

Python 实验报告

虚假新闻检测

学号：2212422

姓名：孙启森

专业：计算机科学与技术

学院：计算机学院

一、实验目的：

给定一个信息的来源、标题、网址、图片以及相关评论，尝试辨别信息真伪
输入信息：信息来源、标题、内容网址、图片、评论
标签（0：消息为真；1：消息为假）

二、实验步骤：

数据获取

从网站中获取训练集和测试集，存到本地文件夹中

数据读取

读取文件中的内容到 python 中，以便于后续应用

数据处理

文本预处理：包括数据清洗、数据分析、特征构建

通过特征构建将原本的数据构建成能体现数据特征并且能应用于模型的输入

模型使用

选择适当的模型，通过特征值利用训练集进行模型训练，并利用训练的模型对测试集进行预测。

三、实验内容

数据读取：通过调用 python 中的 pandas 库读取包含训练集和测试集的 csv 文件，并设置变量接收。

读取后所得 columns 为：'Official Account Name', 'Title', 'News Url', 'Image Url', 'Report Content', 'label'

爬虫：

考虑到要尽可能获取数据的特征，因而通过爬虫爬取网址中的新闻内容，并将内容保存到 News Url 标签中，在此形成一个新的 csv 文件

```
def duqu(url):
    headers={
        "User-Agent":"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/119.0.0.0 Safari/537.36"
    }
    content=requests.get(url,headers=headers).text
    soup=BeautifulSoup(content,"html.parser")
    alltext=soup.find_all("p")
    final=''
    for price in alltext:
        if price.string!=None:
            final=final+price.string.replace('\n', ' ')+ ' '
    if len(final)==0:
        return ' '
    else:
        return final
test_df['News Url']=test_df['News Url'].apply(duqu)
test_df.to_csv('newtest2.csv',index=False,encoding='utf-8')
```

通过 requests 库访问网页获取内容，并通过 BeautifulSoup 解析 html 的内容，观察网页的文字在 p 标签中，获取 p 标签其中的文字，因为有的文本中存在换行符，在用 pandas 读取 csv 文件时会报错，因此将文本中的换行符替换为空格。最终都存入标签中，并形成一个新的 csv 文件 newtrain2。但许多网页已被删除，找不到内容。

数据处理:

本模型采用 TfIdf (“词频” (Term Frequency, 缩写为 TF), 另一层是“逆文档频率” (Inverse Document Frequency, 缩写为 IDF)) 进行特征处理。考虑到特征处理方式与词频与逆文档频率相关, 从而将相加得到一个新的标签 ‘data’, 随后将进行处理, 因为标点与符号在所有类的文章中都含有并且不具有实际意义, 并且还有还有不具有含义的停词, 通过下载网络上的停用词表, 利用 jieba 库将每个文本进行分词并且去除其中的标点符号以及停词, 核心代码如下:

[illegible]

使用 tfidf 进行特征化处理，转换成稀疏矩阵的形式：

```
tfidf_vectorizer = TfidfVectorizer()
```

```
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

通过网格搜索与交叉验证获取最佳的超参数:

```
grid_search = GridSearchCV(MultinomialNB(), param_grid, cv=5)
```

```
best_alpha = grid_search.best_params['alpha']
```

```
nb_classifier = MultinomialNB(alpha=best_alpha)
```

```
y_pred = nb_classifier.predict(X_test_tfidf)
```

```
column=['label']
```

```
y_pred = nb_classifier.predict(X_test_tfidf)
```

```
preds = y_pred
```

```
predictions = []
```

```
for i in preds:
```

```
predictions.append(i)
```

```
print(len(predictions))
```

```
submission = pd.DataFrame({'id': test_df['id'], 'label': predictions})
```

```
submission.to_csv('submit Bayestest.csv', index=False)
```

过程分析：

在训练过程中，还尝试了加入 report content，但准确率反而下降，因为大部分评论都偏向消极，所以影响了准确性。也尝试了加入内容，但大部分文章的内容已被删除，也使准确

率发生了下降。在此情境下，'Official Account Name'，'Title'放在一起的效果较好，且因文本较少，发现不去除停词，去除标点符号的效果较好。

(2) 模型二

`BertForSequenceClassification` 是在 `BertModel` 的基础上，添加了一个线性层 + 激活函数，用于分类。而 Huggingface 提供的预训练模型 `bert-base-uncased` 只包含 `BertModel` 的权重，不包括线性层 + 激活函数的权重。在下面，我们会使用 `model = BertForSequenceClassification.from_pretrained("bert-base-Chinese")` 来加载模型，那么线性层 + 激活函数的权重就会随机初始化。通过微调，学习到线性层 + 激活函数的权重。

数据处理:

使用 bert 的 Tokenizer 对文本进行编码，input_ids 表示每个字的编码，attention_masks 表示需要关注的地方；随后将编码后的数据变成张量并且封装成数据集

```
def dataset(texts, labels=None):
```

```

input_ids = []
attention_masks = []
for text in texts:
    textencode=token.encode_plus(text,add_special_tokens=True,truncation=True,
max_length=50, padding='max_length', return_tensors='pt')
    input_ids.append(textencode['input_ids'])
    attention_masks.append(textencode['attention_mask'])
input_ids=torch.cat(input_ids)
attention_masks=torch.cat(attention_masks)
if labels is not None:
    labels = labels
    return TensorDataset(input_ids, attention_masks, labels)
else:
    return TensorDataset(input_ids, attention_masks)

```

设置最大长度为 50 时截断，不满 50 时用 padding 补全，得到编码，观察结果可知用于补全的 padding 不被 attention

```
(tensor([ 0., 0.]
```

之后使用数据加载器并进行模型训练

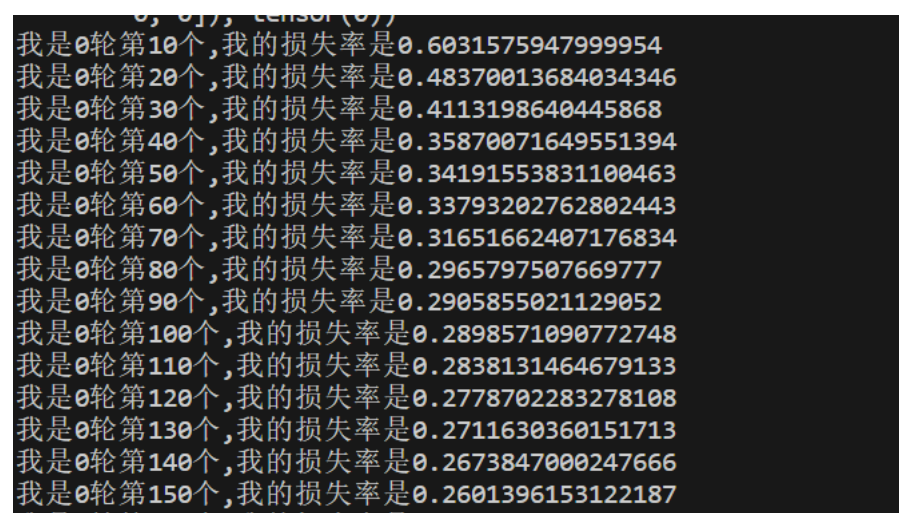
```
traindata_loader=DataLoader(train_dataset, batch_size=batchsize, shuffle=True)
testdata_loader=DataLoader(test_dataset, batch_size=batchsize, shuffle=False)
epochs=2
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
totalstep=0
for i in range(epochs):
    j=0
    totalloss=0.0
    model.train()
```

```

        for batch in traindataloader:
            input_ids, attention_masks, labels=batch
            input_ids, attention_masks, labels = input_ids.to(device),
            attention_masks.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs=model(input_ids, attention_mask=attention_masks,
            labels=labels)
            loss=outputs.loss
            loss.backward()
            optimizer.step()
            totalstep=totalstep+1
            totalloss += loss.item()
            j=j+1
            if totalstep%10==0:
                print(f'我是{i}轮第{j}个,我的损失率是{totalloss/j}')

```

这里使用 AdamW 优化器，通过损失函数与反向传播来进行梯度更新参数，进行多轮训练从而得到训练后的模型。每次训练前进行梯度清零，防止上次造成的影响。



```

我是0轮第10个,我的损失率是0.6031575947999954
我是0轮第20个,我的损失率是0.48370013684034346
我是0轮第30个,我的损失率是0.4113198640445868
我是0轮第40个,我的损失率是0.35870071649551394
我是0轮第50个,我的损失率是0.34191553831100463
我是0轮第60个,我的损失率是0.33793202762802443
我是0轮第70个,我的损失率是0.31651662407176834
我是0轮第80个,我的损失率是0.2965797507669777
我是0轮第90个,我的损失率是0.2905855021129052
我是0轮第100个,我的损失率是0.2898571090772748
我是0轮第110个,我的损失率是0.2838131464679133
我是0轮第120个,我的损失率是0.2778702283278108
我是0轮第130个,我的损失率是0.2711630360151713
我是0轮第140个,我的损失率是0.2673847000247666
我是0轮第150个,我的损失率是0.2601396153122187

```

通过输出损失率，可以看出损失率逐渐下降，训练有效。

```

model.eval()
    predictions = []
    with torch.no_grad():
        num=0
        for batch in testdataloader:
            input_ids, attention_masks=batch
            input_ids, attention_masks = input_ids.to(device),
            attention_masks.to(device)
            outputs = model(input_ids, attention_mask=attention_masks)
            logits = outputs.logits
            predicted_labels = torch.argmax(logits, dim=1)
            predictions.extend(predicted_labels.numpy())
            num=num+1

```



```

        if num%10==0:
            print(f'我已经预测了{num}个了')
print("我好了")
submission = pd.DataFrame({'id': test['id'], 'label': predictions})
submission.to_csv('submit_bert10.csv', index=False)

```

```

我已经预测了500个了
我已经预测了510个了
我已经预测了520个了
我已经预测了530个了
我已经预测了540个了
我已经预测了550个了
我已经预测了560个了
我已经预测了570个了
我已经预测了580个了
我已经预测了590个了
我已经预测了600个了
我已经预测了610个了
我已经预测了620个了
我已经预测了630个了
我好了

```

模型调为评估模式，取消梯度，利用模型进行预测，logits 得到预测的概率，再在行的维度上获取概率高的值的索引，存入预测中并转为 csv 文件。

内容分析：bert 模型所获取的是文本含义，并且根据尝试，发现仅将 title 用于特征处理准确率最高。同时，这里使用的是 bert-base-Chinese 预训练模型，还有 bert-large 的预训练模型，并且能力更强，更改为 large 或许可以进一步提高准确率。

四、模型评估

1 混淆矩阵

在分类任务下，预测结果(Predicted Condition)与正确标记(True Condition)之间存在四种不同的组合，构成混淆矩阵(适用于多分类)

		预测结果	
		正例	假例
真实结果	正例	真正例TP	伪反例FN
	假例	伪正例FP	真反例TN

Accruacy: 比较预测所得结果与测试中的结果，得到准确率

Precision (精确率): $precision = \frac{TP}{TP+FP}$

Recall (召回率): $recall = \frac{TP}{TP+FN}$

F1-score:
$$F1 = \frac{2TP}{2TP + FN + FP} = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

因为没有测试集的标签，在这里将训练集划分为训练集和测试集，从而得到几个数值

```
(10140, 122) 0.2925544679953182  
0.01  
Accuracy: 0.95  
Precision: 0.87  
Recall: 0.93  
f1_score:0.90
```

五、总结

在特征构建时，一开始我只考虑了要尽可能提供更多的信息，因此加入了评论，但准确率反而不如人意，观察发现评论中的字词无论真假都倾向消极。然后我想通过增加内容的方式来进行提升，就去学习了爬虫，从而获取文章内容，但许多文章均已被删除，增加内容后进行特征构建再预测也没有什么提升。然后我直接调用了 python 库中的几个模型如朴素贝叶斯，岭回归，决策树，根据 tfidf 特征直接进行训练预测，结果也都相近。

在之后学习了 bert 模型，刚开始使用的是 title 加 Official Account Name 来进行编码构建特征，但效果不是很理想，然后考虑到得到是句子意思，就只采用了标题，效果较好，之后尝试加入了新闻内容，效果也不是很理想。就更换训练轮数和学习率来进行尝试。

在过程中，掌握了训练和预测的过程，但对于具体的内容则不是很了解，比如机器学习中直接调用了 python 库中的模型进行训练和预测，对于模型本身并不理解。Bert 也是，对于模型整体和原理并不甚了解，怎么使用的过程也只是知道这个地方该使用它。

在刚开始是跟着视频，用普通的预训练模型 BertModel 自己定义了一个增加了一层线性层的下游训练模型，然后在训练中用设置损失函数进行训练。之后了解到 bert 有预训练的用于分类的模型 BertForSequenceClassification，就直接只用了该模型进行训练和预测。在训练和测试过程中，因为是用 cpu 跑的所以速度很慢，时间很长。想尝试用 gpu 跑，但 cuda 页面加载不出来，下载失败，没能使用 gpu 去跑。