合肥工業大學

数 据 挖 掘 实 验 报 告

实验名称：短文本分类任务（设计型）

学生姓名：徐翊航

班　　级：物联网工程 19-1 班

学　　号：2019217093

计算机与信息学院

2022 年 6 月 20 日

目录

一、 实验目的 2

二、 实验任务 2

（1）数据集 2

（2）方法要求 2

（3）结果要求 2

三、 实验内容 3

3.1BERT 3

（1）Masked LM 5

（2）Next Sentence Prediction 6

3.2 数据获取 6

3.3 数据预处理 7

3.4 模型训练与测试 9

四、 实验分析 11

4.1 实验细节 11

4.2 实验比较 11

五、 实验总结 14

一、实验目的

1）理解短文本稀疏性对分类任务的挑战；

2）考察学生对数据预处理步骤的理解，强化短文本预处理的重要性；

3）基模型可以调用已有的包，训练学生熟悉数据挖掘的基本框架；

4）学会多维度的对模型进行评估以及模型中参数的讨论。

二、实验任务

（1）数据集

1）新闻文本分类为中文数据集，需要进行一定的预处理，包括分词，去停用词等；

2）数据中的其他问题可自行酌情处理；

数据说明：自行划分 train 和 test，一般按 7：3 划分。

（2）方法要求

1）要有针对数据特点的预处理步骤，包括去停用词，降维等；重 点关注短文本的特征扩充方法；

2）原则上不限制模型，决策树，NB，NN，SVM，random forest 均可，且不限于上述方法，重点实现适用于短文本的模型；

3）可采用 BOW，主题模型以及词向量等多种表示方式

（3）结果要求

1）实现一个或多个基本分类模型，并计算其评估指标如准确率，召回率等；

2）对模型中的关键参数，（如决策树中停止分裂条件，NN 中层数等参数）进行不同范围的取值，讨论参数的最佳取值范围。

3）对比分析不同的特征表示方法对结果的影响；对比短文本模型和一般模型的分类表现。

4）若对同一数据采用两种或多种模型进行了分类，对多种模型结

果进行对比，以评估模型对该数据集上分类任务的适用性。

5）选做：针对某一问题，改进数据或模型，以提高当前的精度。

三、实验内容

3.1BERT

BERT 来自 Google 的论文 Pre-training of Deep Bidirectional Transformers for Language Understanding， BERT 是“Bidirectional Encoder Representations from Transformers”的首字母缩写，整体是一个自编码语言模型（Autoencoder LM），并且其设计了两个任务来预训练该模型。这两个任务分别是：

（1）第一个任务是采用 MaskLM 的方式来训练语言模型，通俗地说就是在输入一句话的时候，随机地选一些要预测的词，然后用一个特殊的符号[MASK]来代替它们，之后让模型根据所给的标签去学习这些地方该填的词；

（2）第二个任务在双向语言模型的基础上额外增加了一个句子级别的连续性预测任务，即预测输入 BERT 的两段文本是否为连续的文本，引入这个任务可以更好地让模型学到连续的文本片段之间的关系。

最后的实验表明 BERT 模型的有效性，并在 11 项 NLP 任务中夺得 SOTA 结果。BERT 相较于原来的 RNN、LSTM 可以做到并发执行，同时提取词在句子中的关系特征，并且能在多个不同层次提取关系特征，进而更全面反映句子语义。相较于 word2vec，其又能根据句子上下文获取词义，从而避免歧义出现。同时缺点也是显而易见的，模型参数太多，而且模型太大，少量数据训练时，容易过拟合。

3.1.1BERT 是怎么使用 Transfomer?

BERT 只使用了 Transformer 的 Encoder 模块，原论文中，作者分别用 12 层和 24 层 Transformer Encoder 组装了两套 BERT 模型，分别是：

$$BERT_{BASE}: L = 12, H = 768, A = 12, TotalParameters = 110M$$

$$BERT_{LARGE} = L = 24, H = 1024, A = 16, TotalParameters = 340M$$

其中层的数量（即，Transformer Encoder 块的数量）为 L ，隐藏层的维度为 H ，自注意头的个数为 A 。在所有例子中，我们将前馈/过滤器（Transformer Encoder 端的 feed-forward 层）的维度设置为 4H，即当 H=768 时是 3072 ；当 H=1024 是 4096 ，如图 3.1 所示。

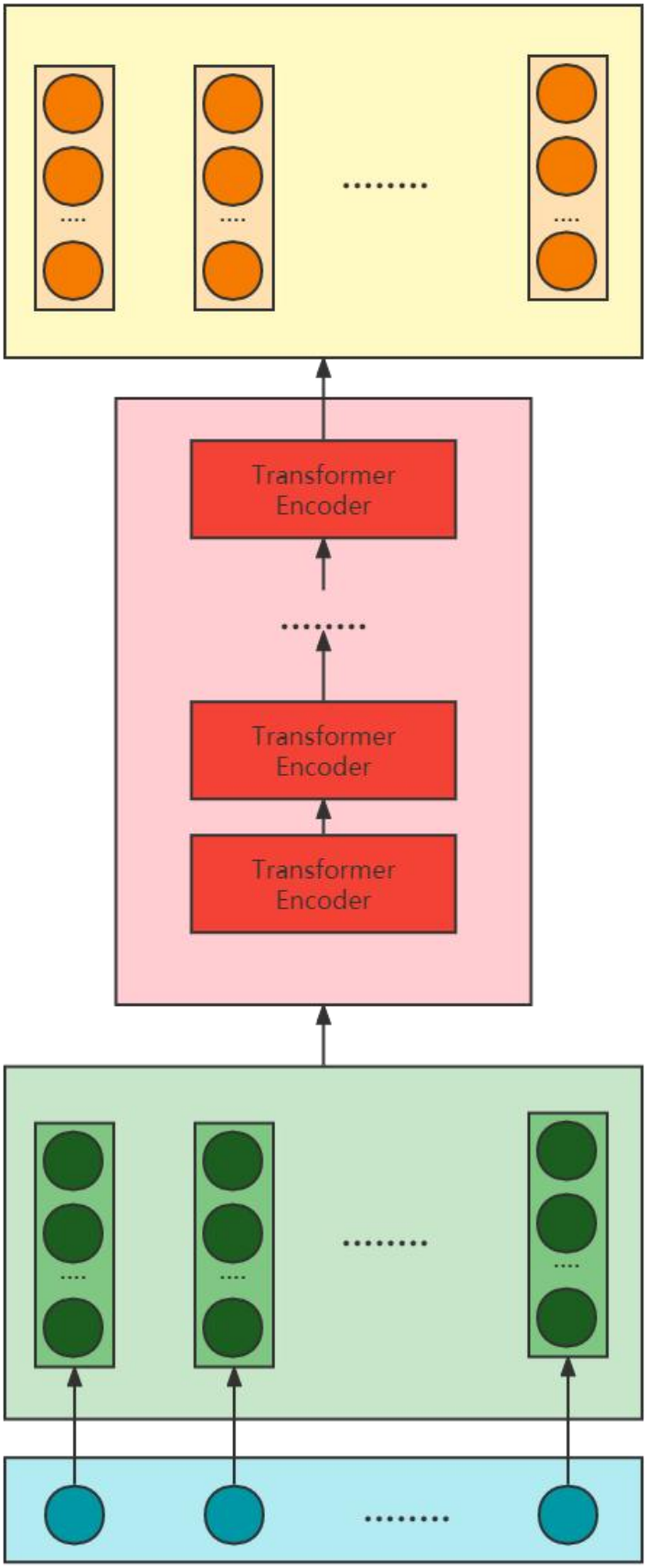

Transformer
Encoder
Transformer
Encoder
Transformer
Encoder

图 3.1 BERT 流程图

3.1.2 BERT 的训练过程

（1）Masked LM

Masked LM 的任务描述为：给定一句话，随机抹去这句话中的一个或几个词，要求根据剩余词汇预测被抹去的几个词分别是什么，如下图 3.2 所示。

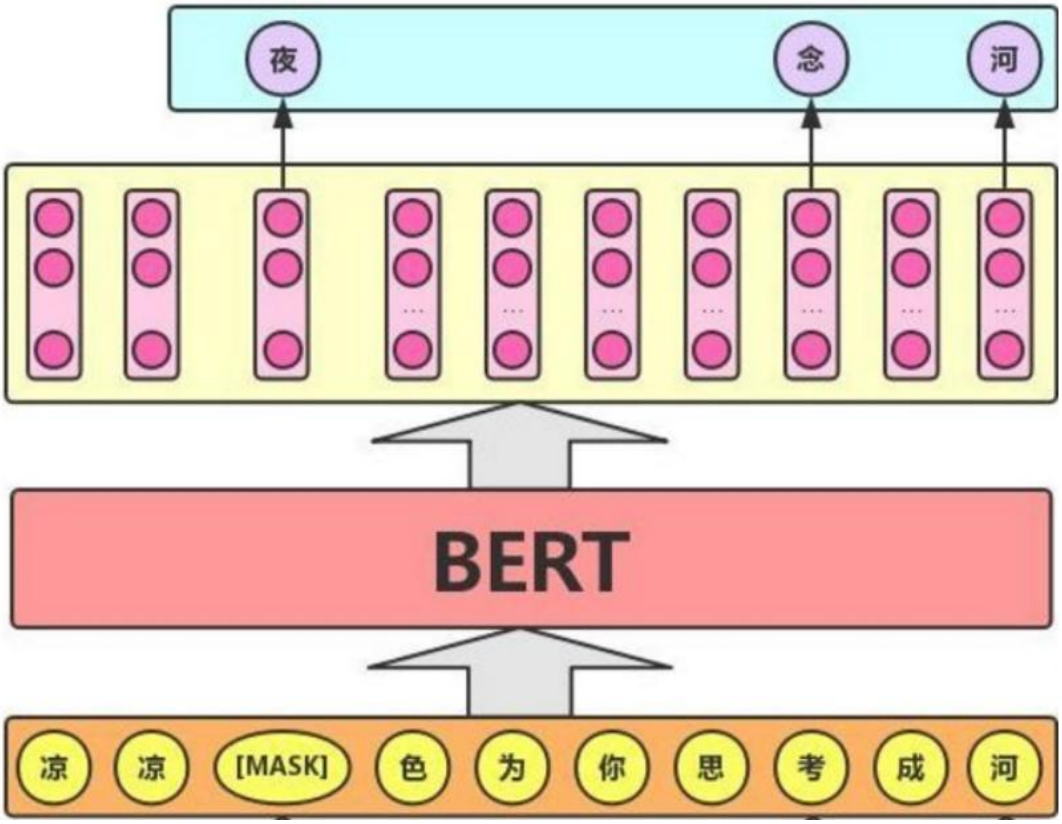


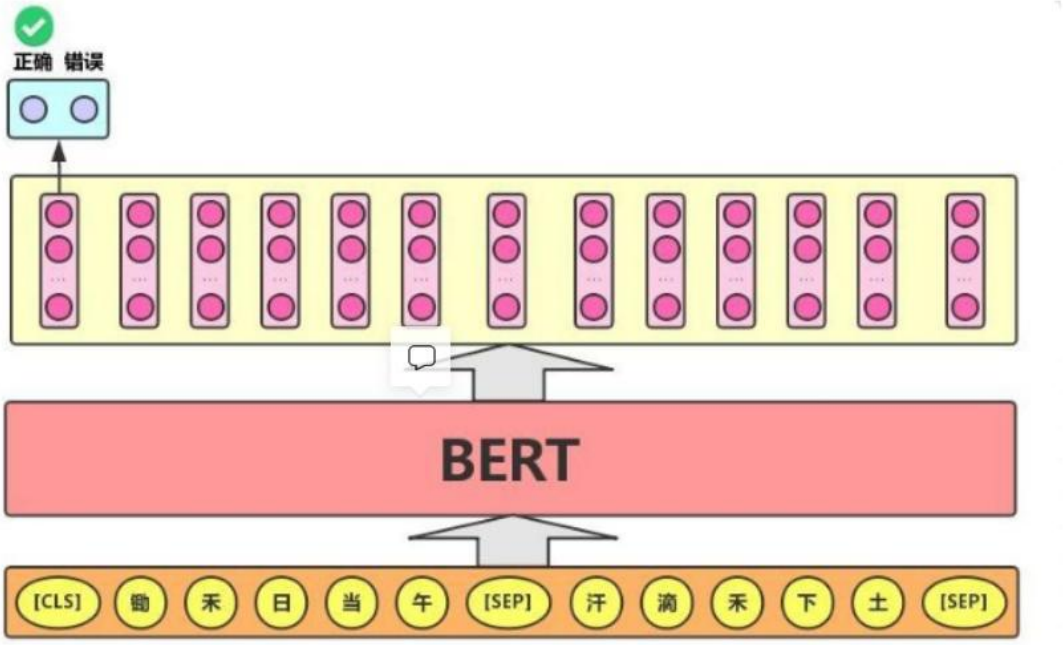
图 3.2 Masked LM 任务过程

BERT 模型的这个预训练过程其实就是在模仿我们学语言的过程，思想来源于「完形填空」的任务。具体来说，文章作者在一句话中随机选择 15% 的词汇用于预测。对于在原句中被抹去的词汇， 80% 情况下采用一个特殊符号 [MASK] 替换， 10% 情况下采用一个任意词替换，剩余 10% 情况下保持原词汇不变。

这么做的主要原因是：在后续微调任务中语句中并不会出现 [MASK] 标记，而且这么做的另一个好处是：预测一个词汇时，模型并不知道输入对应位置的词汇是否为正确的词汇（ 10% 概率），这就迫使模型更多地依赖于上下文信息去预测词汇，并且赋予了模型一定的纠错能力。上述提到了这样做的一个缺点，其实这样做还有另外一个缺点，就是每批次数据中只有 15% 的标记被预测，这意味着模型可能需要更多的预训练步骤来收敛。

（2）Next Sentence Prediction

Next Sentence Prediction 的任务描述为：给定一篇文章中的两句话，判断第二句话在文本中是否紧跟在第一句话之后，如下图 3.3 所示。



这个类似于「段落重排序」的任务，即：将一篇文章的各段打乱，让我们通过重新排序把原文还原出来，这其实需要我们对全文大意有充分、准确的理解。

Next Sentence Prediction 任务实际上就是段落重排序的简化版：只考虑两句话，判断是否是一篇文章中的前后句。在实际预训练过程中，文章作者从文本语料库中随机选择 50% 正确语句对和 50% 错误语句对进行训练，与 Masked LM 任务相结合，让模型能够更准确地刻画语句乃至篇章层面的语义信息。

BERT 模型通过对 Masked LM 任务和 Next Sentence Prediction 任务进行联合训练，使模型输出的每个字 / 词的向量表示都能尽可能全面、准确地刻画输入文本（单句或语句对）的整体信息，为后续的微调任务提供更好的模型参数初始值。

3.2 数据获取

本实验中，我从 HuggingFcae 中获取数据集，获取数据集代码如下，展示前五行数据，可知每条数据由一个 text 和 label 组成，通过该数据集实现文本的情

感分类。

```
import torch
from datasets import load_dataset,load_from_disk
#定义数据集
class Dataset(torch.utils.data.Dataset):
    def __init__(self, split):
        self.dataset = load_from_disk('./data/ChnSentiCorp')[split]
    def __len__(self):
        return len(self.dataset)
    def __getitem__(self, i):
        text = self.dataset[i]['text']
        label = self.dataset[i]['label']
        return text, label
dataset = Dataset('train')
len(dataset)
for i in range(10):
    print(dataset[i])
```

输出结果如下：

```
('选择珠江花园的原因就是方便，有电动扶梯直接到达海边，周围餐馆、食廊、商场、超市、摊位一应俱全。酒店装修一般，但还算整洁。 泳池在大堂的屋顶，因此很小，不过女儿倒是喜欢。 包的早餐是西式的，还算丰富。 服务吗，一般', 1)
('15.4寸笔记本的键盘确实爽，基本跟台式机差不多了，蛮喜欢数字小键盘，输数字特方便，样子也很美观，做工也相当不错', 1)
('房间太小。其他的都一般。。。。。。。。。', 0)
('1.接电源没有几分钟,电源适配器热的不行. 2.摄像头用不起来. 3.机盖的钢琴漆，手不能摸，一摸一个印. 4.硬盘分区不好办.', 0)
('今天才知道这书还有第6卷,真有点郁闷:为什么同一套书有两种版本呢?当当网是不是该跟出版社商量商量,单独出个第6卷,让我们的孩子不会有所遗憾。', 1)
('机器背面似乎被撕了张什么标签，残胶还在。但是又看不出是什么标签不见了，该有的都在，怪', 0)
('呵呵，虽然表皮看上去不错很精致，但是我还是能看得出来是盗的。但是里面的内容真的不错，我妈爱看，我自己也学着找一些穴位。', 0)
('这本书实在是太烂了,以前听浙大的老师说这本书怎么怎么不对,哪些地方都是误导的还不相信,终于买了一本看一下,发现真是~~~无语,这种书都写得出来', 0)
('地理位置佳，在市中心。酒店服务好、早餐品种丰富。我住的商务数码房电脑宽带速度满意,房间还算干净，离湘南路小吃街近。', 1)
('5.1期间在这住的，位置还可以，在市委市政府附近，要去商业区和步行街得打车，屋里有蚊子，虽然空间挺大，晚上熄灯后把窗帘拉上简直是伸手不见五指，很适合睡觉，但是会被该死的蚊子吵醒！打死了两只，第二天早上还是发现又没打死的，卫生间挺大，但是设备很老旧。', 1)
```

3.3 数据预处理

首先我对收集到的数据进行了预处理操作，检查是否所有的数据都有 label 标签，对没有 label 标签的数据进行舍去，其次检查标签的值是否为 0 或 1，因为是进行二分类，对标签不符合的数据也进行舍去。

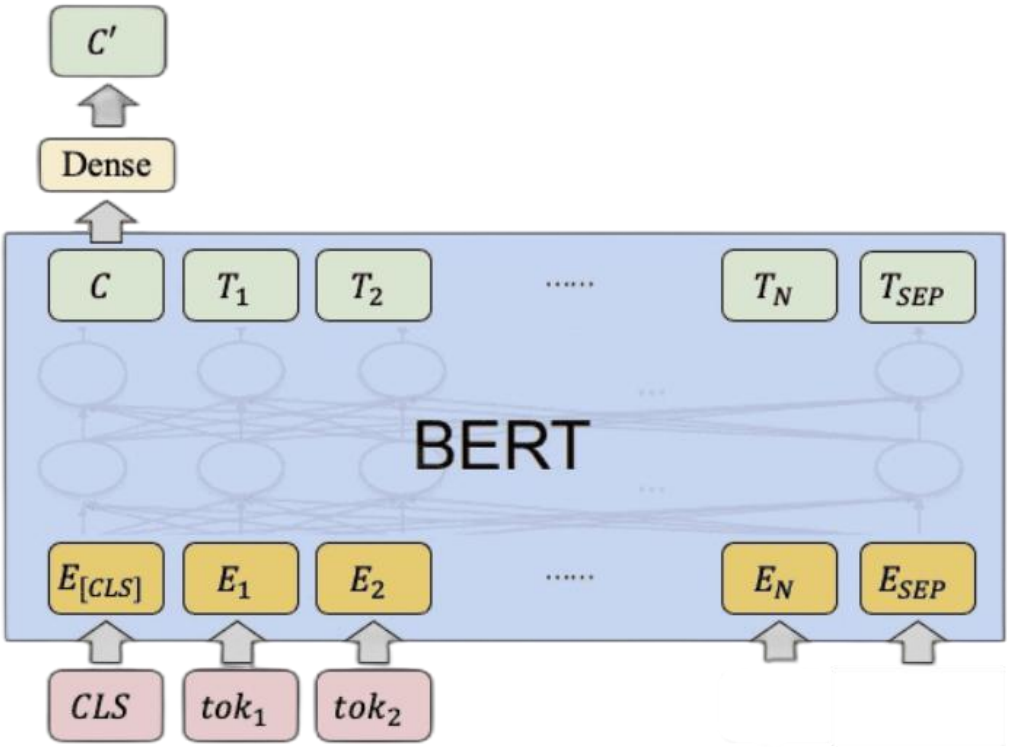
利用字典和分词工具 bert-base-chinese 对数据进行预处理，将文本转化为词向量的形式，每个词转化为一个 1*500 的矩阵，对超过 500 的部分清楚，不足的词进行补 0 操作，定义相应的批处理函数，因为在数据集中数据是一条一条的，而在训练和测试的时候，我们应该一批一批的来处理这些数据。随后将编码后的数据装入数据加载器中，以 16 条数据作为一轮进行向与训练模型进行输入，并查看数据样例，具体的代码和数据样例的查看结果如下图所示。

```
def collate_fn(data):
    sents = [i[0] for i in data]
    labels = [i[1] for i in data]
```

```
    data = token.batch_encode_plus(batch_text_or_text_pairs=sents,
                                   truncation=True,
                                   padding='max_length',
                                   max_length=500,
                                   return_tensors='pt',
                                   return_length=True)
    input_ids = data['input_ids']
    attention_mask = data['attention_mask']
    token_type_ids = data['token_type_ids']
    labels = torch.LongTensor(labels)
    return input_ids, attention_mask, token_type_ids, labels

loader = torch.utils.data.DataLoader(dataset=dataset,
                                     batch_size=16,
                                     collate_fn=collate_fn,
                                     shuffle=True,
                                     drop_last=True)
for i, (input_ids, attention_mask, token_type_ids,
        labels) in enumerate(loader):
    break
print(len(loader))
input_ids.shape, attention_mask.shape, token_type_ids.shape, labels
```

```
600

(torch.Size([16, 500]),
 torch.Size([16, 500]),
 torch.Size([16, 500]),
 tensor([1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1]))
```

定义下游任务模型，执行相应的单句子分类任务（属于序列任务），即对于每个输入输入序列，只计算 BERT 模型中一个输出的损失。利用 BERT 抽取数据中的特征，将模型抽取出来的特征放到全连接神经网络中，即在 [SEP] 位置对应的输出后增加一个分类层（全连接层+softxax 层）。而这个特征的结果只需要第 0 个词的特征即可，运算的结果将会变成一个二分类的结果。用于输出最后的分类概率修改后的模型如下图所示：



将 BERT 模型输出的结果输入到一个神经网络中，输入层由 728 个神经元，输出层为 2 个神经元且要以概率分布的形式输出，相关代码如下：

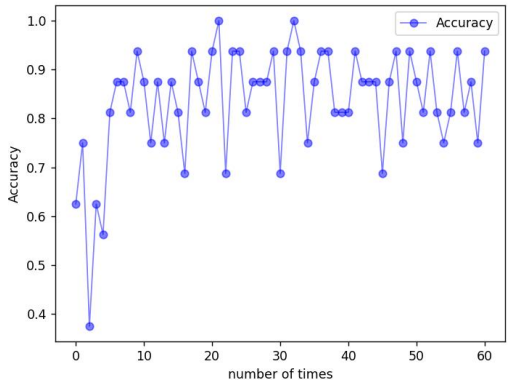
```
class Model(torch.nn.Module):
    def __init__(self):
        super().__init__()
        self.fc = torch.nn.Linear(768, 2)

    def forward(self, input_ids, attention_mask, token_type_ids):
        with torch.no_grad():
            out = pretrained(input_ids=input_ids,
                             attention_mask=attention_mask,
                             token_type_ids=token_type_ids)

        out = self.fc(out.last_hidden_state[:, 0])
        out = out.softmax(dim=1)
        return out

model = Model()
model(input_ids=input_ids,
      attention_mask=attention_mask,
      token_type_ids=token_type_ids).shape
```

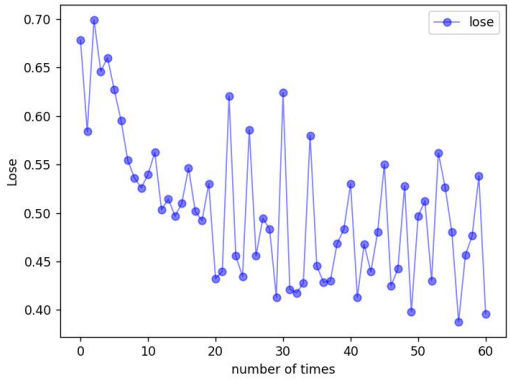
3.4 模型训练与测试

利用之前预训练得到模型对数据进行训练，从数据预处理得到的数据加载器向其模型输入，因训练时间较长，本次训练共训练 300 轮，每 5 论计算其损失函数的值与准确度并打印其值，相关代码和得到的结果如下：

```
0 0.6782209873199463 0.625
5 0.5840441584587097 0.75
10 0.6991336941719055 0.375
15 0.6459898948669434 0.625
20 0.6601291298866272 0.5625
25 0.6269813776016235 0.8125
30 0.595515251159668 0.875
35 0.5542500019073486 0.875
40 0.5360901355743408 0.8125
45 0.5258952379226685 0.9375
50 0.5396168828010559 0.875
55 0.5625669956207275 0.75
60 0.503290057182312 0.875
65 0.5144746899604797 0.75
70 0.4966588318347931 0.875
75 0.5100283622741699 0.8125
80 0.5462980270385742 0.6875
85 0.5015004873275757 0.9375
90 0.4921759068965912 0.875
95 0.5301690697669983 0.8125
100 0.4322471618652344 0.9375
105 0.4398854672908783 1.0
110 0.6205551028251648 0.6875
115 0.4555570185184479 0.9375
120 0.43458104133605957 0.9375
125 0.5856747031211853 0.8125
130 0.45579853653907776 0.875
135 0.49450209736824036 0.875
140 0.4834059476852417 0.875
145 0.41298091411590576 0.9375
150 0.6239964365959167 0.6875
155 0.42134153842926025 0.9375
160 0.41760149598121643 1.0
165 0.4275535047054291 0.9375
170 0.5800575613975525 0.75
175 0.44518887996673584 0.875
180 0.42843857407569885 0.9375
185 0.4298834800720215 0.9375
190 0.46833470463752747 0.8125
195 0.48308607935905457 0.8125
200 0.5299521684646606 0.8125
205 0.41276633739471436 0.9375
210 0.4676920771598816 0.875
215 0.4392228424549103 0.875
220 0.4800220727920532 0.875
225 0.5499932169914246 0.6875
230 0.4250292479991913 0.875
235 0.44226840138435364 0.9375
240 0.527982771396637 0.75
245 0.398252934217453 0.9375
250 0.4963655173778534 0.875
255 0.5123838186264038 0.8125
260 0.4301964342594147 0.9375
265 0.561985969543457 0.8125
270 0.5266203880310059 0.75
275 0.4799845516681671 0.8125
280 0.3876492977142334 0.9375
285 0.4564688503742218 0.8125
290 0.4763532280921936 0.875
295 0.538104772567749 0.75
300 0.39595580101013184 0.9375
```

将得到的结果绘制出相关的曲线，如下图所示：





由结果可知，当由 BERT 预训练模型来抽取特征，然后再做下游的一个迁移学习时，可以在非常短的时间内以一个非常快的速度，就可以达到一个很高的正确率。接下来在测试级上进行正确率的评估，最终达到了 86%的正确率。查阅相关材料得知，相比与其他文本情感分类模型，该模型已具备较好的效果。而当今在 BERT 模型提出后，所有的 NLP 任务都是基于预训练模型抽取特征，其不仅能节约极大的时间与精力，同时具备一个较好的模型训练效果。

四、实验分析

4.1 实验细节

4.1.1 开发环境

- 操作系统：windows 10
- IDE:Python3.8、pycharm
- pytorch 1.10.0

4.1.2 训练环境

- 硬件设施：GPU:RTX 3090 * 1 显存 24G、CPU:10 核 Intel(R) Xeon(R) Silver 4210R CPU @ 2.40GHz 内存:30GB
- 训练时长：4 小时

4.2 实验比较

4.2.1 神经网络模型训练

刚开始实验时，我并没有使用特征提取，而是对数据集做了简单的数据清洗任务，包括利用 jiaba 分词和去停词等，将每条数据转换为一个词向量，词向量的维度设置为 200，将词向量输入到一个单层神经网络中，其中 batch_size 设置为 32，共训练是个 epoch。打印每次的结果。

```
Epoch 1/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6407 - accuracy: 0.6366 - val_loss: 0.6354 - val_accuracy: 0.6473
Epoch 2/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6380 - accuracy: 0.6394 - val_loss: 0.6311 - val_accuracy: 0.6510
Epoch 3/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6374 - accuracy: 0.6395 - val_loss: 0.6327 - val_accuracy: 0.6498
Epoch 4/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6372 - accuracy: 0.6389 - val_loss: 0.6305 - val_accuracy: 0.6510
Epoch 5/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6372 - accuracy: 0.6392 - val_loss: 0.6310 - val_accuracy: 0.6498
Epoch 6/10
2527/2527 [==============================] - 3s 995us/step - loss: 0.6371 - accuracy: 0.6392 - val_loss: 0.6321 - val_accuracy: 0.6498
Epoch 7/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6370 - accuracy: 0.6390 - val_loss: 0.6306 - val_accuracy: 0.6507
Epoch 8/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6370 - accuracy: 0.6390 - val_loss: 0.6307 - val_accuracy: 0.6479
Epoch 9/10
2527/2527 [==============================] - 3s 1ms/step - loss: 0.6370 - accuracy: 0.6399 - val_loss: 0.6308 - val_accuracy: 0.6507
Epoch 10/10
2527/2527 [==============================] - 3s 999us/step - loss: 0.6368 - accuracy: 0.6397 - val_loss: 0.6305 - val_accuracy: 0.6510
Model: "sequential"
```

而随后，我进一步调整神经网络的层数，提高神经网络的层数，但是经过测试并没有提高相应的准确度，准确度仅能维持到 65%的附近。随后我对模型进行改良，在全连接神经网络前加入卷积神经网络包括卷积层、池化层多次迭代。采用 textcnn 模型，在卷积层中我们使用卷积核对 embedbed 做卷积处理，size 选取三个值分别为 2，3，4，使得特征学习包含临近词信息，建立上下文之间的关系，而在池化层中池化的操作是 max_pooling，就是将列向量中最大值取出来，

对输入补 0 做过滤（此外还有平均池化等等）池化操作是对整个向量，所以它的 shape 是[1,sentence_length-size+1,1,1]，其中 sentence_length-size+1 是上文提到的经过卷积处理后得到的列向量长度。然后对卷积得到的每个列向量进行池化操作后，会得到 size_len*num_filter 个元素，将他们合并在一起形成一个 size_len*num_filter 维的向量。最后将上面的结果输入到全连接层进行拟合，为了防止过拟合的出现，我在模型中加入了 dropout，并采用交叉验证的方式对数据集进行检验，即计算真实标签和预测标签之间的交叉熵损失。

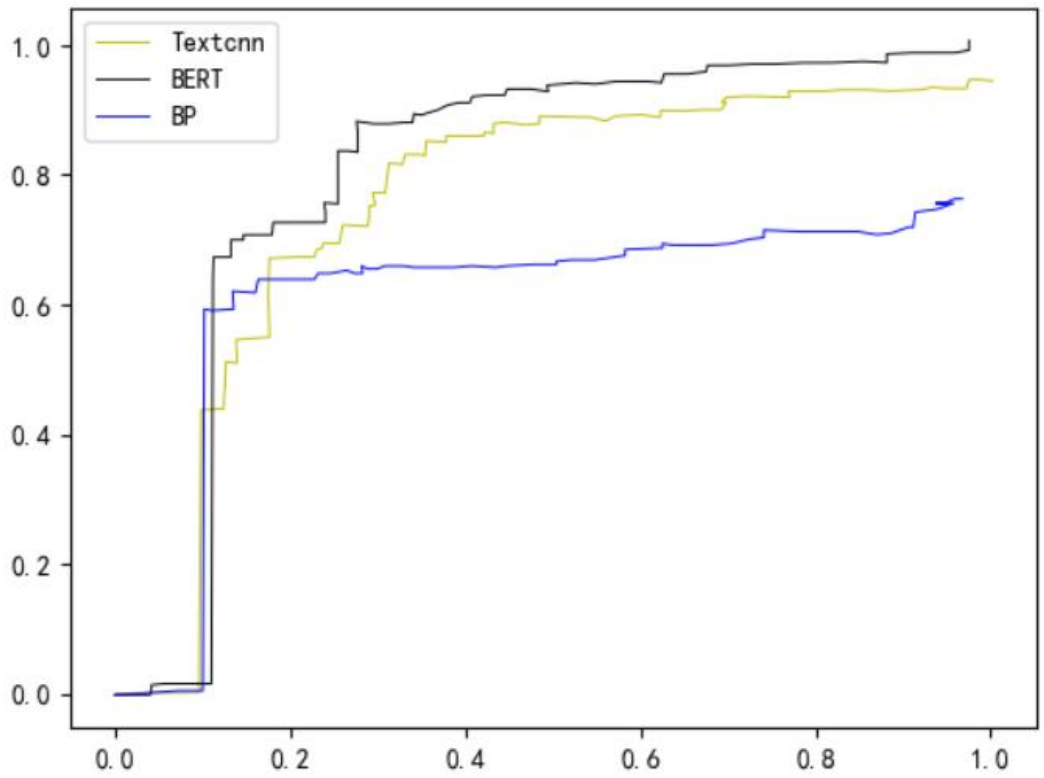
```
  def convolution(config):
      sequence_length=config.sequenceLength
      embedding_dimension=config.embeddingSize
      inn = Input(shape=(sequence_length, embedding_dimension, 1))
      cnns = []
      filter_sizes=config.filterSizes
      for size in filter_sizes:
          conv = Conv2D(filters=config.numFilters, kernel_size=(size, embedding_dimension),
                                strides=1, padding='valid', activation='relu')(inn)
          pool = MaxPool2D(pool_size=(sequence_length-size+1, 1), padding='valid')(conv)
          cnns.append(pool)
      outt =concatenate(cnns)
      model = Model(inputs=inn, outputs=outt)
      return model

 def cnn_mulfilter(n_symbols,embedding_weights,config):

      model =Sequential([
          Embedding(input_dim=n_symbols, output_dim=config.embeddingSize,
                            weights=[embedding_weights],
                            input_length=config.sequenceLength),
          Reshape((config.sequenceLength, config.embeddingSize, 1)),
          convolution(config),
          Flatten(),
          Dense(10, activation='relu',kernel_regularizer=regularizers.l2(config.l2RegLambda)),
          Dropout(config.dropoutKeepProb),
          Dense(1, activation='sigmoid')])
      model.compile(optimizer=optimizers.Adam(),
```

```
                    loss=losses.BinaryCrossentropy(),
                    metrics=['accuracy'])
    return model
wordEmbedding = data.wordEmbedding
n_symbols=data.n_symbols
model = cnn_mulfilter(n_symbols,wordEmbedding,config)
model.summary()
```

实验结果如下：

```
Epoch 1/10
885/885 [==============================] - 22s 12ms/step - loss: 0.4715 - accuracy: 0.8892 - val_loss: 0.2015 - val_accuracy: 0.9323
Epoch 2/10
885/885 [==============================] - 9s 11ms/step - loss: 0.2241 - accuracy: 0.9334 - val_loss: 0.1880 - val_accuracy: 0.9361
Epoch 3/10
885/885 [==============================] - 9s 11ms/step - loss: 0.1974 - accuracy: 0.9417 - val_loss: 0.2447 - val_accuracy: 0.9230
Epoch 4/10
885/885 [==============================] - 9s 10ms/step - loss: 0.1776 - accuracy: 0.9481 - val_loss: 0.2028 - val_accuracy: 0.9309
Epoch 5/10
885/885 [==============================] - 9s 10ms/step - loss: 0.1599 - accuracy: 0.9535 - val_loss: 0.2212 - val_accuracy: 0.9178
Epoch 6/10
885/885 [==============================] - 9s 10ms/step - loss: 0.1450 - accuracy: 0.9573 - val_loss: 0.2175 - val_accuracy: 0.9310
Epoch 7/10
885/885 [==============================] - 9s 10ms/step - loss: 0.1324 - accuracy: 0.9622 - val_loss: 0.2467 - val_accuracy: 0.9306
632/632 [==============================] - 2s 2ms/step - loss: 0.2517 - accuracy: 0.9290
```

由数据可以得知，经过 7 次训练后，准确度可以超过 90%，相比于神经网络，精确程度大大提高，而 BERT 模型需要多次训练才能得到好的效果，训练时间长对硬件要求高，而 textcnn 可以在短期内得到较好的结果，而 BERT 在多次训练后可以达到更好。下图为三种模型的 ROC 曲线。



五、实验总结

（1）通过本次实验，我搜索了许多有关短文本分类的方法。从最开始使用简单的全连接神经网络开始，再到 Textcnn 和 BERT 模型，我对短文本分类问题有了更深的了解。在本次实验中，也使得我体会了从数据的收集与清洗、模型的搭建与优化的全部过程，对数据挖掘的内容有了更加深刻的了解；

（2）在本次实验中，我首先通过自己事先的积累，搭建了一个简单的全连接神经网络进行训练，在发现训练效果不佳后，分别采用 Textcnn 和 BERT 模型进行训练，并对三者进行了对比，在这个过程中，其大大提升了我的 python 语言编程能力，尤其是对深度学习框架 tensorflow 和 pytorch 有了更好的掌握，对卷积神经网络也有了更好的了解，同时极大的锻炼了我的实验分析和逻辑表达能力，也让我对数据挖掘有了更加深刻的了解，我也希望以后从事数据挖掘有关的研究。在以后的学习中，我也会继续关注这方面的相关知识，不断提升自己的能力。

参考文献

[1] 王永恒,贾焰,杨树强.大规模文本数据库中的短文本分类方法[J].计算机工程与应用,2006,22:5~7

[2] Zelikovitz S,Transductive M F.Learning for Short-Text Classification Problem using Latent Semantic Intexing International [J].Journal of Pattern Recognition and Artificial Intelligence, 2005, 19（2）:143~163

[3]Pu Qiang,Yang Guo,Wei. Short -Text Classification Based on ICA and LSA [J]Proceedings of International Symposium on Neural Networks,2005,（ISNN 2）:265~270

[4]张俐,李晶皎,胡明涵等.中文 WordNet 的研究及实现[J]. 东北大学学报（自然科学版）,2003, 4（24）:147~150

[5]P.W.Lord, R.D.Stevens, A.Brass, C.A.Goble, Semantic Simi - larity Measures as Tools for Exploring the Genge Ontology[J] Proceedings of the 8th Pacific Symposium on Biocomputing, 2003