南開大學

Python 程序语言设计

基于 BERT 的真假新闻检测模型

学　　院　网络空间安全学院
专　　业　信息安全
学　　号　2212046
姓　　名　王昱
班　　级　信息安全班

2023 年 12 月 1 日

一、实验目的

- **问题描述**

在信息爆炸的时代，虚假新闻已经成为一个严重的社会问题。虚假新闻不仅误导公众，破坏社会秩序，还可能对个人、企业和国家的安全造成威胁。因此，自动化检测虚假新闻在当今时代格外具有意义和实用价值，而本次实验的目的便是机器学习来根据文本来预测新闻的真假。

- **具体要求**

给定一个信息的标题、出处、相关链接以及相关评论，尝试辨别信息真伪输入：信息来源、标题、超链接、评论、真伪标签（0：消息为真；1：消息 为假）。输出：通过模型构建、训练、测试，得到模型预测的准确率（accuracy）、精确度（precision）、召回率（recall）、ROC 曲线以及 AUC。

二、实验原理

（1）数据来源与获取

- 通过大数据竞赛平台获取数据集:数据集是中文微信消息，包括微信消息的 Official Account Name，Title，News Url，Image Url，Report Content，label。Title 是微信消息的标题，label 是消息的真假标签（0 是 real 消息，1 是 fake 消息）。其中训练数据保存在 train.news.csv，测试数据保存在 test.feature.csv，训练数据有 label，用来训练模型；而测试数据没有 label，需要我们将预测好的 label 提交到平台进行检测。文件读取可以通过 pandas 或者 python 自带的文件读取方式，本次实验使用了 pandas 来进行数据的读取与处理。
- 在 BERT 方法中需要用到中文文本分类预训练模型 bert-base-chinese，可以从 huggingface 的官网（bert-base-chinese at main (huggingface.co)）中下载。

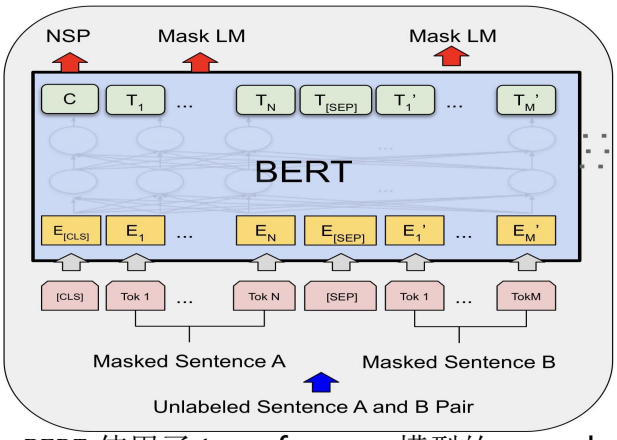
（2）方法选择及原理介绍

本次 python 大作业中我学习了许多种方法，包括贝叶斯分类器，岭回归线性分类器，随机森林等等，以及基于 Transformer 的预训练模型 BERT。最终训练的效果最好的是 BERT，LinearSVC 支持向量机次之。本次实验我将主要介绍 BERT：

- BERT（Bidirectional Encoder Representations from Transformers）是一种预训练模型，那么什么是预训练呢？举例子进行简单的介绍。假设已有 A 训练集，先用 A 对网络进行预训练，在 A 任务上学会网络参数，然后保存以备后用，当来一个新的任务 B，采取相同的网络结构，网络参数初始化的时候可以加载 A 学习好的参数，其他的高层参数随机初始化，之后用 B 任务的训练数据来训练网络，当加载的参数保持不变时，称为"frozen"，当加载的参数随着 B 任务的训练进行不断的改变，称为“fine-tuning”，即更好地把参数进行调整使得更适合当前的 B 任务。

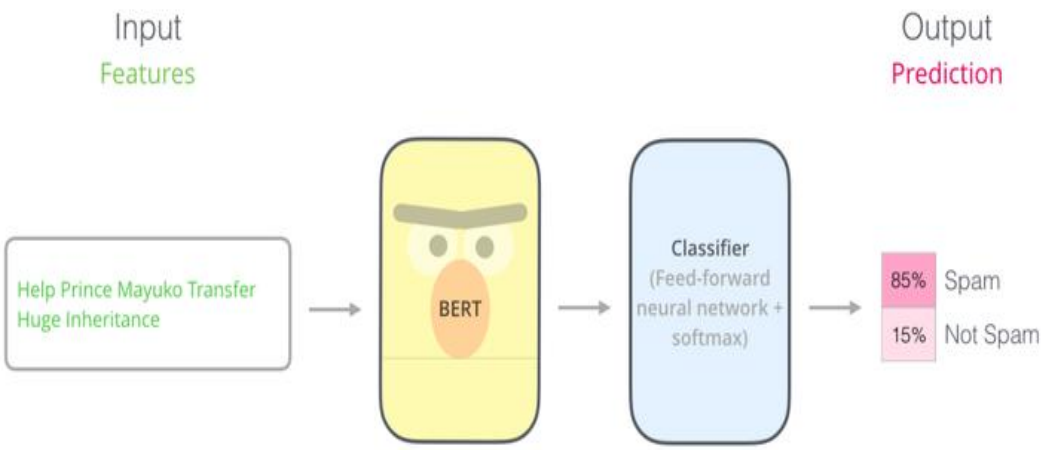
BERT 是自然语言处理（NLP）领域的重大里程碑。BERT 的设计理念和结构基于 Transformer 模型，通过无监督学习方式进行训练，并且在各种 NLP 任务中表现优异。那么为什么 BERT 的性能这么优越呢？这来源于它的第一个单词 Bidirectional，也就是 BERT 是一个双向语言模型。

以往的预训练模型的结构会受到单向语言模型的限制，因而也限制了模型的表征能力，使其只能获取单方向的上下文信息。而 BERT 利用 MLM 进行预训练并且采用深层的双向 Transformer 组件（单向的 Transformer 一般被称为 Transformer decoder，其每一个 token（符号）只会 attend 到目前往左的 token。而双向的 Transformer 则被称为 Transformer encoder，其每一个 token 会 attend 到所有的 token。）来构建整个模型，因此最终生成能融合左右上下文信息的深层双向语言表征。经过多层 BERT 的堆叠可以形成以下结构：



BERT 使用了 transformer 模型的 encoder 层来进行特征的提取，采用了预训练+fine-tuning 的训练模式预训练模型是指在大规模文本数据上进行大量无监督训练，学习得到丰富的语言表示。BERT 的输入为每一个 token

对应的表征，并且单词字典是采用 WordPiece 算法来进行构建的。为了完成具体的分类任务，除了单词的 token 之外，作者还在输入的每一个序列开头都插入特定的**分类 token**，该分类 token 对应的最后一个 Transformer 层输出被用来起到聚集整个序列表征信息的作用。将该 token 作为分类器的输入来进行分类。过程如下图所示：



BERT 的预训练任务是通过 MLM 和 NSP 两个阶段进行的。在 MLM 中，输入文本的一部分被随机掩码，模型需要预测这些被掩码的词。在 NSP 中，模型需要判断两个句子是否是原文中的连续句子。BERT 的无监督训练使得它能够学习到丰富的句子级和词级表示，因为它需要理解上下文并进行语言推理。这种能力使得 BERT 成为适配各种 NLP 任务的理想选择。因此我选择 BERT 来进行本次任务。

（以上内容参考知乎和 BERT 论文）

- 线性支持向量机算法(Linear Support Vector Classification)，属于有监督学习，是 SVM 的一个分支。SVC 的实现基于 libsvm。拟合时间至少与样本数成二次方关系，超过数万个样本后可能就不可行了。因此针对本次实验的训练数据的数目，我选择了 LinearSVC。LinearSVC 基于 liblinear 实现的，事实上会惩罚截距，因此在大量的数据上收敛速度会比较高，适合于处理大规模的数据集。

  LinearSVC 的输入是特征矩阵，因此需要用到特征工程来将文本转为成向量特征来供算法和模型使 用。我使用的是 TF-IDF 进行特征提取。TF-idf 的优势是它能够兼顾词语在文本中的频率和在语料库中的普遍程度，从而过滤掉一些无关的常见词，保留一些能够反映文本主题的重要词。但是由于

LinearSVC 的经过调参的最终结果只有 0.78，效果不是很好，因此不再详细介绍。

（3）二分类评测指标

二类分类问题常用的评价指标是准确率(accuracy)精准度（precision）、召回率（recall）、F1-Score，AUC。本次排名的评分标准为 AUC 值。

- **混淆矩阵**

预测值 / 实际值	Positive	Negtive
正	TP	FN
负	FP	TN

- **准确率，精确率，召回率，F1-Score**

$$\text{Accuracy}=\frac{TP+FP}{TP+FP+FN+TN}$$

*在数据的类别不均衡时，不能很好地反映性能

$$\text{Precison}=\frac{TP}{TP+FP}$$

$$\text{Recall}=\frac{TP}{TP+FN}$$

*精准率和召回率是此消彼长的，即精准率高了，召回率就下降，在一些场景下要兼顾精准率和召回率，就有 F1-score——精确率和召回率的兼顾指标。

$$\text{F1}=\frac{2*presion*recall}{presion+recall}$$

- **ROC**

横坐标：$\text{FPR}=\frac{FP}{FP+TN}$

纵坐标：$\text{TPR}=\frac{FP}{FP+TN}$

修改二分类的阈值得到的（FDR,TDR）的散点组成的曲线为 ROC 曲线。

- **AUC**

ROC 曲线下面的面积大小，通常为（0.5，1）

三、实验过程

(1)读取数据以及数据集分析

为了文本的数据更加可视化，我使用了 matplotlib 等工具在 Jupyter Notebook 上进行了数据集的分析。

- 用 pandas 的 read_csv 函数读取训练集和测试集。

核心代码：

```
import pandas as pd
train = pd.read_csv(r'D:\Python 大作业\toUser\train.news.csv')
test = pd.read_csv(r'D:\Python 大作业\toUser\train.news.csv')
```

- 因为训练集的新闻分布有规律，所以需要进行打乱，并且将新闻重新编号；真假新闻的个数通过图表的方式呈现，为了让数据更加直观，我还调用了 sns 包来设置绘图的颜色，并且进行填补空值等操作。调用 describe()函数对数据进行一个大致的了解，主要是查看缺失值和异常值。

核心代码：

```
import seaborn as sns
import matplotlib.pyplot as plt
#sns 可以设置绘图的颜色和背景等等
sns.set(style='whitegrid', palette='muted', font_scale=1.2)
HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00",
"#FF006D", "#ADFF02", "#8F00FF"]
sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))
#重排顺序
train=train.sample(frac=1).reset_index(drop=True)
#展示文本信息
train.info()
#样本 label 分布
train['label'].value_counts()
sns.countplot(train.label)
plt.xlabel('label counts')
##查看缺失值并且用空值填补
train.isnull().sum()
train['Ofiicial Account Name']=train[‘Ofiicial Account Name’].fillna(‘’)
```

运行结果截图

```
train.head()
```

	Ofiicial Account Name	Title
0	经济学家圈	中国专家与美国代表团的讨论与激辩（全文）
1	唐人街故事	主持人朱军被爆 猥亵女实习生
2	私人打扮师	邓超孙俪早已离婚?只因她,为孩子没有公开消息

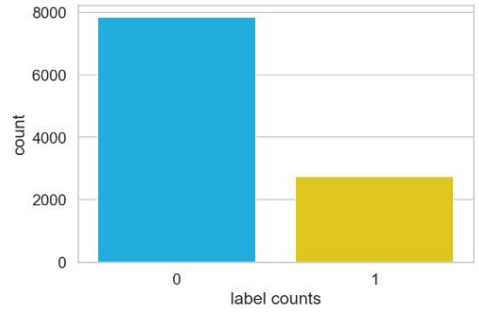
数据集被打乱

```
#展示文本信息
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10587 entries, 0 to 10586
Data columns (total 6 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Ofiicial Account Name  10585 non-null  object
 1   Title                 10587 non-null  object
 2   News Url              10587 non-null  object
 3   Image Url             10587 non-null  object
 4   Report Content        10587 non-null  object
 5   label                 10587 non-null  int64 
dtypes: int64(1), object(5)
memory usage: 496.4+ KB
```

文本信息展示

```
Text(0.5, 0, 'label counts')
```



label 分布的图表

```
0    7844
1    2743
Name: label, dtype: int64
```

label 分布以及数据类型

- 将 csv 文件的内容进行拼接得到完整的内容，用 map 函数以及图表展示文本长度的分布，以此来确定后续 BERT 模型的输入的 max_size。

核心代码：

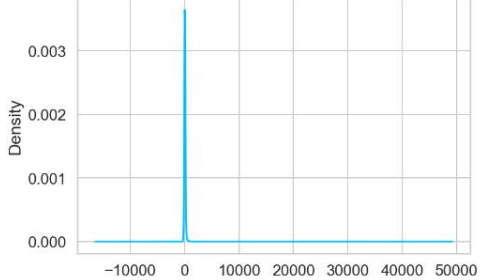
```
train['text']=train['Title']+' '+train['Report Content']
#描述文本长度
train['text_lens'] = train['text'].map(len)
train['text_lens'].describe()
#绘图
train['text_lens'].plot(kind='kde')
```

运行结果截图：

```
count    10587.000000
mean        82.413904
std        633.602287
min         11.000000
25%         37.000000
50%         46.000000
75%         64.000000
max      32802.000000
Name: text_lens, dtype: float64
```

```
train['text_lens'].plot(kind='kde')
```

```
<AxesSubplot:ylabel='Density'>
```



文本长度分布　　文本长度分布图表

可以看到，75%的数据长度在 64 之下，大部分的样本长度都在 128 之下，因此选择 max_size=128 可以覆盖大多数的数据，也就是 padding 的最大长度。

（2）数据处理

- 将数据处理部分的代码储存在 data_process.py 中，数据处理的代码中，定义了一个 fill_paddings 函数，负责将输入的数据补全或截断到 max_len 的长度，保证长度一致；定义了一个 read_data 的函数，用于读取数据并且对数据进行处理。并且打乱顺序，切分训练集和验证集。值得一提的是，数据处理我尝试将 report content 一并加进去，并且进行去除中文停用词，但是，由于效果没有提高甚至会下降，所以我就只用了新闻的 Title 作为输入。

核心代码：

```
def fill_paddings(data, maxlen):
    if len(data) < maxlen:
        pad_len = maxlen-len(data)
        paddings = [0 for _ in range(pad_len)]
        data = torch.tensor(data + paddings)
    else:
        data = torch.tensor(data[:maxlen])
    return data
```

- 接下来我定义了一个 InputDataSet 的类，传入分词器，max_size 和数据，输出三个 Embedding 层，该类可以将文本数据转化为 BERT 模型所需要的输入格式，使用了 tokenizer 的 convert_tokens_to_ids 方法，将 tokens 中的每个词转换为对应的词汇表索引，并赋值给 tokens_ids 列表。在 tokens_ids 的开头和结尾分别添加 101 和 102，表示[CLS]和[SEP]特殊符号。之后调用一个例子来展示其作用。

核心代码：

```
class InputDataSet():
    def __init__(self,data,tokenizer,max_len):
        self.data = data
        self.tokenizer = tokenizer
```

```
        self.max_len = max_len
    def __len__(self,):
        return len(self.data)
    def __getitem__(self, item):    # item 是索引 用来取数据
        text = str(self.data['text'][item])
        labels = self.data['label'][item]
        labels = torch.tensor(labels, dtype=torch.long)
        #分词
        tokens = self.tokenizer.tokenize(text)
        tokens_ids = self.tokenizer.convert_tokens_to_ids(tokens)
        tokens_ids = [101] + tokens_ids + [102]
        #将分好词后的序列补充到最大长度
        input_ids = fill_paddings(tokens_ids,self.max_len)
        #构建注意力掩码序列
        attention_mask = [1 for _ in range(len(tokens_ids))]
        attention_mask = fill_paddings(attention_mask,self.max_len)
        #用来表示输入序列中的不同句子的边界
        token_type_ids = [0 for _ in range(len(tokens_ids))]
        token_type_ids = fill_paddings(token_type_ids,self.max_len)
        return { 'text':text,'input_ids':input_ids, 'attention_mask':attention_mask,
'token_type_ids':token_type_ids,'labels':labels}
```

运行截图

```
   1469, 3152, 4995, 6820],
  [ 101, 5799, 1520,  701, 2767, 1548, 1744, 3625, 6158, 2196, 8013, 2772,
   2199,  967, 2157, 5782],
  [ 101, 1093, 3333, 2787, 1366, 4638,  782, 2621, 1599,  749, 8013, 3124,
   5032, 3341,  749, 8024],
  [ 101, 3173,  740, 2356, 1062, 2128, 2229, 2196,  687, 1344,  769, 6858,
   6356, 2175, 7339, 6824]]), 'attention_mask': tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'token_type_ids': tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])}
```

可以看到数据处理如上图所示

（3）模型构建

至此，我们已经成功构建了一个类来生成模型输入数据。现在使用具有 12 层 Transformer 编码器的预训练 BERT 基础模型构建实际模型，代码储存在 modeling.py 中。命名的 pooled 包含 **[CLS]** token 的 Embedding 向量作为分类器的输入，然后输出 logits 和 loss,用 argmax 得出预测值。我用一个简单的例子展示了代码的输出。

核心代码：

```
class BertForSeq(BertPreTrainedModel):
    def __init__(self,config):
        super(BertForSeq,self).__init__(config)
        self.config = BertConfig(config)
        self.num_labels = 2                    #二分类任务，因此 label 的种类为 2
        self.bert = BertModel(config) #加载这个配置文件，获得 bert 模型
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, self.num_labels)
        self.init_weights()
    def forward(
            self,
            input_ids,
            attention_mask = None,
            token_type_ids = None,    #输入的 embadding
            labels = None,
            return_dict = None
    ):#要加 label
        return_dict = return_dict if return_dict is not None else s
        elf.config.use_return_dict #false 输出 tensor  不设置输出 string
        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            return_dict=return_dict
```

```
        )  #调用 bert 模型把 embedding 传进去，出预测值
        pooled_output = outputs[1]
        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        loss = None
        if labels is not None:
            loss_fct = nn.CrossEntropyLoss()       #交叉熵损失,用到了 softmax
            loss = loss_fct(logits.view(-1, self.num_labels), labels.view(-1))
        if not return_dict:
            output = (logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output
        return SequenceClassifierOutput(
            loss=loss,   #损失
            logits=logits,   #softmax 层的输入，可以理解为是个概率
            hidden_states=outputs.hidden_states,
                attentions=outputs.attentions,
        )
```

运行截图：

```
        [-0.1247,  0.8102],
        [-0.3378,  0.7476],
        [-0.0121,  0.6861],
        [-0.1283,  0.8314],
        [ 0.0979,  0.6744],
        [-0.2811,  0.3070],
        [-0.1595,  0.3312],
        [-0.2173,  0.2168]], device='cuda:0', grad_fn=<AddmmBackward0>)
loss= 0.8136587738990784
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1], device='cuda:0')
```

模型的 MODEL 部分输出

至此模型构建完毕

（4）模型训练

- 接下来需要进行模型训练，使用标准的 PyTorch 训练循环来训练模型。并且输出训练集的 loss，验证集的 acc,loss 来确定模型的训练程度。使用 Adam

为优化器，用 cuda（GPU）来进行训练提升训练的速度。具体代码储存在 train_and_eval.py 中。将训练好的模型储存在 model_stu.bin 之中

- 核心代码：

```
def train(batch_size,EPOCHS):#每一组有多少个句子进行训练和训练轮数
    best_acc = 0
    model = BertForSeq.from_pretrained('bert-base-chinese')
    train = read_data('data/train.csv')
    val = read_data('data/dev.csv')
    tokenizer = BertTokenizer.from_pretrained('bert-base-chinese')
    train_dataset = InputDataSet(train, tokenizer, 128)
    val_dataset = InputDataSet(val, tokenizer, 128)
    train_dataloader = DataLoader(train_dataset,batch_size)
    val_dataloader = DataLoader(val_dataset,batch_size)
    optimizer = AdamW(model.parameters(), lr=2e-5)  #bert 官方优化器
    total_steps = len(train_dataloader) * EPOCHS  # 总步数等于长度乘以轮数
    scheduler = get_linear_schedule_with_warmup(optimizer,num_warmup_steps=0,
num_training_steps=total_steps)
#对训练轮次进行循环
    for epoch in range(EPOCHS):
        total_train_loss = 0
        model.to(device)
        model.train()
        for step, batch in enumerate(train_dataloader):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            token_type_ids = batch['token_type_ids'].to(device)
            labels = batch['labels'].to(device)
            model.zero_grad()#先进行梯度清零
```

```
                outputs =
model(input_ids,attention_mask=attention_mask,token_type_ids=token_type_ids,labe
ls=labels)
                loss = outputs.loss
                total_train_loss += loss.item()# item 才是数值
                loss.backward()
                nn.utils.clip_grad_norm_(model.parameters(), 1.0)#避免梯度爆炸
                optimizer.step()
                scheduler.step()
            avg_train_loss = total_train_loss / len(train_dataloader)#求平均损失
            model.eval()
            #验证集的损失和准确率
            avg_val_loss, avg_val_acc = evaluate(model, val_dataloader)
            if avg_val_acc > best_acc:#保存最好的模型
                best_acc=avg_val_acc
                torch.save(model,'./cache/model_stu.bin2')
                print('Model Saved!')
def evaluate(model,val_dataloader):
    total_val_loss = 0
    corrects = []
    for batch in val_dataloader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        token_type_ids = batch['token_type_ids'].to(device)
        labels = batch['labels'].to(device)
        with torch.no_grad():
            outputs =
model(input_ids,attention_mask=attention_mask,token_type_ids=token_type_ids,labe
ls=labels)
        logits = torch.argmax(outputs.logits,dim=1) #先进行 softmax
```

```
preds = logits.detach().cpu().numpy()
labels_ids = labels.to('cpu').numpy()
corrects.append((preds == labels_ids).mean())
loss = outputs.loss
total_val_loss += loss.item()
    avg_val_loss = total_val_loss / len(val_dataloader)
    avg_val_acc = np.mean(corrects)
    return avg_val_loss, avg_val_acc
```

- 之后调用 train(batch_size,EPOCHS)，这两个参数需要自己根据模型的性能以及运行过程中 loss 和 acc 的变化来进行调整，这会影响的模型性能。

训练过程截图：

```
==> ====Epoch:[8/10] avg_val_loss=0.20043 avg_val_acc=0.96740====
==> ====Validation epoch took: 03:36====
==>
==> ====Epoch:[9/10] avg_train_loss=0.01777====
==> ====Training epoch took: 03:19====
==> Running Validation...
==> ====Epoch:[9/10] avg_val_loss=0.16785 avg_val_acc=0.97358====
==> ====Validation epoch took: 03:35====
==>
```

可以看到训练过程输出了每一轮的 *loss* 和 *acc*

（5）模型输出

1.调用保存好的训练模型来预测测试集的 label，代码保存到 export.py 中。预训练好的模型用 model = torch.load(model_path) 加载，使用测试数据来评估模型在未见数据上的性能和泛化能力。

核心代码：

```
test = read_data('data/test.csv')
test_dataset=InputDataSet2(test,tokenizer=tokenizer,max_len=64)
test_dataloader = DataLoader(test_dataset,batch_size=16,shuffle=False)
```

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') #GPU
model.to(device)
model.eval()
corrects=[]
for batch in test_dataloader:
    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    token_type_ids = batch['token_type_ids'].to(device)
    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask,
        token_type_ids=token_type_ids)
    logits = torch.argmax(outputs.logits, dim=1)
    preds = logits.detach().cpu().numpy()
    for i in preds:
        corrects.append(i)
```

2.将储存模型预测结果的 corrects 输出到提交文件中。

```
pds=pd.read_csv(r'C:\Users\ASUS\Desktop\submit_example.csv')
df=pd.DataFrame({'id':pds['id'],'label':corrects})
df.to_csv(r'C:\Users\ASUS\Desktop\submit_example.csv',index=False,sep=
',')
```

```
D:\python2\sww\Scripts\python.exe D:\2212046wy\export.py
DONE!

Process finished with exit code 0
```

DONE 表示模型输出的结果储存到了提交文件中

至此 BERT 模型实现的展示完毕。

四、实验结果

模型性能评估

（因为本次实验并没有提供测试集的 label，所以只能给出示例代码）

acc,precision,F1-score

- **核心代码：**

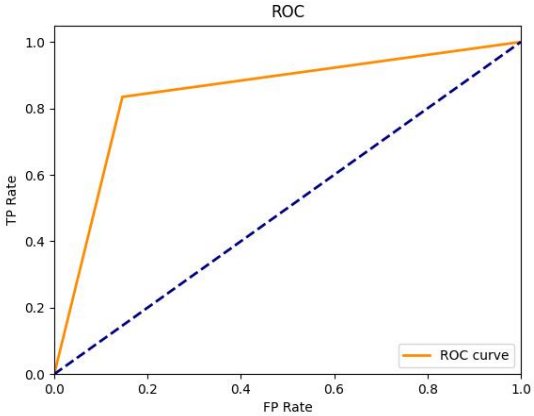
```
from sklearn import metrics
from sklearn.metrics import precision_recall_fscore_support
from sklearn.metrics import accuracy_score
test_dataset=pd.read_csv(answer_path)
print(accuracy_score(test_dataset['label'], corrects))
print(precision_recall_fscore_support(test_dataset['label'], corrects,average='binary'))
```

- **绘制 ROC 曲线**

核心代码：

```
from sklearn.metrics import roc_curve,auc
fpr,tpr,threshold=roc_curve(test_dataset[‘’label],corrects,pos_label=0)
roc_auc=auc(fpr,tpr)
#绘制曲线
import matplotlib.pyplot as plt plt.xlabel('False Positive Rate') plt.ylabel('True
Positive Rate') plt.title('Roc 曲线') plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0]) plt.plot(fpr,tpr,color='r',linestyle='-',linewidth=1.0,label= 'ROC
curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='b', linestyle='--', linewidth=1.0) plt.legend(loc='lower
right',bbox_to_anchor=(0.2,0.95))
plt.show()
```

可能的 ROC 曲线如下：



- **结果输出**

```
auc_value = metrics.auc(fpr, tpr)
print("AUC value=", auc_value)
```

- **评分结果**

在大数据平台提交后模型的分数是 0.87，也就是 auc 值为 0.87，模型的性能良好。

五、实验分析

- 数据处理过程中，通过不断地测试，我发现加入数据的 Report Content 和 Ofiicial Account Name 对模型的性能并没有提升，因为无论新闻是真是假，content 大多都是负面消极的评论，所以我进行了去除并且输入 title，这样模型的性能更好。

  同时，由于数据集不平衡，我又尝试调整不同类别的比例，改变输出的阈值，最后改进了模型。
- 模型训练过程中，因为 Bert 基础模型大概包含 1.1 亿个参数，如果使用 CPU 时间会很慢，这样模型很难进行改进，所以我下载了 torch 1.13.0+cuda11.6，可以通过 CPU 进行运行，大大减少了模型运行的时间和成本。在训练轮数 epoch 的调参中，由于训练过程具有随机性，epoch 的轮数很难确定，所以我通过 val_acc 的比较来输出有最好 val_acc 的模型。由于模型在训练时发生了过拟合的现象，所以我把 Dropout 的参数进行调参来解决过拟合的问题。

- 模型的预测过程中，一开始我没有加入 model.eval()函数，发现每次结果都不同，查阅资料后发现测试时，应该用整个训练好的模型，因此不需要 dropout 来对神经元进行随机置 0，Batch Normalization 在训练时需要用到，但是在预测时会直接拿训练过程中对整个样本空间估算的均值和方差直接来用。故预测时要将模型转化为 eval()模式。

- 模型输出的时候，我发现 Inputdataset 类需要传入 label 的一列，但是 test.csv 中并没有 label 这一列，导致了数据处理出现报错。我通过自定义了一个不用传入 label 的 Inputdataset2 类来读取测试数据。

六、LinearSVC 模型

因为这个模型的效果并不是很好，所以我只简要介绍一下我使用的步骤：

- 根据数据集的 url 进行新闻文本内容的爬取，将标题内容和评论合并在一起让信息更加全面，之后进行数据清洗，使用 jieba 分词去除中文停用词，停用词库从 github 中找到。
- 进行 TD-IDF 特征提取，将文本转化为特征矩阵，并且去除了一些出现频率太高和太低的词，由于特征比较多，我又进行了 svd 降维来去除噪声，防止模型过拟合。同时利用 StandardScaler 标准化工具来标准化训练和测试数据。
- 将处理好的矩阵作为 x 输入，用 label 作为 y 来进行拟合，LinearSVC 的参数通过 GridSearchCV，以 auc 为评分标准，来选出最好的参数。由于数据不平衡，我使用了‘balanced’模式来改进，调整各个参数来防止过拟合等现象。下图是根据 gridsearchcv 得到的最好的参数列表。

```
  warnings.warn(
{'dual': True, 'loss': 'squared_hinge', 'multi_class': 'ovr', 'penalty':
'l2'}
Best score: 0.9040348311233805
0.01
YES
```

- 使用在评分标准上效果最好的参数，clf = grid_search.best_estimator，用这个模型的参数来进行拟合和预测，最后得到了 0.78 的评分。

七、实验感悟

本次 Python 大作业，从一无所知的小白到能够通过查阅资料，自主完成一个机器学习任务，过程是艰难的，但结果是美好。发现问题，解决问题是我在这个过程中不断重复进行的，有时候解决一个问题还需要解决另一个问题，正是在这样的过程我学习到了很多，也收获到了很多，学习能力与信息查找能力得到了很大的提高。同时，我也感受到了 Python 程序语言的强大之处， Python 配备

了大量库和框架供我们使用。在机器学习中，我们不需要用 Python 从头开始整个开发流程，节省了大量的时间。同时，csdn，知乎，github 等网站的利用让我的学习过程更加高效。

“拨云见日终有时,一碧万顷醉晴空”。大作业的完成标志着 python 课程接近尾声，但这门课带给我的影响是深远的，远远没有结束。只要我们有一个正向积极，进取的心，定能克服困难，拨云见日。