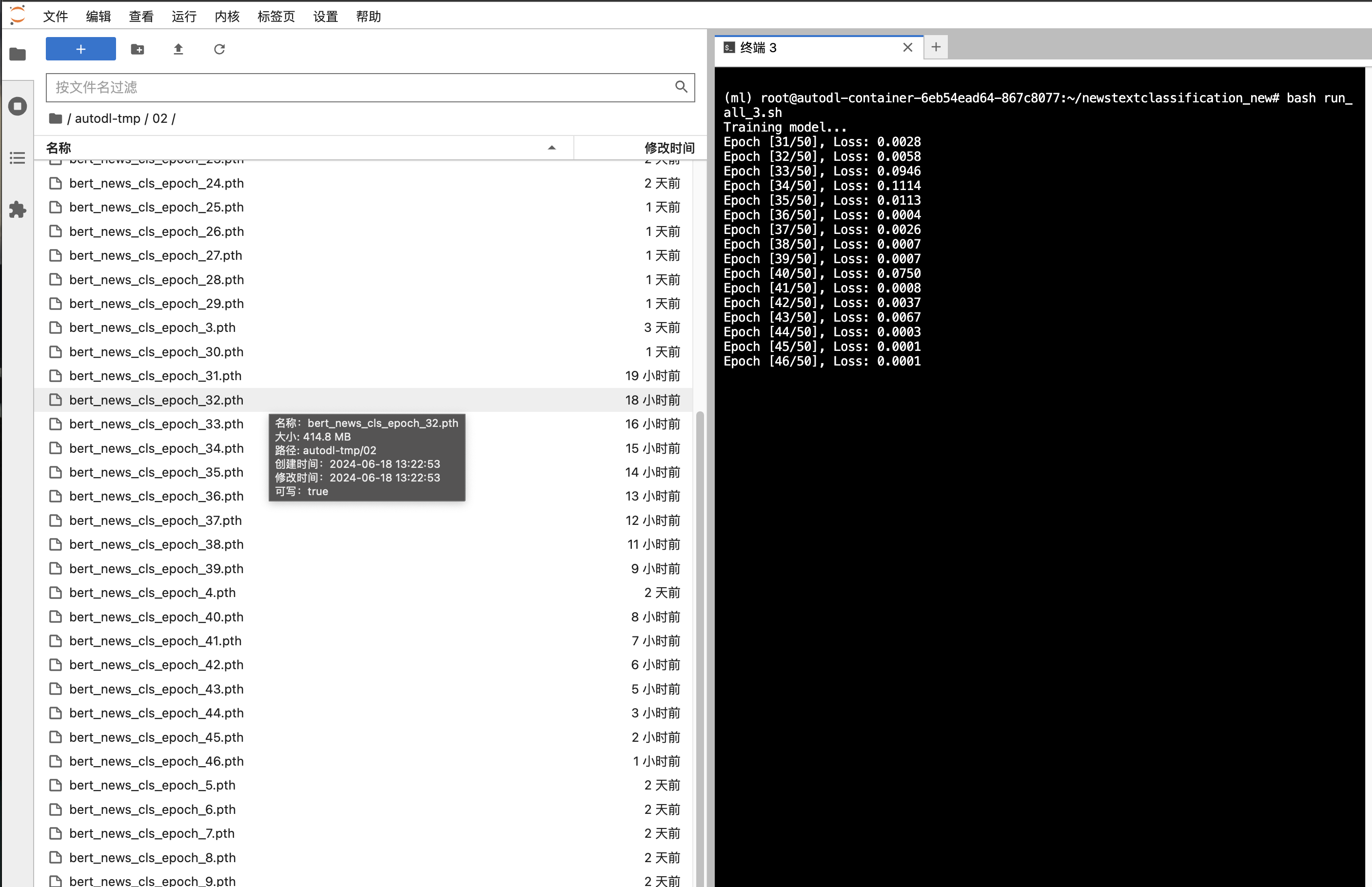
首先,代码设置了一些超参数,如类别数、隐藏层维度、LSTM层数、最大序列长度、批次大小和注意力头数等,这些参数与训练时使用的超参数保持一致。然后,代码加载了训练好的BertLSTMForNewsCls模型,并将其移动到适当的设备上(GPU或CPU)。

接下来,代码读取待预测的数据,并使用自定义训练的BERT tokenizer对其进行预处理,得到NewsDataset对象和DataLoader对象,用于加载和批次化预测数据。

在模型预测部分,代码将模型设置为评估模式,然后遍历数据加载器中的每个批次,将输入数据移动到设备上,并将其传递给模型进行前向传播。通过torch.max函数,代码获取模型输出的最大值对应的类别索引,作为预测结果。这些预测结果被收集并存储在一个列表中。

最后,代码将预测结果转换为DataFrame对象,并将其保存到文件中,生成最终的提交文件。

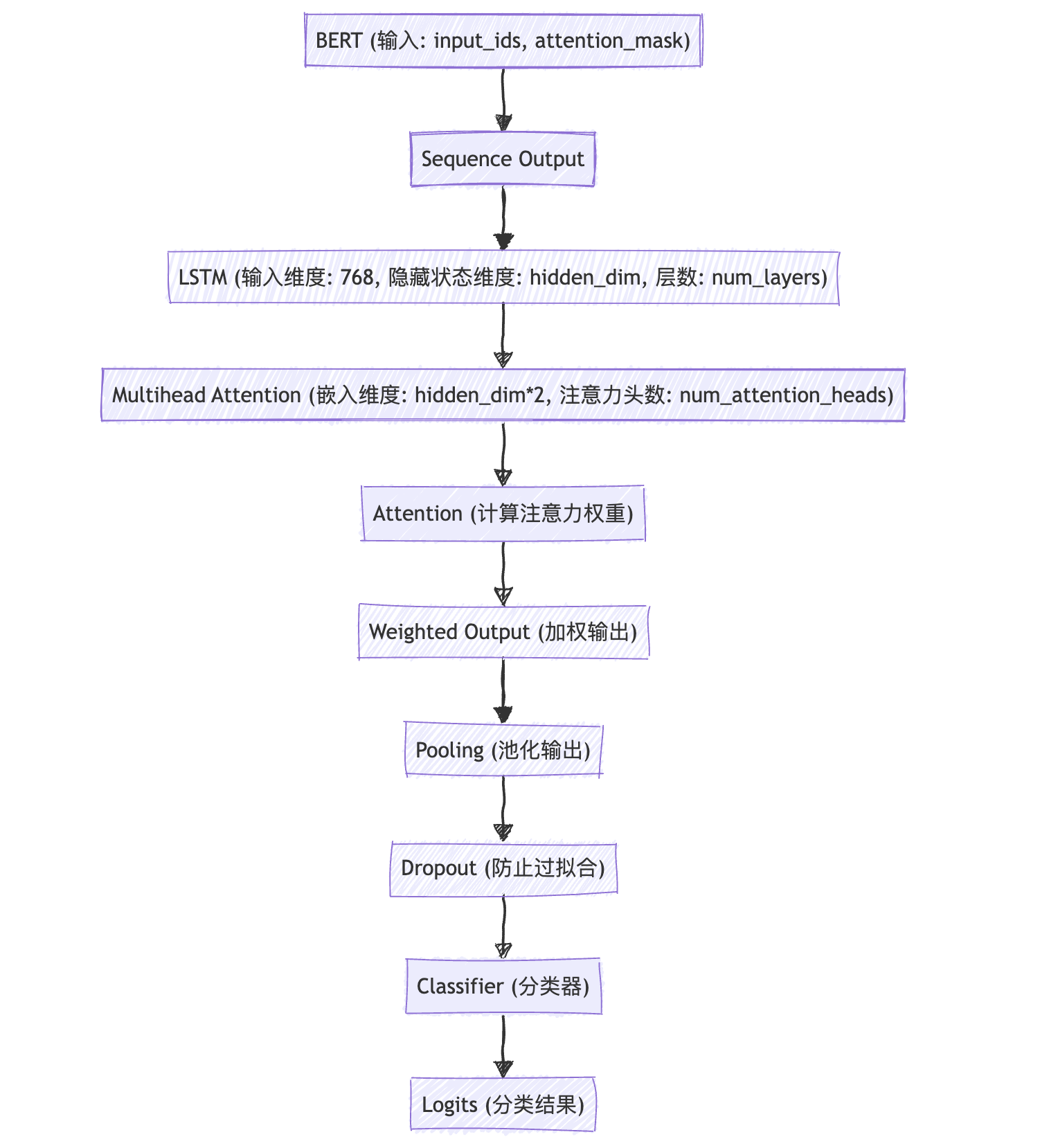
通过这段代码,我们可以使用训练好的模型对新的数据进行预测,并生成符合提交格式的文件。这样,我们就可以将预测结果提交到比赛平台,并评估模型在实际应用中的性能表现。同时,我们也可以使用这个预测脚本对其他类似的数据集进行预测,扩展模型的应用范围。



图八: 二次训练终端演示

上述代码解析展示的是30个epoch的训练，后续又再此基础上训练了20个epoch，50个epoch才是本实验代码的最终形态。

5 模型结构图



图九: 模型结构图

为了更好地理解本次实验中使用的基于BERT和LSTM的新闻文本分类模型的结构和工作原理,我绘制了模型的结构示意图,并对其进行了详细的说明。这个示意图直观地展示了数据在模型中的流动和转换过程,帮助我深入理解了模型的内部机制。

从图中可以看出,该模型主要由BERT层、LSTM层、多头注意力机制、Attention机制、加权输出、Dropout层和分类器等组成。输入的文本数据首先经过BERT层进行特征提取,BERT层接收输入ID(input\_ids)和注意力掩码(attention\_mask),并输出序列表示(sequence\_output),其形状为(batch\_size, sequence\_length, hidden\_size)。

接下来,BERT层的输出被传递到LSTM层进行进一步的序列建模。LSTM层的输入维度为768(与BERT的隐藏状态维度一致),隐藏状态维度为hidden\_dim,层数为num\_layers。同时,LSTM层采用了双向结构,能够同时捕捉序列的前向和后向信息。LSTM层的输出形状为(batch\_size, sequence\_length, hidden\_size \* 2)。

在LSTM层之后,模型引入了多头注意力机制,用于捕捉序列中的长距离依赖关系。多头注意力机制的输入为LSTM层的输出,嵌入维度为hidden\_dim \* 2(双向LSTM的隐藏状态维度),注意力头数为num\_attention\_heads。多头注意力机制的输出(attn\_output)形状与输入相同,为(batch\_size, sequence\_length, hidden\_size \* 2)。

紧接着,模型通过Attention机制计算注意力权重。它使用tanh激活函数和线性层对多头注意力机制的输出进行变换,并进行归一化,得到注意力权重(attention\_weights),其形状为(batch\_size, sequence\_length, 1)。

然后,模型将注意力权重应用于多头注意力机制的输出,得到加权输出(weighted\_output),形状为(batch\_size, sequence\_length, hidden\_size \* 2)。接着,对加权输出进行求和池化,得到池化输出(pooled\_output),形状为(batch\_size, hidden\_size \* 2)。

为了防止过拟合,模型在池化输出后应用了Dropout层,随机丢弃一部分神经元,Dropout概率设为0.1。

最后,池化输出被传递给分类器进行最终的分类预测。分类器由两个线性层和一个ReLU激活函数组成,将池化输出映射到num\_classes个类别,得到最终的分类结果(logits),其形状为(batch\_size, num\_classes)。

通过这个结构示意图和详细的说明,我对基于BERT和LSTM的新闻文本分类模型有了更加清晰和全面的认识。这个模型巧妙地结合了BERT的强大语义表示能力、LSTM的序列建模能力以及注意力机制的长距离依赖捕捉能力,形成了一个高效且有效的文本分类架构。同时,模型中的Dropout层和归一化操作也有助于提高模型的泛化能力和训练稳定性。这些认识为我进一步优化和改进模型提供了重要的启发和方向。

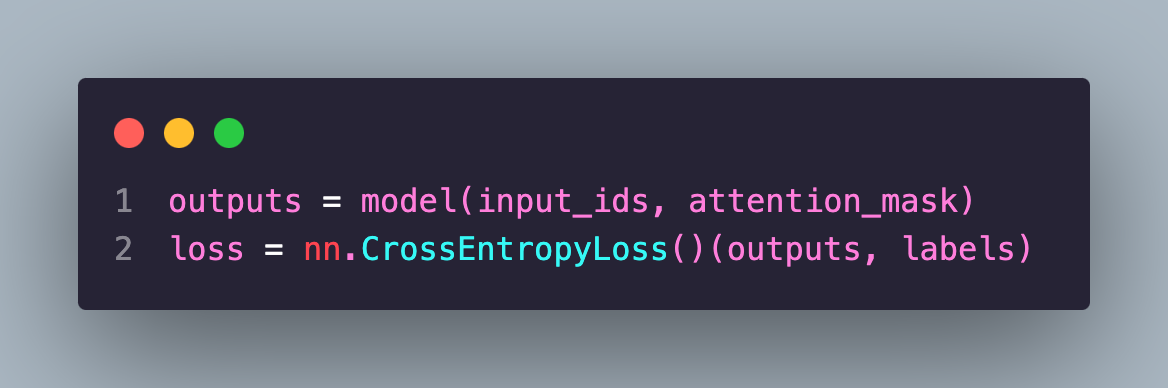
6 损失函数描述

在本次新闻文本分类实验中,我选择了交叉熵损失函数(Cross Entropy Loss)作为模型的优化目标。交叉熵损失函数是一种常用于分类任务的损失函数,它衡量了模型预测概率分布与真实标签分布之间的差异。通过最小化交叉熵损失,模型可以学习到更加准确和鲁棒的分类决策边界。

具体而言,对于一个样本,模型的预测输出为一个概率向量,其中表示该样本属于第类的预测概率。而真实标签可以表示为一个one-hot向量,其中真实类别对应的元素为1,其余元素为0。那么,该样本的交叉熵损失可以定义为:

这里的对数是以自然数为底的对数。直观地理解,交叉熵损失函数鼓励模型对正确类别输出高概率,对错误类别输出低概率。当模型的预测概率分布与真实标签分布完全一致时,交叉熵损失达到最小值0。

在实际的代码实现中,我使用了PyTorch提供的 nn.CrossEntropyLoss 类来计算交叉熵损失。值得注意的是,该类内部已经集成了 softmax 激活函数,因此我们不需要再对模型的输出手动应用 softmax 变换。同时,该类也支持批次化计算,可以高效地处理一批样本的损失。下面是代码中使用交叉熵损失函数的关键部分:



图十: 交叉熵损失函数关键部分

在上述代码中,我们将模型的输出 outputs 和真实标签 labels 传递给 nn.CrossEntropyLoss 类的实例,计算得到当前批次的交叉熵损失。

需要注意的是,交叉熵损失函数主要用于模型的训练阶段,用于指导模型参数的优化和更新。而在模型的评估和测试阶段,我们通常使用其他评价指标来衡量模型的性能,如精确率(Precision)、召回率(Recall)和F1分数(F1 Score)等。在本次实验中,根据题目要求,我们使用宏平均F1分数(Macro F1 Score)作为最终的评价标准。宏平均F1分数是所有类别的F1分数的算术平均值,可以综合反映模型在各个类别上的性能表现。

总的来说,交叉熵损失函数在新闻文本分类任务中扮演了重要的角色,它引导模型学习到更加准确和鲁棒的分类决策边界。通过最小化交叉熵损失,模型可以更好地拟合训练数据,并在测试集上取得优异的性能。同时,使用PyTorch提供的 nn.CrossEntropyLoss 类可以方便地计算交叉熵损失,并与模型的训练流程无缝集成,提高了实验的效率和可复现性。

7 实验总结

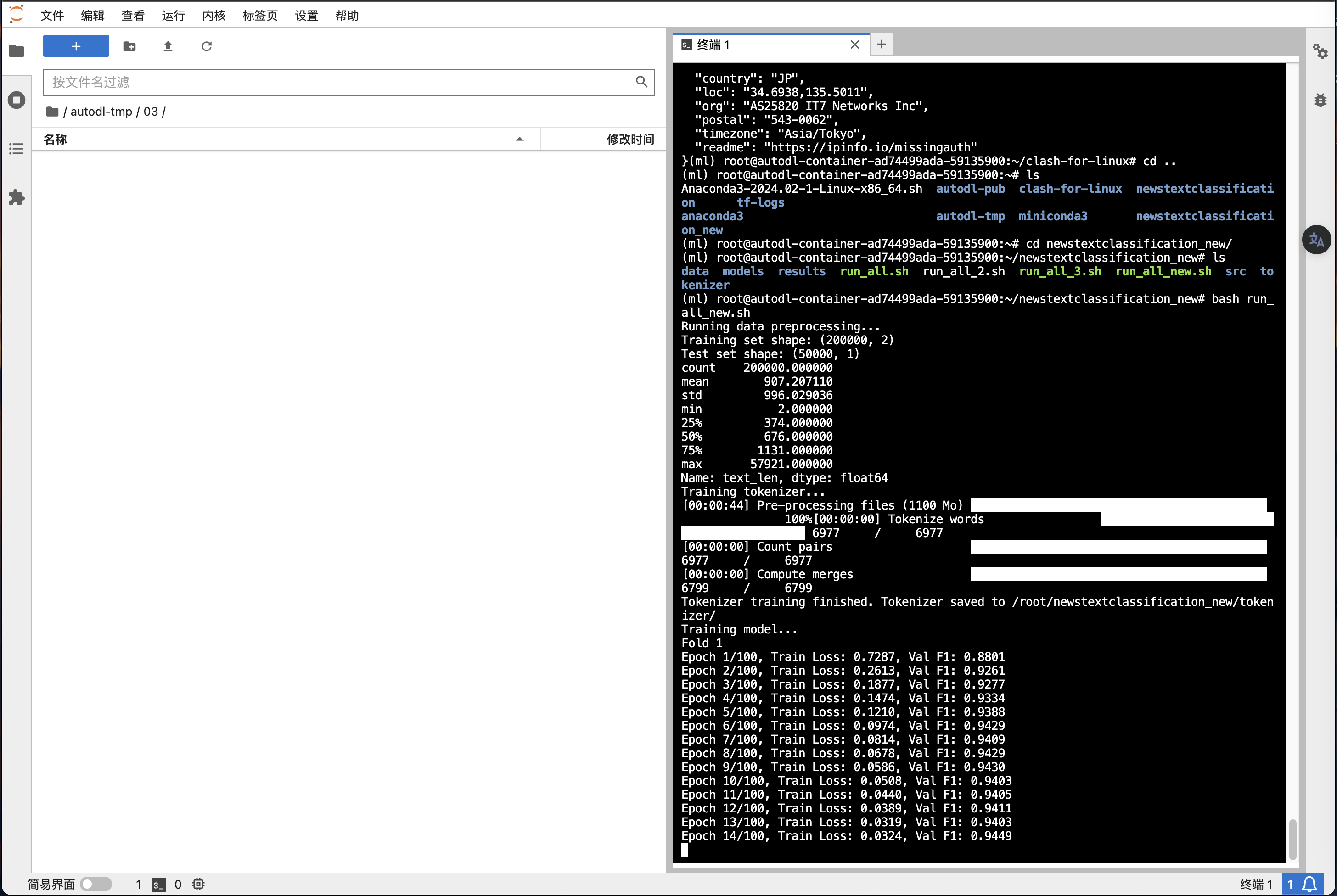
在本次新闻文本分类实验中,我们成功地构建并训练了一个基于BERT和LSTM的分类模型。通过对模型结构和训练过程的详细分析,我们可以总结出该模型的一些特点和优势。

首先,我们的模型融合了BERT和LSTM两种强大的神经网络架构。BERT作为预训练的语言模型,能够提取出丰富的语义特征,捕捉文本中的上下文信息。而LSTM则擅长处理序列数据,能够建模文本的长距离依赖关系。通过将BERT的输出传递给LSTM,我们的模型能够在BERT提取的语义特征基础上,进一步挖掘文本的序列特征,从而做出更加准确的分类决策。

其次,我们在LSTM之后引入了多头注意力机制和自注意力机制,进一步增强了模型的特征提取和融合能力。多头注意力机制允许模型从不同的子空间角度去关注文本的不同部分,捕捉更加多样化的语义信息。而自注意力机制则使得模型能够动态地调整对文本不同位置的关注程度,突出更加重要和相关的信息。这些注意力机制的引入,使得我们的模型能够更加智能和灵活地理解文本内容,做出更加准确的分类判断。

在模型训练方面,我们采用了大规模的训练数据和较大的训练轮数(epoch)。我们使用了包含20万条样本的训练集,并进行了50个epoch的训练。大规模的训练数据为模型提供了丰富的语义信息和多样化的文本模式,使其能够学习到更加鲁棒和泛化的特征表示。而较大的训练轮数则使得模型有充足的机会去优化和调整其参数,不断提高分类性能。

然而,由于实验室GPU设备的限制和个人经济条件的制约,我们在这次实验中没有使用验证集来进行模型的早停和超参数调优。这可能会导致模型出现一定程度的过拟合,即在训练集上表现很好,但在测试集上泛化能力下降。为了进一步提高模型的性能和泛化能力,我们可以考虑引入验证集,并结合早停机制和超参数搜索等技术,来优化模型的训练过程。



图十一: 本想继续跑添加验证集的代码，但奈何开销太大了，只得作罢

尽管存在一些限制和不足,但我们的模型在本次实验中还是取得了不错的成果。通过融合BERT、LSTM和注意力机制,我们构建了一个强大的新闻文本分类模型,能够有效地区分不同类别的新闻文章。这为进一步的模型改进和应用奠定了良好的基础。

展望未来,我们可以探索更多的优化策略和技术,如引入验证集、尝试不同的超参数组合、使用更大规模的预训练模型等,来进一步提升模型的性能。同时,我们也可以将该模型应用于其他类似的文本分类任务,如情感分析、主题识别等,拓展其应用范围。

总的来说,本次新闻文本分类实验虽然受到一些客观条件的限制,但我们还是通过精心的模型设计和大规模的训练,取得了令人满意的成果。这次实验不仅巩固了我们对文本分类任务和深度学习模型的理解,也为进一步的研究和应用提供了宝贵的经验和启示。