自然语言处理技术报告-文本分类

袁昭新 2020K8009926029

2023 年 7 月 13 日

目录

1 概述

本次作业中使用了公开的酒店评论数据集 [2] 进行文本分类，使用了统计方法——贝叶斯分类器和神经网络方法——CNN、Transformer 进行文本分类。并对分类方法的性能进行了比较，对错误分类的样本进行了分析。

2 数据集

使用的公开酒店评论数据集包含了酒店评论的文本和对应的标签，标签共有 2 类，分别为正面评论和负面评论。数据集中包含 7766 条评论，其中正面评论约 5300 条，负面评论约 2400 条。

3 数据预处理

3.1 划分

首先将数据集划分为训练集、验证集和测试集，其中训练集占 90%，测试集各占 10%。采用 sklearn 中提供的 train_test_split 函数进行划分，将原数据划分后重新保存为 train.csv 和 test.csv。并修改了保存的格式，方便后续读取。

3.2 分词

使用 jieba 对文本进行分词，对于任意输入文本，返回分词后的结果。

```
1 def tokenizer(text):
2     return [word for word in jieba.cut(text) if word.strip()]
```

在不同模型中，仍然有更细致和不同的数据预处理过程，将在各个模型中详细介绍。

4 模型

分类所采用的模型包括统计方法——贝叶斯分类器和神经网络方法——CNN、Transformer。

4.1 贝叶斯分类器

贝叶斯分类器是一种基于贝叶斯定理的分类器，其计算方式如 1 所示。

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} \tag{1}$$

其中，$P(c|x)$ 表示在给定 x 的条件下，c 的概率，$P(x|c)$ 表示在给定 c 的条件下，x 的概率，$P(c)$ 表示 c 的概率，$P(x)$ 表示 x 的概率。

贝叶斯分类器的计算过程如下所示。

1. 计算每个类别的先验概率 $P(c)$。
2. 计算每个类别中每个特征的条件概率 $P(x|c)$。
3. 对于给定的输入 x，计算每个类别的后验概率 $P(c|x)$。
4. 选择具有最大后验概率的类别作为输出。

4.1.1 预处理

在提取特征之前，先删除文本中的停用词，停用词来自开源的停用词表，包含了中文中常见的停用词，例如：

```
这
这个
这么
这么些
这么样
这么点儿
这些
这会儿
这儿
这就是说
```

在贝叶斯分类器中，使用的分类特征是通过 TF-IDF 提取的稀疏矩阵，TF-IDF 的计算方式如 2 所示。

$$\begin{aligned} \mathrm{TF}(t) &= \frac{\text{该词在文档中出现的次数}}{\text{文档中所有词的出现次数之和}} \\ \mathrm{IDF}(t) &= \log \frac{\text{语料库中文档的总数}}{\text{包含该词的文档数}+1} \\ \text{TF-IDF}(t) &= \mathrm{TF}(t) \times \mathrm{IDF}(t) \end{aligned} \tag{2}$$

具体实现上，使用了 sklearn 中提供的 TfidfVectorizer 类进行 TF-IDF 的计算，并在训练集上拟合和转换，然后在测试集上进行转换。

```
tfidf = TfidfVectorizer(tokenizer=tokenizer, stop_words=stop_words)
train_features = tfidf.fit_transform(train['text'])
test_features = tfidf.transform(test['text'])
```

4.1.2 训练与评估

使用 sklearn 中提供的 MultinomialNB 类进行训练，相关的代码如下所示。

```
clf = MultinomialNB(alpha=0.001).fit(train_features, train_labels)
```

在测试集上进行预测，然后使用 sklearn 中提供的 accuracy_score 函数计算准确率。

```
# 预测
predicted_labels = clf.predict(test_features)
# 评估
print('准确率为：', accuracy_score(test_labels, predicted_labels))
```

使用贝叶斯分类器进行分类，准确率为 84.70%。另外，将预测错误的结果保存到文件中，方便后续分析。

4.2 CNN

使用 CNN 模型进行文本分类，首先用 Pytorch 中的 torchtext 进行数据的预处理，然后使用 Pytorch 搭建 CNN 模型。

4.2.1 预处理

使用 torchtext 中的 data.Field 类对文本进行预处理，相关的代码如下所示。

```
# 读取数据
text_field = data.Field(lower=True, tokenize = tokenizer)
label_field = data.Field(sequential=False)
filds = [('label', label_field), ('text', text_field)]
train_dataset, test_dataset = data.TabularDataset.splits(
    path = '', format = 'tsv', skip_header = False,
    train = 'train.tsv', test = 'test.tsv', fields = filds
)
# 构建词典
text_field.build_vocab(train_dataset, test_dataset, \
    min_freq = 5, max_size = 50000)
label_field.build_vocab(train_dataset, test_dataset)
# 构建迭代器
train_iter, test_iter = data.Iterator.splits((train_dataset, test_dataset),
    batch_sizes = (batch_size, batch_size), sort_key = lambda x: len(x.text))
```

4.2.2 模型搭建

使用 Pytorch 搭建 CNN 模型，模型的具体实现参考了 [1] 中的实现方式，相关的代码如下所示。

```
class TextCnn(nn.Module):
    def __init__(self, embed_num, embed_dim, class_num, kernel_num,
        kernel_sizes, dropout = 0.5):
        super(TextCnn, self).__init__()
        Ci = 1
        Co = kernel_num

        self.embed = nn.Embedding(embed_num, embed_dim)
        self.convs = nn.ModuleList([nn.Conv2d(Ci, Co, (f, embed_dim), \

        padding = (2, 0)) for f in kernel_sizes])
        self.dropout = nn.Dropout(dropout)
        self.fc = nn.Linear(Co * len(kernel_sizes), class_num)

    def forward(self, x):
        x = self.embed(x)
        x = x.unsqueeze(1)
        x = [F.relu(conv(x)).squeeze(3) for conv in self.convs]
        x = [F.max_pool1d(i, i.size(2)).squeeze(2) for i in x]
        x = torch.cat(x, 1)
        x = self.dropout(x)
        output = self.fc(x)
        return output
```

4.2.3 训练与评估

使用 Adam 优化器，学习率为 0.0007，batch size 为 64，训练 20 个 epoch。在训练的每个 step 记录 loss 和 accuracy，在每个 epoch 后对模型进行评估，记录其在测试集合上的准确率。最后将训练过程中的 loss、accuracy 和测试集上的准确率绘制成曲线图，如图 1 和 2 所示。

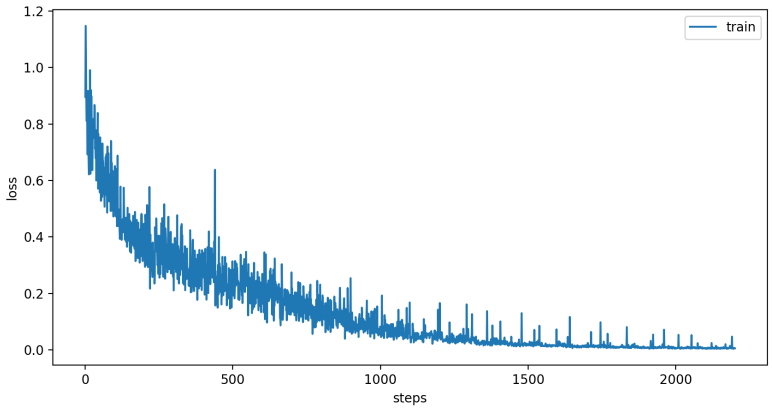

Figure 1: CNN 模型训练过程中的 loss

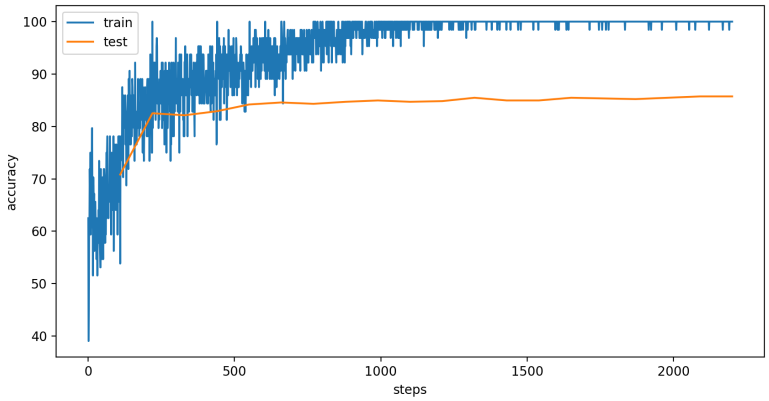

Figure 2: CNN 模型训练过程中的 accuracy

模型大约在 1250 个 step 后收敛，此时在测试集上的准确率为 85.73%。同样将预测错误的结果保存到文件中，方便后续分析。

4.3 Transformer

使用基于 Transformer 的 BertForSequenceClassification 模型进行文本分类，在预训练的 bert-base-chinese 模型上进行微调。

4.3.1 预处理

使用预训练的 bert-base-chinese 进行 token 化。

```
1 tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
```

4.3.2 模型搭建

加载预训练的 bert-base-chinese 模型，在此基础上上进行微调。

```
model = BertForSequenceClassification.from_pretrained('bert-base-chinese')
```

4.3.3 训练与评估

使用 Adam 优化器，学习率为 0.00001，batch size 为 16，训练 10 个 epoch。训练过程中，每个 step 后记录 loss，每个 epoch 后对模型进行评估，记录其在测试集合上的准确率。最后将训练过程中的 loss 和测试集上的准确率绘制成曲线图，如图 3 和 4 所示。

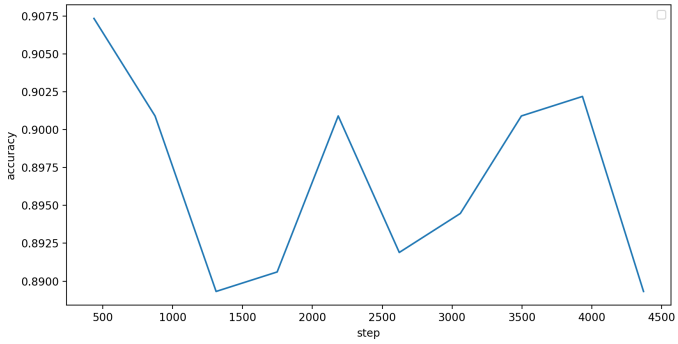

Figure 3: Bert 分类器训练过程中的 loss，学习率为 0.00001，batch size 为 16

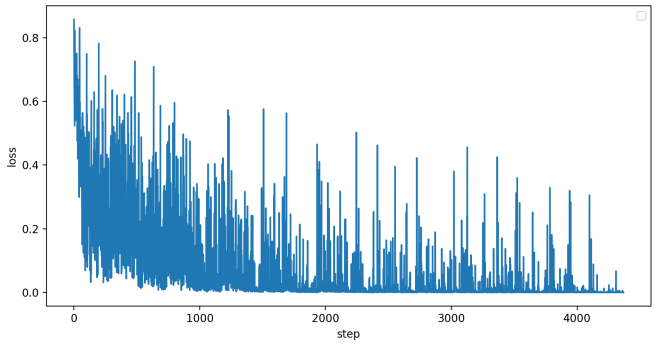

Figure 4: Bert 分类器训练过程中的 accuracy，学习率为 0.00001，batch size 为 16

模型几乎在第一个 epoch 后就在测试集上达到了最好的效果，为 90.73%，并且随着训练的进行，模型的效果越来越差。随后尝试了更高的学习率以及更小的 batch size，但效果变得非常差。

猜测可能的原因是使用的学习率过大，一般认为 fine-tuning 时学习率应该比较小，因此将学习率调整为 7e-6，同时将 batch size 调整为 8，epoch 调整为 20。调整后，模型最高的准确率达到了 91.51%，并且平均正确率也有所提升，说明调整后的参数更合理。调整后，训练过程中的 loss 和 accuracy 如图 5 和 6 所示。

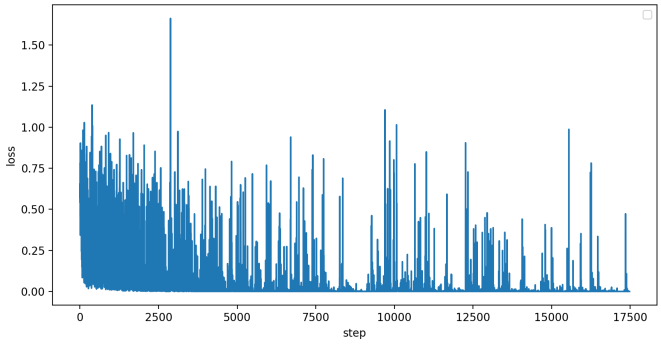

Figure 5: Bert 分类器训练过程中的 loss，学习率为 0.000007，batch size 为 8

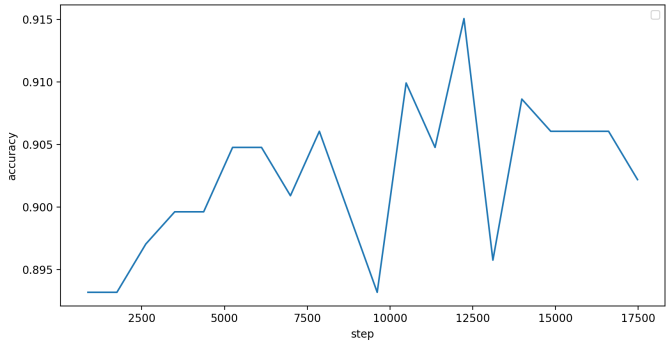

Figure 6: Bert 分类器训练过程中的 accuracy，学习率为 0.000007，batch size 为 8

同样，将预测错误的结果保存到文件中，方便后续分析。

5 错误分析

总体而言，三种模型的准确率都在 80% 以上，使用预训练的 BertForSequenceClassification 模型的准确率最高，为 91.51%。汇集三种模型的错误分类结果，三种模型都分类错误的结果如下，lb 表示标签（1 为积极，0 为消极），pr 表示预测结果，text 表示文本。

	lb	pr	text
2	1	0	地角-虽然打车去哪都不贵但是前不着村后不着店步行太冷了不过早上吃饭还是很方便客房很烂~似乎是后改造的跟北京的洲际没法比人家那个是闪闪发亮这个是暗淡无光还很阴冷~安排的朝北的房子朋友先洗澡结果说淋浴的下水道堵了里面全是头发恶心无比地砖也是脏渍斑驳的里面配套的毛巾什么的根本就不全浴巾和浴袍各只有一条我宁愿加点钱住别家真不知道5星是怎么得来的
3	1	0	就是隔音比较差，房间外人说话一清二楚。其他都可以，在赣州都敢有什么奢求了
4	1	0	很一般的酒店，因为定不到赣电了勉强入住下吧
5	1	0	房间比较旧，骚扰电话太多，一个接一个，受不了补充点评2008年6月27日：忘了说，大床房是用两张小床拼的，不平，千万别住
6	1	0	我于10月28日入住酒店.白天游览黄龙后,晚上7点登记入住,结果房间门卡有问题,以至我楼上楼下跑了三趟才能进入房间,这太不应该了.另外该酒店在预订时强制选择信用卡担保,当行程临时改变不能入住时,携程答复是:酒店将自动从信用卡中扣除当天的房费,太霸道了.
7	0	1	主要是办事便利才选择“霉园”宾馆房间有霉味耶三星宾馆可能是淮安标准没有服务连台灯都没有不过早餐是绝对绿色五谷杂粮品种也多
8	1	0	大堂大门口的平开合式的塑钢门边缘很锋利，没有进行任何的软性保护包装，也没有警示提醒，导致11日进入时手指夹伤。尽管大堂经理马上帮助进行了包扎处理。仍旧觉得安全之事无小事，这些细节是体现管理水准的重要地方。已经再三告知酒店管理人员，尽快做出修整处理，防止类似情况发生。
9	1	0	马马乎乎!能住.标准不要太高.在小街上,出门没法打车.还行!住吧!
10	0	1	房间还比较干净,交通方便,离外滩很近.但外面声音太大,休息不好
11	1	0	隔音效果确实不好，但好象还没到影响到睡眼，毕竟是走廊不是大街；但是在电梯里有湘西人（酒鬼酒开的）抽烟大谈牌经。让人不快！！！（没有看不起湘西人）
12	0	1	酒店生意清淡，大堂里都没几个客人。房间倒是不小，但觉得被褥窗帘都灰蒙蒙的有股味道，不知道多久没人用了。酒店里的娱乐设施，包括游泳池，台球等等，都要收费。白天房间里没有空调。晚上的热水，要先放10分钟才会来。优点么，就是位置不错，出门就是老街，吃饭逛街还算方便。
13	1	0	本来觉得这个宾馆住得挺好的,没想到退房的时候出事了,烟灰缸没了前台认定是我们拿了，我们一来全是女的不抽烟二来没小孩会打碎,真是奇了怪了,碰到这种事情真是说也说不清楚了,还好后来经理出现事情才算了结,没被继续冤枉.但是时间还是浪费了半个小时,还生了点气.前台态度不大好,房间挺宽敞,有电脑,海景也不错，但是到半夜的时候真的是太吵了，因为旁边就是马路，三亚市区好象没有禁止喇叭的规定,吵死了,没怎么睡好

将正面样本预测为负面的结果中，所有的评价明显都是负面的，可能是客人在评价时不认真打分导致的。

将负面样本预测成正面的，往往是因为在评价中除了负面评价外，还有一些积极的描述，导致分类错误。

可以肯定的是，三个模型都预测错误的这些文本的评价行为都是不合理的，即便是人类也无法正确判断这些文本的情感极性。而对于每个模型不同的预测错误的文本，其错误的原因也大多是因为评价的情感极性不明显，或者是评价的情感极性与评价的内容不一致。

6 性能比较

本次作业中使用了三种模型进行文本分类，分别是贝叶斯分类器、CNN 和 Transformer。其中，贝叶斯分类器是统计方法，CNN 和 Transformer 是神经网络方法。

在速度上，贝叶斯分类器的训练时间和推理速度都远远快于 CNN 和 Transformer，因为其计算量较小。最慢的是 Transformer，因为其参数量较大，训练和推理所需要的时间都较长。

在准确率上，Transformer 的准确率最高，为 91.51%，CNN 的准确率次之，为 85.73%，贝叶斯分类器的准确率最低，为 84.70%。基于 Transformer 的 BertForSequenceClassification 模型基于预训练的 bert-base-chinese 模型，它能够学习到更多的语义信息，因此准确率更高。CNN 模型的准确率次之，因为其能够学习到局部的语义信息，但是由于其模型结构的限制，可能无法学习到全局的语义信息。贝叶斯分类器的准确率最低，因为其只利用了词频信息，无法真正学习到语义信息。若抛去数据集中存在的不合理数据，三种模型的正确率都能再提高一些。

7 总结

本次作业中使用了三种模型对开源的酒店评论数据集进行文本分类，分别是贝叶斯分类器、CNN 和 Transformer。其中，贝叶斯分类器是统计方法，CNN 和 Transformer 是神经网络方法。

通过具体实践，包括数据预处理、模型搭建、训练与评估、错误分析和性能比较，我对文本分类有了更深入的理解，体会到了良好的数据集、数据预处理的重要性。此外，也对不同模型的优缺点、擅长的任务有了更清晰的认识。

参考文献

[1] finisky. Textcnn. `https://github.com/finisky/TextCNN`, 2022.

[2] SophonPlus. Chinesenlpcorpus. `https://github.com/SophonPlus/ChineseNlpCorpus`, 2019.