**机器学习与知识发现**

**课程实验报告**

**（剧本角色情感识别）**

一、小组成员

|  |  |  |
| --- | --- | --- |
| 姓名 | 学号 | 完成情况打分 |
| 刘硕 | SA21011026 | 10 |
| 侯懿文 | SA21011054 | 10 |
| 孙浩源 | SA21011078 | 10 |
| 严嘉鹏 | SA21011148 | 10 |
| 张牧野 | SA21011261 | 10 |

二、参赛情况







三、赛题内容

剧本角色情感识别：

赛题提供一部分电影剧本作为训练集，训练集数据已由人工进行标注，参赛队伍需要对剧本场景中每句对白和动作描述中涉及到的每个角色的情感从多个维度进行分析和识别。

输入：训练数据中包含id，content，character，emotions等数据列，其中id通过script\_id，scene\_num,以及sentence\_num组成，emotion是情感识别结果，按顺序分别对应love, joy, fright, anger, fear, sorrow的情感值。一行数据的内容：(id(script\_id+scene\_num+sentence\_num), content, character, emotion)。例如：

|  |  |  |  |
| --- | --- | --- | --- |
| id | content | character | emotion |
| 1171\_0001\_A\_7 | c1开心地点了点头。 | c1 | 0,1,0,0,0,0 |

emotion的各个分量中，love=0, joy=1, fright=0, anger=0, fear=0, sorrow=0；情感值范围是[0, 1, 2, 3]：0-没有，1-弱，2-中，3-强。验证数据的格式和训练数据基本一致，只是去掉了最后一列emotion。

输出：对于每条验证数据，为其标注对应的emotion，格式为每行一条内容 (id,emotion)。

四、运行环境

运行环境：python3;

使用的框架：torch;

其它可能需要的python库：tqdm、pandas、numpy、functools；

运行方法：（示例）python baseline.py --bert\_id 1；不同的bert\_id指定选择不同的tokenizer的ckpt版本，可以参考baseline.py查看id对应的ckpt

五、数据分析和模型设计

数据的读取：按照数据集中的格式，按行读取数据，然后将每行切分为'id','text','love', 'joy', 'fright', 'anger', 'fear', 'sorrow'，然后用Dataloader加载；

接着我们尝试了'hfl/chinese-roberta-wwm-ext-large', 'voidful/albert\_chinese\_ba se', 'bert-base-chinese', 'hfl/chinese-macbert-base', 'nghuyong/ernie-1.0'多种预训练模型在此基础上进行训练；

网络模型设计：我们设计的模型基于bert模型得到的句子的词向量特征，并在此基础上接一个全连接层得到每个单词对应的情感特征概率，并根据得到的每个情感的概率值作为权值向量，生成最终的句子特征向量，再根据这个句子特征向量，针对每一个情感，分别连接一个全连接神经网络，得到每一个情感值的归一化的概率，生成最终的输出向量，即为情感预测的值向量。具体的，我们首先根据选定的tokenizer得到句子文本所对应的input\_ids和attention\_mask，并根据input\_ids和attention\_mask得到对应的bert预训练网络模型。我们选择该预训练模型的hidden\_layyer的最后一层作为其特征表示，并根据这个特征表示来训练attention层的表示，来根据该情感分析任务得到一个句子中对于情感值影响权重最大的注意力权值，从而得到句子的向量表示。attention层先使用一个全连接层，然后采用tanh作为激活函数，再接一个全连接层，最后用softmax分类器分类。

对于网络训练，采用的优化器使用AdamW。用训练集进行若干轮训练后，对验证集进行预测标注。

其它所做的一些优化内容：尝试了多种tokenizer进行分词(roberta/ernie/base)；测试了是否在数据集中导入id条目的模型训练效果；对数据集中的句子进行了预处理，结合character，将句子统一转换为“（角色是+character+）+句子内容”的格式；使用学习率预热，使学习率自适应地调整，在预热期间学习率从0线性增加到优化器中的初始lr，在预热阶段之后创建一个schedule使其学习率从优化器中的初始lr线性降低到0。

六、源码中各模块的功能

原始数据的预处理模块（dataset.py）：输入是原始训练/测试数据，输出的是经过排序、整合的训练数据。在该模块中，先按照序号将原本有些乱序的样本重排，然后给每一条数据加上一句“（角色是:xx）”，以助于训练。

数据转化模块（RoleDataset）：它是一个继承了Dataset的类，将上个模块预处理后的文本调用已有模型包装成Dataset类。在它被调用之前会先在./data 下生成被转化为csv格式的训练数据。重写了len方法和getitem索引方法，并使用一个已有的分词器（本实验是BertTokenizer）对某一行文本进行编码（encode\_plus），然后取出其“input\_ids”和“attention\_mask”作为样本的编码信息。

1. **class** RoleDataset(Dataset):
2. def \_\_init\_\_(self, tokenizer, max\_len, mode='train'):
3. super(RoleDataset, self).\_\_init\_\_()
4. **if** mode == 'train':
5. self.data = pd.read\_csv('data/train.csv',sep='\t', quoting=csv.QUOTE\_NONE)
6. **else**:
7. self.data = pd.read\_csv('data/test.csv',sep='\t', quoting=csv.QUOTE\_NONE)
9. self.texts=self.data['text'].tolist()
10. self.labels=self.data[target\_cols].to\_dict('records')
11. self.tokenizer = tokenizer
12. self.max\_len = max\_len
14. def \_\_getitem\_\_(self, index):
15. text=str(self.texts[index])
16. label=self.labels[index]
18. encoding=self.tokenizer.encode\_plus(text,
19. add\_special\_tokens=True,
20. max\_length=self.max\_len,
21. return\_token\_type\_ids=True,
22. pad\_to\_max\_length=True,
23. return\_attention\_mask=True,
24. return\_tensors='pt',)
26. sample = {
27. 'texts': text,
28. 'input\_ids': encoding['input\_ids'].flatten(),
29. 'attention\_mask': encoding['attention\_mask'].flatten()
30. }
32. **for** label\_col in target\_cols:
33. sample[label\_col] = torch.tensor(label[label\_col]/3.0, dtype=torch.**float**)
34. **return** sample
36. def \_\_len\_\_(self):
37. **return** len(self.texts)

Dataloader模块，将Dataset数据处理成Dataloader类，训练过程中在每个epoch开始的时候对数据进行打乱。

1. def create\_dataloader(dataset, batch\_size, mode='train'):
2. shuffle = True **if** mode == 'train' **else** False
4. **if** mode == 'train':
5. data\_loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=shuffle)
6. **else**:
7. data\_loader = DataLoader(dataset, batch\_size=batch\_size, shuffle=shuffle)
8. **return** data\_loader

网络定义模块（IQIYModelLite）：它是一个继承了nn.Module的类，定义了本模型的网络结构。网络以“input\_ids”和“attention\_mask”作为输入，以每个情绪的估计值作为输出，由于有6个情绪，所以输出是6维。每次forward时，先用attention 网络生成一个权重，然后用torch.sum得到一个文本向量context\_vector，文本向量被分别输入6个全连接层，以得出6个情绪的估计。

1. **class** IQIYModelLite(nn.Module):
2. def \_\_init\_\_(self, n\_classes, model\_name):
3. super(IQIYModelLite, self).\_\_init\_\_()
4. config = AutoConfig.from\_pretrained(model\_name)
5. config.update({"output\_hidden\_states": True,
6. "hidden\_dropout\_prob": 0.0,
7. "layer\_norm\_eps": 1e-7})
9. self.base = BertModel.from\_pretrained(model\_name, config=config)
11. dim = 1024 **if** 'large' in model\_name **else** 768
13. self.attention = nn.Sequential(
14. nn.Linear(dim, 512),
15. nn.Tanh(),
16. nn.Linear(512, 1),
17. nn.Softmax(dim=1)
18. )
19. # self.attention = AttentionHead(h\_size=dim, hidden\_dim=512, w\_drop=0.0, v\_drop=0.0)
21. self.out\_love = nn.Sequential(
22. nn.Linear(dim, n\_classes)
23. )
24. self.out\_joy = nn.Sequential(
25. nn.Linear(dim, n\_classes)
26. )
27. self.out\_fright = nn.Sequential(
28. nn.Linear(dim, n\_classes)
29. )
30. self.out\_anger = nn.Sequential(
31. nn.Linear(dim, n\_classes)
32. )
33. self.out\_fear = nn.Sequential(
34. nn.Linear(dim, n\_classes)
35. )
36. self.out\_sorrow = nn.Sequential(
37. nn.Linear(dim, n\_classes)
38. )
40. init\_params([self.out\_love, self.out\_joy, self.out\_fright, self.out\_anger,
41. self.out\_fear,  self.out\_sorrow, self.attention])
43. def forward(self, input\_ids, attention\_mask):
44. roberta\_output = self.base(input\_ids=input\_ids,
45. attention\_mask=attention\_mask)
47. last\_layer\_hidden\_states = roberta\_output.hidden\_states[-1]
48. weights = self.attention(last\_layer\_hidden\_states)
49. # print(weights.size())
50. context\_vector = torch.sum(weights\*last\_layer\_hidden\_states, dim=1)
51. # context\_vector = weights
53. love = self.out\_love(context\_vector)
54. joy = self.out\_joy(context\_vector)
55. fright = self.out\_fright(context\_vector)
56. anger = self.out\_anger(context\_vector)
57. fear = self.out\_fear(context\_vector)
58. sorrow = self.out\_sorrow(context\_vector)
60. **return** {
61. 'love': love, 'joy': joy, 'fright': fright,
62. 'anger': anger, 'fear': fear, 'sorrow': sorrow,
63. }

训练模块（do\_train）：对于每个样本，使用nn.BCEWithLogitsLoss()得到loss，使用AdamW作为优化器进行梯度更新，训练后的参数保存在文件里。

1. def do\_train(model, date\_loader, criterion, optimizer, scheduler, metric=None):
2. model.train()
3. global\_step = 0
4. tic\_train = time.time()
5. log\_steps = 100
6. **for** epoch in range(EPOCHS):
7. losses = []
8. **for** step, sample in enumerate(train\_loader):
9. input\_ids = sample["input\_ids"].cuda()
10. attention\_mask = sample["attention\_mask"].cuda()
12. outputs = model(input\_ids=input\_ids, attention\_mask=attention\_mask)
14. loss\_love = criterion(outputs['love'], sample['love'].view(-1, 1).cuda())
15. loss\_joy = criterion(outputs['joy'], sample['joy'].view(-1, 1).cuda())
16. loss\_fright = criterion(outputs['fright'], sample['fright'].view(-1, 1).cuda())
17. loss\_anger = criterion(outputs['anger'], sample['anger'].view(-1, 1).cuda())
18. loss\_fear = criterion(outputs['fear'], sample['fear'].view(-1, 1).cuda())
19. loss\_sorrow = criterion(outputs['sorrow'], sample['sorrow'].view(-1, 1).cuda())
20. loss = loss\_love + loss\_joy + loss\_fright + loss\_anger + loss\_fear + loss\_sorrow
22. losses.append(loss.item())
24. loss.backward()
26. #             nn.utils.clip\_grad\_norm\_(model.parameters(), max\_norm=1.0)
27. optimizer.step()
28. scheduler.step()
29. optimizer.zero\_grad()
31. global\_step += 1
33. **if** global\_step % log\_steps == 0:
34. print("global step %d, epoch: %d, batch: %d, loss: %.5f, speed: %.2f step/s, lr: %.10f"
35. % (global\_step, epoch, step, np.mean(losses), global\_step / (time.time() - tic\_train),
36. **float**(scheduler.get\_last\_lr()[0])))
37. state = {
38. 'state': model.state\_dict(),
39. 'epoch': epoch
40. }
41. torch.save(state, './ckpt/iqiyi\_model\_{}.ckpt'.format(str(epoch)))
43. **if** args.no\_train:
44. checkpoint = torch.load('./ckpt/iqiyi\_model\_2.ckpt')
45. model.load\_state\_dict(checkpoint['state'])
46. **else**:
47. do\_train(model, train\_loader, criterion, optimizer, scheduler)

预测模块（predict）：对于每个测试样例，调用训练后的网络预测其情绪。

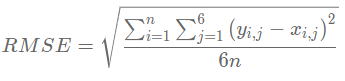
1. from collections import defaultdict
3. model.eval()
5. def predict(model, test\_loader):
6. val\_loss = 0
7. test\_pred = defaultdict(list)
8. model.eval()
9. model.cuda()
10. **for**  batch in tqdm(test\_loader):
11. b\_input\_ids = batch['input\_ids'].cuda()
12. attention\_mask = batch["attention\_mask"].cuda()
13. with torch.no\_grad():
14. logists = model(input\_ids=b\_input\_ids, attention\_mask=attention\_mask)
15. **for** col in target\_cols:
16. out2 = logists[col].sigmoid().squeeze(1)\*3.0
17. test\_pred[col].extend(out2.cpu().numpy().tolist())
19. **return** test\_pred

集成模块（vote.py）：使用不同的分词器得到了不同的结果，我们使用投票法进行集成，可惜实验结果表明没有单一的方法高。

七、实验结果分析



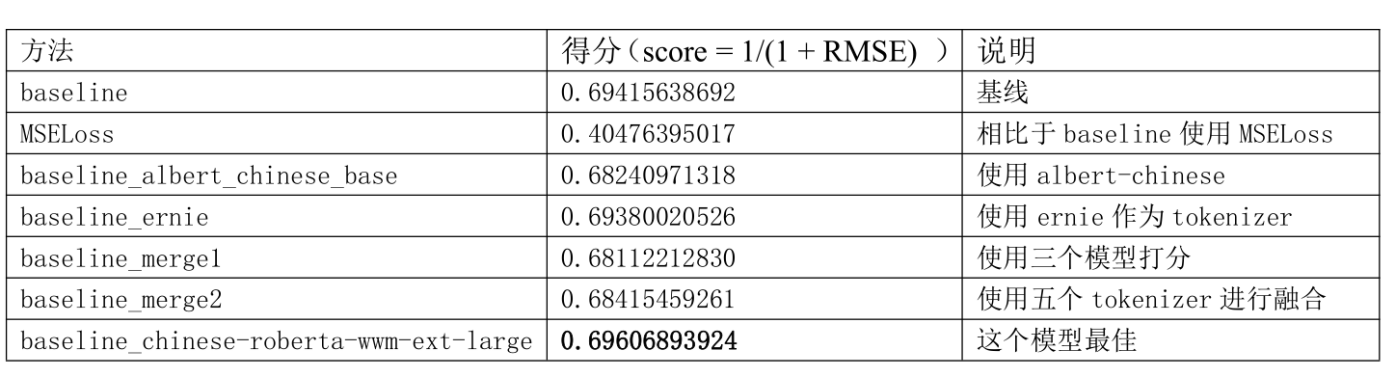
在比赛过程中，我们提交的最好成绩如上图所示。比赛的评测标准如下：赛题算法评分采用常用的均方根误差（RMSE）来计算评分，按照“文本内容+角色名”识别出的6类情感对应的情感值来统计。



score = 1/(1 + RMSE)

其中yi,j是预测的情感值，xi,j是标注的情感值，n是总的测试样本数。也就是说得分越高误差越小，我们最终的RMSE为0.43664。每一个错判的情感类型都会增加RMSE；即使情感类型判断正确，对情感程度的取值不准仍然会增加RMSE。

实验各个方法如表：



可以看到 chinese-roberta-wwm-ext-large 是这些方法中得到分数最高的，这也是我们队的最高分。对比于本实验的尺度，我们的RMSE已经算是控制得比较小，模型在比赛的验证集上取得了较好的效果，但距离榜首的0.40105的RMSE仍有一定差距。

对于这个结果，我们认为chinese-roberta-wwm-ext-large在这个数据集上的效果提升来源于使用更大的预训练的模型得到的词向量的更好的表示能力。baseline\_merge是采用了集成学习的结果，使用多个模型进行打分，并选择得分最高的值作为输出。从实验结果可以看到，使用不同tokenizer进行集成，结果相比使用单一的tonkenizer并不能得到提升，应该是因为不同的tokenizer的结果并不具有足够的差异性，因而在集成的时候，简单的投票并不会对结果产生一个正向的效果，而是会得到一个更加平均的结果，故集成的结果优于最差的模型，却不如最好的模型。

通过对于数据的分析，我们知道了每个id通过script\_id，scene\_num,以及sentence\_num组成，而每个emotion是和这几项单独相关的，因此我们做了特征工程，将id中的这三个编号添加到context最后，当做输入的一部分先验信息。实验结果表明，优化前的score = 0.69472047458，优化后的score = 0.69606893924。这个结果表明了emotion和剧本，场景的相关性。

我们还尝试了将数据集按照剧本，场景，台词的顺序进行排序，尝试利用上下文关系来进行情感推断。但实验结果没有得到提升，合并上下文句子之后作为输入进行训练，得到的结果为score = 0.62906298487。结果下降的原因可能是上下文的输入引入了额外的干扰，导致对当前句子中的角色的emotion的预测不准确。可能需要再对上下文进行些额外的处理，以利用其有效信息而排除干扰信息。

八、参考文献

《Speech and Language Processing(3rd ed.)》, Dan Jurafsky and James H. Martin

《自然语言处理入门》，何晗

知乎专栏-剧本角色情感识别baseline-pytorch-v2，<https://zhuanlan.zhihu.com/p/428941904>

Cui Y, Che W, Liu T, et al. Pre-training with whole word masking for chinese bert[J]. arXiv preprint arXiv:1906.08101, 2019.