廣東工業大學

机器学习课程设计

题目：LCF 方法在基于方面的情感分类的应用

学　　院　计算机学院

专　　业　计算机科学与技术

学　　号　3121005309

学生姓名　孙天一

编　　号　91

2023　年　10　月

LCF 方法在基于方面的情感分类的应用

摘要：LCF 方法是由曾碧卿、杨恒等人于 2019 年提出，应用于基于方面的情感分类，并且发表于当年的 Applied Sciences 中。在本次课程设计中，本人将该方法进行了复现，从而实现了预测句子或文档中不同方面的情感极性。这个机制包括 LCF 设计，其中使用 Context Features Dynamic Mask (CDM)来选择保留或减弱与目标方面相关性不高的上下文特征，从而减少这些无关特征对情感分类的影响。最后，该方法引入了 BERT-shared 层，从而捕捉局部上下文和全局上下文的内部长期依赖关系。
关键词：情感分类　自注意力　预训练模型

一、 设计简介

基于方面的情感分类，英文名称 Aspect-based sentiment classification，下文中简称 ABSC。该分类任务关注的是在文本中对不同方面或实体的情感进行分类, 具体而言，其目标是指在给定的文本中，针对文本中的不同方面或实体，预测其情感极性。LCF 方法便是根据目标方面的局部上下文和全局上下文，从而进行细粒度的情感分类。

二、 配置环境

该项目的训练过程采用设备为 AutoDL 云端服务器，操作系统 ubuntu20.04，CPU 为 Intel(R) Xeon(R) Gold 5320，GPU 为 RTX A4000，显存大小为 16G；该项目的测试所采用的设备为本地计算机，操作系统 Windows11 操作系统，版本号 22H2，CPU 为 AMD Ryzen 7 5800H，GPU 为 Nvidia GTX1650，显存大小为 4G。

对于运行环境的配置，在两个过程中均保持了一致，即 Python 版本号为 3.8、torch 版本为 1.13、transformers 版本号为 4.35.2、numpy 版本为 1.24、sklearn 版本号为 0.0.post11。

三、 具体构造

在原论文中，LCF 方法分为了两个变种，他们分别为基于 GloVe 工具以及基于 BERT 工具来进行设计的。该方法的设计主要分为了以下的几层：文本预处理、嵌入层、预特征提取层、特征提取层、特征交互学习层以及输出层。两

个变种的主要区别在于嵌入层和预特征提取层。在本次课程设计中，本人仅对基于 BERT 工具的 LCF 设计进行了复现。下图展示了基于 BERT 工具的 LCF 设计总体架构。

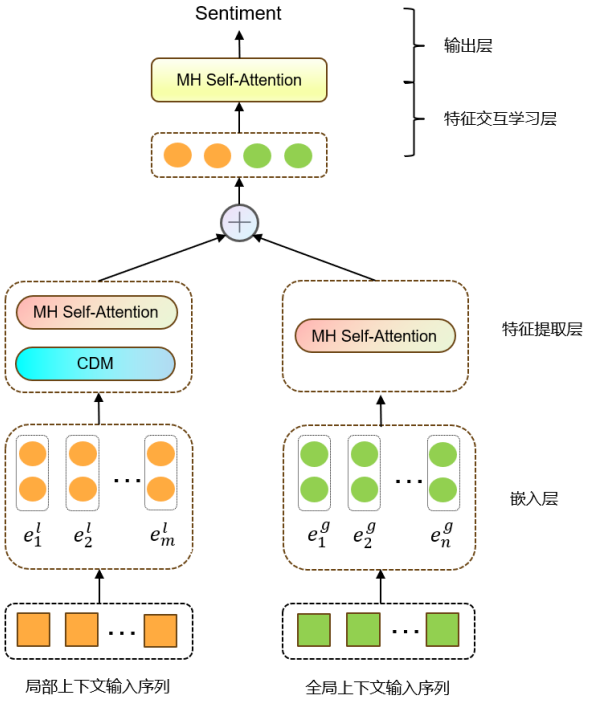

图 1 基于 BERT 工具的 LCF 设计总体架构

3.1 文本预处理

在该层中，我们构建了一个用于将文本转换为序列的工具类，并且命名为“Tokenizer”。因为你文本数据需要在输入神经网络之前进行预处理，因此该类的作用即为将文本中的单词映射为数字索引。首先，在该类会遍历给定的文本，将每个单词添加到词汇表中，并为每个单词分配一个唯一的整数索引。随后，该类会按单词拆分，将每个单词映射到词汇表中的整数索引。未在词汇表中找到的单词将被映射为一个特殊的未知单词索引。最后，该方法会对序列进行填充或截断，以确保其长度不超过指定的最大长度。

当预处理完成后，我们可以得到一个含有四个张量的 list 列表，这四个张量分别为由 BERT 词汇表中的索引组成，并且用于表示全局上下文的输入序列的 text_bert_indices、表示全局上下文的分段标识，用于区分上下文和方面的 bert_segments_ids、由 BERT 词汇表中的索引组成，用于表示局部上下文的输入序列的 text_local_indices 以及用于表示方面的输入序列的 aspect_indices。

当处理完成后，该列表将会作为训练数据而被送入模型中，用于模型的训练。

3.2 嵌入层

在本次课程设计所实现的 LCF-BERT 方法中，我们将使用的为预训练的 BERT 共享层来作为嵌入层。BERT 是一种基于 Transformer 架构的深度学习模型，被广泛应用在自然语言处理任务中。在该方法中，我们采用了用了两个独立的 BERT-shared 层，分别用于建模局部上下文序列特征和全局上下文特征。该层可以将输入序列中的每个单词和标记映射到一个向量空间，从而以便能够在机器学习模型中进行处理。

3.3 预特征提取层

在 LCF-GloVe 方法中，作者采用了该层用于用于提高 LCF 设计的语义特征学习能力。但是对于 LCF-BERT 方法，作者认为 BERT-shared 层足够强大，能够充分捕捉上下文特征。因为，在本次课程设计中，我们也直接省略了这一层，选择直接使用 BERT 共享层的输出。

3.4 特征提取层

该层级被用于学习局部上下文特征和全局上下文特征。局部上下文特征提取层包含一个局部上下文关注层，该上下文关注层命名为上下文特征动态掩码（Dynamic Mask for Context Features，下文简称 CDM）层，用于根据语义相关距离（Semantic-Relative Distance,下文简称 SRD）来保留或减弱与目标方面相关性不高的上下文特征。全局上下文特征提取层只包含一个 MHSA（全称. Multi-Head Self-Attention）层，用于捕获整个输入序列的特征。

3.4.1 语义相关距离（Semantic-Relative Distance）

语义相关距离（下文简称 SRD）是一个用来评估上下文词和方面之间的依赖关系的指标。SRD 的计算方法是统计每个上下文词和方面之间的词距离，然后减去方面的长度的一半。计算公式如下：

$$SRD_i = |i - P_a| - \left\lfloor \frac{m}{2} \right\rfloor$$

其中，i 和 P_a 分别是上下文单词的位置和方面的中心位置。m 是方面的序列长度。SRD_i 表示第 i 个上下文令牌与特定方面之间的 *SRD*。*SRD* 的值越小，则表示上下文词和方面的语义相关性越强，反之则越弱。*SRD* 的作用是帮助 LCF 设计模型识别出方面的局部上下文，即与方面有更密切的情感联系的上下文词。

3.4.2 上下文特征动态掩码层（Dynamic Mask for Context Feature）

已知在计算来自嵌入层的所有标记的输出之后，*SRD* 阈值以上每个输出位置的输出特征将被掩盖或减弱，而局部上下文单词的输出特征将被完全保留。LCF 设计的输入序列主要基于全局上下文。但是 LCF 设计模型为通过采用局部上下文关注层，来去聚焦于局部上下文。在本次课程设计中，我们便采用了 CDM 层来实现对上下文的局部关注。

CDM 层具体过程如下：

(1). 首先，计算每个上下文单词与目标方面之间的语义相关距离（SRD）；

(2). 然后，根据一个预设的 SRD 阈值，构建一个掩码矩阵，对于每个上下文单词，如果其 SRD 大于阈值，就将其对应的掩码向量设为全零向量，否则设为全一向量；

(3). 最后，将掩码矩阵与上下文特征矩阵进行元素级别的乘法，从而保留与目标方面相关性较高的上下文特征，而屏蔽掉相关性较低的上下文特征。

该层可以在自注意力机制的基础上，增强对局部上下文的关注，从而提高方面级情感分类的性能。

3.4.3 多头自注意力（Multi-Head Self-Attention）

多头自注意力是一种用于自注意力机制的改进方法。它通过将原始的单头自注意力机制扩展为多头自注意力机制，从而提高了模型对输入序列的建模能力。具体来说，它将输入序列分别投影到多个不同的子空间中，然后在每个子空间中分别计算自注意力，最后将多个子空间的自注意力结果拼接起来，得到

最终的自注意力表示。这样做的好处是，每个子空间可以关注输入序列的不同方面，从而提高了模型对输入序列的表征能力。计算过程如下：

$$\begin{cases} \text{Attention}(Q,K,V)=\text{softmax}(\dfrac{QK^T}{\sqrt{d_k}})\text{V} \\ Q=X\cdot W^q, K=X\cdot W^k, V=X\cdot W^v \end{cases}$$

其中，Q 为 *Query* 矩阵，用于计算每个位置的注意力得分；K 为 *Key* 矩阵，用于表示每个位置的特征；V 为 *Value* 矩阵，用于表示每个位置的值。W^q 、W^k、W^v 为可训练的权重矩阵。[1]当完成自注意力的计算后，我们会通过一个 *Tanh* 激活函数来对自注意力结果进行了处理，得到多头自注意力层的最终输出。下图展示了该层的具体过程。

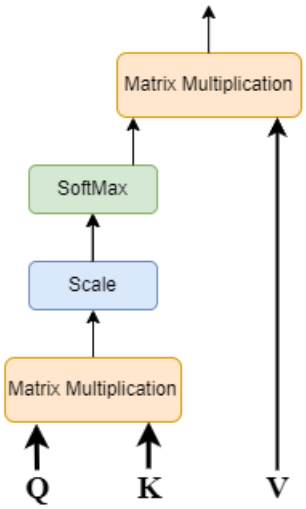

图 2 多头自注意力计算过程

3.5 特征交互学习层

在该层中，我们会先将局部上下文特征和方面特征拼接起来，得到一个维度为 $2d$ 的特征向量；然后，通过一个全连接层将特征向量映射到一个维度为 d 的向量；最后，再使用一个自注意力机制，从而对映射后的特征向量进行加权求和，得到最终的局部上下文表示。

3.6 输出层

在该层中，我会先对特征交互学习层得到的输出通过一个池化层进行池化，得到一个维度为 H 的向量；随后，我们将池化后的向量通过一个全连接层进行线性变换，得到一个维度为 C 的向量。C 的大小等于我们所设定的情绪的种类的个数。本次课程设计中 C 等于 3。最后，我们将线性变换后的向量通过一个激活函数（此处为 softmax 函数）来进行非线性变换，得到最终的分类结果。

四、 模型评估

在训练过程中，我们所采用的数据集为 SemEval-2014 中的餐馆数据集[2]。该数据集包含了来自不同来源的餐馆评论，其中包括用户对餐馆的评价和观点。该数据集中共含有 3608 条训练样本以及 1120 条测试样本。其中训练样本中 2164 条为积极，807 条为消极，637 条为中性；测试样本中，共有 728 条为积极，196 条为消极，196 条为中性。该数据集的所有方面都标记为三类情感极性：积极的、中性的和消极的。在实验中，这些数据集是原始的，没有经过重构，没有删除任何冲突的标签。

在训练的过程中，我们设置了学习率为1×10^{-3}，丢弃率为 0，L2 正则化设置为1×10^{-5}，每轮会送入 16 个样本用于预测或测试。最后，我们采用准确度和 Macro-F1 分数来评估 LCF 设计的性能。*Macro-F1* 的计算公式如下：

$$\begin{cases} F1=\dfrac{2\times Precision\times Recall}{recision+Recall} \\ Macro\text{-}F1=\dfrac{\sum_{i=1}^{C}F1_i}{C} \end{cases}$$

其中，*Precision* 表示被模型预测为正类别的样本中有多少是真正的正类别；*Recall* 为召回率，表示所有真正的正类别样本中有多少被模型成功预测为正类别；C 为类别的总数[3]。

当两个得分结果均优于上一轮的时候，我们便会将当前模型进行保存。且如果当前轮数减去最大验证准确率的轮数大于等于我们所设定的耐心值，那么

则表明验证准确率不再会有提升，可以提前终止训练，从而避免出现过拟合。在 10 轮的训练过后，我们得到了最后的最佳模型，该模型的正确率为 86.93%，且 *Macro-F1* 得分为 79.19%。

五、 训练中遇到的问题以及解决方案

1. 预训练 bert 模型无法下载且加载：

在源代码中，原作者采用了 BertTokenizer.from_pretrained 这一方法来对预训练模型进行加载。若预训练模型在本地中不存在，则会自动在官网中对所需的文件进行下载。但由于源文件的服务器位于国外，无法进行直接连接。因此，我选择了在国内的镜像网站中找到对应文件，并且根据报错提示，在项目中新建了文件夹命名为 bert-base-uncased，随后将下载的文件复制到该文件夹中。

当下载完成后，确保下载的文件不存在损坏。但是在对预训练模型进行加载的时候依然存在报错无法加载。此时，本人根据官方网站中”How to use from the transformers”中得到提示，将原文中的 self.tokenizer = BertTokenizer.from_pretrained(pretrained_bert_name)修改为了更为通用的 self.tokenizer = AutoTokenizer.from_pretrained(pretrained_bert_name)来去加载，从而解决了该问题。

2. 预训练 bert 模型输出结果为”hidden_layer”字符串而非 tensor；

在正常情况下，若 bert 模型输入为 tensor，则输出一定为多个 tensor 类型。但是在对模型进行加载测试的时候，出现了 bert 输出字符串 last_hidden_state 的情况。经过查询可得，BERT 模型的输出通常包括 last_hidden_state，其为最后一层的隐藏状态、pooler_output，为池化层的输出、hidden_states，为所有层的隐藏状态、attentions，为注意力权重四个部分。在本次实验中，我们仅需其 last_hidden_state 部分，从而进行后续的操作。因此，我们将原文中的

```
bert_spc_out, _ = self.bert_spc(text_bert_indices, oken_type_ids=bert_segments_ids)
```

修改为了下面的代码

```
bert_spc_out = self.bert_spc(text_bert_indices, ken_type_ids=bert_segments_ids)
bert_spc_out = bert_spc_out[0]
```

在修改后，该问题得到了解决。

六、 总结

在此次课程设计中，我们有以下几点的收获：

(1). 我学会了如何去阅读一篇学术论文，并且成功将其复现。这助于培养实际问题解决和实验设计的能力，并且加深对该领域的理解和知识；

(2). 我对一个模型的建立、训练以及评估有了更深的理解。这能让我更为深入地理解机器学习模型的内部原理和工作机制；

(3). 我在配置环境以及训练的过程中也遇到了很多的问题，如 arrey、tensor 以及 list 三中数据类型的转化。在本人的课程设计中，模型仅接受 tensor 类型的数据，因此，我们需要将其他的数据类型进行合理的转化。

当然，在本次课程设计中，本人认为也有一下几点可以改进的地方：

(1). 过拟合的改进：当出现过拟合的时候，我选择了早停的方法进行解决，并且将最有的一次模型进行保存。对于正交化、剪枝等可能可以进一步提升模型准确率的方法并没有采用；

(2). 缺少自己的改进：没有引入自己的改进总而可能导致模型缺乏创新性。因此，该模型可能只能用于解决特定问题或任务，不能在实际应用中直接应用。

七、 参考文献

[1] Voita E, Talbot D, Moiseev F, et al. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned[J]. arXiv preprint arXiv:1905.09418, 2019.

[2] Kirange D K, Deshmukh R R, Kirange M D K. Aspect based sentiment analysis semeval-2014 task 4[J]. Asian Journal of Computer Science and Information Technology (AJCSIT) Vol, 2014, 4.

[3] Opitz J, Burst S. Macro f1 and macro f1[J]. arXiv preprint arXiv:1911.03347, 2019.

八、 附录

```
# lcf_bert.py framework LCF 整体框架
import torch
import torch.nn as nn
import copy
import numpy as np

from transformers.models.bert.modeling_bert import BertPooler, BertSelfAttention
```

```
class SelfAttention(nn.Module):
    def __init__(self, config, opt):
        super(SelfAttention, self).__init__()
        self.opt = opt
        self.config = config
        self.SA = BertSelfAttention(config)
        self.tanh = torch.nn.Tanh()

    def forward(self, inputs):
        zero_tensor = torch.tensor(
            np.zeros(
                (inputs.size(0),
                 1,
                 1,
                 self.opt.max_seq_len),
                dtype=np.float32),
            dtype=torch.float32).to(
            self.opt.device)
        SA_out = self.SA(inputs, zero_tensor)
        return self.tanh(SA_out[0])


class LCF_BERT(nn.Module):
    def __init__(self, bert, opt):
        super(LCF_BERT, self).__init__()

        self.bert_spc = bert
        self.opt = opt
        self.bert_local = bert    # Default to use single Bert and reduce memory requirements
        self.dropout = nn.Dropout(opt.dropout)
        self.bert_SA = SelfAttention(bert.config, opt)
        self.linear_double = nn.Linear(opt.bert_dim * 2, opt.bert_dim)
        self.linear_single = nn.Linear(opt.bert_dim, opt.bert_dim)
        self.bert_pooler = BertPooler(bert.config)
        self.dense = nn.Linear(opt.bert_dim, opt.polarities_dim)

    def feature_dynamic_mask(self, text_local_indices, aspect_indices):
        texts = text_local_indices.cpu().numpy()
        asps = aspect_indices.cpu().numpy()
        mask_len = self.opt.SRD
        masked_text_raw_indices = np.ones(
            (text_local_indices.size(0),
             self.opt.max_seq_len,
             self.opt.bert_dim),
            dtype=np.float32)
        for text_i, asp_i in zip(range(len(texts)), range(len(asps))):
            asp_len = np.count_nonzero(asps[asp_i]) - 2
            try:
                asp_begin = np.argwhere(texts[text_i] == asps[asp_i][1])[0][0]
            except BaseException:
                continue
            if asp_begin >= mask_len:
                mask_begin = asp_begin - mask_len
            else:
                mask_begin = 0
            for i in range(mask_begin):
                masked_text_raw_indices[text_i][i] = np.zeros((self.opt.bert_dim),
```

```
dtype=np.float64)
                    for j in range(asp_begin + asp_len + mask_len, self.opt.max_seq_len):
                        masked_text_raw_indices[text_i][j] = np.zeros((self.opt.bert_dim),
dtype=np.float64)
            masked_text_raw_indices = torch.from_numpy(masked_text_raw_indices)
            return masked_text_raw_indices.to(self.opt.device)

        def forward(self, inputs):
            text_bert_indices = inputs[0]
            bert_segments_ids = inputs[1]
            text_local_indices = inputs[2]
            aspect_indices = inputs[3]

            bert_spc_out = self.bert_spc(
                text_bert_indices,
                token_type_ids=bert_segments_ids)
            bert_spc_out = bert_spc_out[0]
            bert_spc_out = self.dropout(bert_spc_out)

            bert_local_out = self.bert_local(text_local_indices)
            bert_local_out = bert_local_out[0]
            bert_local_out = self.dropout(bert_local_out)

            # CDM 层
            masked_local_text_vec = self.feature_dynamic_mask(
                text_local_indices, aspect_indices)
            bert_local_out = torch.mul(bert_local_out, masked_local_text_vec)

            out_cat = torch.cat((bert_local_out, bert_spc_out), dim=-1)
            mean_pool = self.linear_double(out_cat)
            self_attention_out = self.bert_SA(mean_pool)
            pooled_out = self.bert_pooler(self_attention_out)
            dense_out = self.dense(pooled_out)

            return dense_out

# train.py
import logging
import argparse
import math
import os
import sys
import random
import numpy

from sklearn import metrics
from time import strftime, localtime

from transformers import BertModel

import torch
import torch.nn as nn
from torch.utils.data import DataLoader, random_split

from data_utils import Tokenizer4Bert, ABSADataset
from models import LCF_BERT

logger = logging.getLogger()
```

```
logger.setLevel(logging.INFO)
logger.addHandler(logging.StreamHandler(sys.stdout))

class Instructor:
    def __init__(self, opt):
        self.opt = opt

        tokenizer = Tokenizer4Bert(opt.max_seq_len, opt.pretrained_bert_name)
        bert = BertModel.from_pretrained(
            opt.pretrained_bert_name, return_dict=False)
        self.model = opt.model_class(bert, opt).to(opt.device)

        self.trainset = ABSADataset(opt.dataset_file['train'], tokenizer)
        self.testset = ABSADataset(opt.dataset_file['test'], tokenizer)
        assert 0 <= opt.valset_ratio < 1
        if opt.valset_ratio > 0:
            valset_len = int(len(self.trainset) * opt.valset_ratio)
            self.trainset, self.valset = random_split(
                self.trainset, (len(self.trainset) - valset_len, valset_len))
        else:
            self.valset = self.testset

        if opt.device.type == 'cuda':
            logger.info(
                'cuda memory allocated: {}'.format(
                    torch.cuda.memory_allocated(
                        device=opt.device.index)))
         self._print_args()

    def _print_args(self):
        n_trainable_params, n_nontrainable_params = 0, 0
        for p in self.model.parameters():
            n_params = torch.prod(torch.tensor(p.shape))
            if p.requires_grad:
                n_trainable_params += n_params
            else:
                n_nontrainable_params += n_params
        logger.info(
            '> n_trainable_params: {0}, n_nontrainable_params: {1}'.format(
                n_trainable_params,
                n_nontrainable_params))
        logger.info('> training arguments:')
        for arg in vars(self.opt):
            logger.info('>>> {0}: {1}'.format(arg, getattr(self.opt, arg)))

    def _reset_params(self):
        for child in self.model.children():
            if not isinstance(child, BertModel):    # skip bert params
                for p in child.parameters():
                    if p.requires_grad:
                        if len(p.shape) > 1:
                            self.opt.initializer(p)
                        else:
                            stdv = 1. / math.sqrt(p.shape[0])
                            torch.nn.init.uniform_(p, a=-stdv, b=stdv)

    def _train(self, criterion, optimizer, train_data_loader, val_data_loader):
        max_val_acc = 0
```

```
        max_val_f1 = 0
        max_val_epoch = 0
        global_step = 0
        path = None
        for i_epoch in range(self.opt.num_epoch):
            logger.info('>' * 100)
            logger.info('epoch: {}'.format(i_epoch))
            n_correct, n_total, loss_total = 0, 0, 0
            # switch model to training mode
            self.model.train()
            for i_batch, batch in enumerate(train_data_loader):
                global_step += 1
                # clear gradient accumulators
                optimizer.zero_grad()

                inputs = [batch[col].to(self.opt.device)
                          for col in self.opt.inputs_cols]
                outputs = self.model(inputs)
                targets = batch['polarity'].to(self.opt.device)

                loss = criterion(outputs, targets)
                loss.backward()
                optimizer.step()

                n_correct += (torch.argmax(outputs, -1)
                              == targets).sum().item()
                n_total += len(outputs)
                loss_total += loss.item() * len(outputs)
                if global_step % self.opt.log_step == 0:
                    train_acc = n_correct / n_total
                    train_loss = loss_total / n_total
                    logger.info(
                        'loss: {:.4f}, acc: {:.4f}'.format(
                            train_loss, train_acc))

            val_acc, val_f1 = self._evaluate_acc_f1(val_data_loader)
            logger.info(
                '> val_acc: {:.4f}, val_f1: {:.4f}'.format(
                    val_acc, val_f1))
            if val_acc > max_val_acc:
                max_val_acc = val_acc
                max_val_epoch = i_epoch
                if not os.path.exists('state_dict'):
                    os.mkdir('state_dict')
                path = 'state_dict/{0}_{1}_val_acc_{2}'.format(
                    self.opt.model_name, self.opt.dataset, round(val_acc, 4))
                torch.save(self.model.state_dict(), path)
                logger.info('>> saved: {}'.format(path))
            if val_f1 > max_val_f1:
                max_val_f1 = val_f1
            if i_epoch - max_val_epoch >= self.opt.patience:
                print('>> early stop.')
                break

        return path

    def _evaluate_acc_f1(self, data_loader):
        n_correct, n_total = 0, 0
```

```
        t_targets_all, t_outputs_all = None, None
        # switch model to evaluation mode
        self.model.eval()
        with torch.no_grad():
            for i_batch, t_batch in enumerate(data_loader):
                t_inputs = [t_batch[col].to(self.opt.device)
                            for col in self.opt.inputs_cols]
                t_targets = t_batch['polarity'].to(self.opt.device)
                t_outputs = self.model(t_inputs)
                n_correct += (torch.argmax(t_outputs, -1)
                              == t_targets).sum().item()
                n_total += len(t_outputs)

                if t_targets_all is None:
                    t_targets_all = t_targets
                    t_outputs_all = t_outputs
                else:
                    t_targets_all = torch.cat(
                        (t_targets_all, t_targets), dim=0)
                    t_outputs_all = torch.cat(
                        (t_outputs_all, t_outputs), dim=0)

        acc = n_correct / n_total
        f1 = metrics.f1_score(t_targets_all.cpu(), torch.argmax(
            t_outputs_all, -1).cpu(), labels=[0, 1, 2], average='macro')
        return acc, f1

    def run(self):
        # Loss and Optimizer
        criterion = nn.CrossEntropyLoss()
        _params = filter(lambda p: p.requires_grad, self.model.parameters())
        optimizer = self.opt.optimizer(
            _params, lr=self.opt.lr, weight_decay=self.opt.l2reg)

        train_data_loader = DataLoader(
            dataset=self.trainset,
            batch_size=self.opt.batch_size,
            shuffle=True)
        test_data_loader = DataLoader(
            dataset=self.testset,
            batch_size=self.opt.batch_size,
            shuffle=False)
        val_data_loader = DataLoader(
            dataset=self.valset,
            batch_size=self.opt.batch_size,
            shuffle=False)

        self._reset_params()
        best_model_path = self._train(
            criterion,
            optimizer,
            train_data_loader,
            val_data_loader)
        self.model.load_state_dict(torch.load(best_model_path))
        test_acc, test_f1 = self._evaluate_acc_f1(test_data_loader)
        logger.info(
            '>> test_acc: {:.4f}, test_f1: {:.4f}'.format(
                test_acc, test_f1))
```

```
def main():
    # Hyper Parameters
    parser = argparse.ArgumentParser()
    parser.add_argument('--model_name', default='lcf_bert', type=str)
    parser.add_argument(
        '--dataset',
        default='restaurant',
        type=str,
        help='twitter, restaurant, laptop')
    parser.add_argument('--optimizer', default='adam', type=str)
    parser.add_argument('--initializer', default='xavier_uniform_', type=str)
    parser.add_argument(
        '--lr',
        default=2e-5,
        type=float,
        help='try 5e-5, 2e-5 for BERT, 1e-3 for others')
    parser.add_argument('--dropout', default=0.1, type=float)
    parser.add_argument('--l2reg', default=0.01, type=float)
    parser.add_argument(
        '--num_epoch',
        default=20,
        type=int,
        help='try larger number for non-BERT models')
    parser.add_argument(
        '--batch_size',
        default=1,
        type=int,
        help='try 16, 32, 64 for BERT models')
    parser.add_argument('--log_step', default=10, type=int)
    parser.add_argument('--embed_dim', default=300, type=int)
    parser.add_argument('--hidden_dim', default=300, type=int)
    parser.add_argument('--bert_dim', default=768, type=int)
    parser.add_argument(
        '--pretrained_bert_name',
        default='bert-base-uncased',
        type=str)
    parser.add_argument('--max_seq_len', default=85, type=int)
    parser.add_argument('--polarities_dim', default=3, type=int)
    parser.add_argument('--hops', default=3, type=int)
    parser.add_argument('--patience', default=5, type=int)
    parser.add_argument('--device', default=None, type=str, help='e.g. cuda:0')
    parser.add_argument('--seed', default=1234, type=int,
                                help='set seed for reproducibility')
    parser.add_argument(
        '--valset_ratio',
        default=0,
        type=float,
        help='set ratio between 0 and 1 for validation support')
    # The following parameters are only valid for the lcf-bert model
    parser.add_argument(
        '--local_context_focus',
        default='cdm',
        type=str,
        help='local context focus mode, cdw or cdm')
    parser.add_argument(
        '--SRD',
```

```
        default=3,
        type=int,
        help='semantic-relative-distance, see the paper of LCF-BERT model')
    opt = parser.parse_args()

    if opt.seed is not None:
        random.seed(opt.seed)
        numpy.random.seed(opt.seed)
        torch.manual_seed(opt.seed)
        torch.cuda.manual_seed(opt.seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False
        os.environ['PYTHONHASHSEED'] = str(opt.seed)

    model_classes = {
        'lcf_bert': LCF_BERT,
        # default hyper-parameters for LCF-BERT model is as follws:
        # lr: 2e-5
        # l2: 1e-5
        # batch size: 16
        # num epochs: 5
    }
    dataset_files = {
        'restaurant': {
            'train': './datasets/semeval14/Restaurants_Train.xml.seg',
            'test': './datasets/semeval14/Restaurants_Test_Gold.xml.seg'
        },
    }
    input_colses = {
        'lcf_bert': [
            'concat_bert_indices',
            'concat_segments_indices',
            'text_bert_indices',
            'aspect_bert_indices'],
    }
    initializers = {
        'xavier_uniform_': torch.nn.init.xavier_uniform_,
        'xavier_normal_': torch.nn.init.xavier_normal_,
        'orthogonal_': torch.nn.init.orthogonal_,
    }
    optimizers = {
        'adadelta': torch.optim.Adadelta,    # default lr=1.0
        'adagrad': torch.optim.Adagrad,    # default lr=0.01
        'adam': torch.optim.Adam,    # default lr=0.001
        'adamax': torch.optim.Adamax,    # default lr=0.002
        'asgd': torch.optim.ASGD,    # default lr=0.01
        'rmsprop': torch.optim.RMSprop,    # default lr=0.01
        'sgd': torch.optim.SGD,
    }
    opt.model_class = model_classes[opt.model_name]
    opt.dataset_file = dataset_files[opt.dataset]
    opt.inputs_cols = input_colses[opt.model_name]
    opt.initializer = initializers[opt.initializer]
    opt.optimizer = optimizers[opt.optimizer]
    opt.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') \
        if opt.device is None else torch.device(opt.device)

    log_dir = 'logs'
```

```
        os.makedirs(log_dir, exist_ok=True)
        log_file = os.path.join(log_dir,
                                '{}-{}-{}.log'.format(opt.model_name, opt.dataset,
strftime("%y%m%d-%H%M", localtime())))
        logger.addHandler(logging.FileHandler(log_file))

        ins = Instructor(opt)
        ins.run()


if __name__ == '__main__':
        main()
```

机器学习课程设计争优申请表	
设计题目	LCF 方法在基于方面的情感分类的应用
学生姓名	孙天一
报告编号	91
班级和学号	计算机科学与技术（伏羲班）-3121005309
设计内容	1. 文本预处理层：将原始文本数据转换为了模型可以处理的格式。这个过程也包括了分词、去除停用词、词干提取等； 2. 嵌入层：使用了 BERT 共享层，从而实现了将文本数据转换为了向量表示； 3. 特征提取层：使用了上下文特征动态掩码，从而实现了在关注与方面更相关的局部上下文词的同时，也减弱与方面不太相关的上下文词的影响； 4. 特征交互学习层：该层用于得到最终的局部上下文表示； 5. 输出层：根据特征交互学习层的输出来预测方面的情感极性，从而实现了分类。
开发平台和编程语言	开发平台：Pycharm 2022 编程语言与框架：Python 基于 Torch 的框架
创新点或特色	1. 提出了一种基于多头自注意力的局部上下文关注机制（LCF 机制），用于细粒度的方面级情感分类任务； 2. 提出了上下文词之间的语义相关距离（SRD），从而可以动态地选择更有影响力的局部上下文特征； 3. 设计了上下文特征动态掩码（CDM），从而实现了通过掩码矩阵来屏蔽或弱化与方面词较远的上下文词的特征； 4. 结合了预训练的 BERT 模型作为共享层，用于提取局部上下文和全局上下文的特征。