

人工智能与机器学习报告

4. 作家风格识别

3220101111 洪晨辉

1. 实验介绍

1.1 实验背景

作家风格是作家在作品中表现出来的独特的审美风貌。通过分析作品的写作风格来识别作者这一研究有很多应用，比如可以帮助人们鉴定某些存在争议的文学作品的作者、判断文章是否剽窃他人作品等。

作者识别其实就是一个文本分类的过程，文本分类就是在给定的分类体系下，根据文本的内容自动地确定文本所关联的类别。写作风格学就是通过统计的方法来分析作者的写作风格，作者的写作风格是其在语言文字表达活动中的个人言语特征，是人格在语言活动中的某种体现。

1.2 实验要求

- a) 建立深度神经网络模型，对一段文本信息进行检测识别出该文本对应的作者。
- b) 绘制深度神经网络模型图、绘制并分析学习曲线。
- c) 用准确率等指标对模型进行评估。

1.3 实验环境

可以使用基于 Python 分词库进行文本分词处理，使用 Numpy 库进行相关数值运算，使用 Keras 等框架建立深度学习模型等。

2. 实验操作

2.1 数据集

包含了五位中国作家的 8438 个经典作品片段，每个片段基本都是一句完整的话。

序号	中文名	英文名	文本片段个数
1	鲁迅	LX	1500 条
2	莫言	MY	2219 条
3	钱钟书	QZS	1419 条
4	王小波	WXB	1300 条
5	张爱玲	ZAL	2000 条

2.2 制作数据集

分词：在做文本挖掘的时候，首先要做的预处理就是分词。

英文单词天然有空格隔开容易按照空格分词，但是也有时候需要把多个单词做为一个分词，比如一些名词如 "New York"，需要做为一个词看待。而中文由于没有空格，分词就是一个需要专门去解决的问题了。本报告遵循原报告格式，利用 jieba 包进行分词，使用精确模式、全模式和搜索引擎模式进行分词对比。

词频：TF-IDF (term frequency - inverse document frequency，词频-逆向文件频率) 是一种用于信息检索与文本挖掘的常用加权技术。

TF-IDF 是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。TF-IDF 的主要思想是：如果某个单词在一篇文章中出现的频率 TF 高，并且在其他文章中很少出现，则认为此词或者短语具有很好的类别区分能力，适合用来分类。

这里我们使用 jieba 中的默认语料库来进行关键词抽取，并展示每位作者前 5 个关键词。

```
LX :
阿Q 0.05379690966906414
没有 0.03501956188388567
一个 0.02659384736489112
知道 0.026370791166196325
什么 0.026117200927953624
MY :
西门 0.04035127611822447
父亲 0.03577176072663162
我们 0.02835442224012238
金龙 0.0274694159504008
一个 0.024059865345607147
QZS :
鸿渐 0.22516872869267315
辛楣 0.12453008658571695
小姐 0.06799326435687081
孙小姐 0.06114419277994029
柔嘉 0.05635906861892125
WXB :
李清 0.05500282755382402
海鹰 0.048308857103309115
但是 0.03985236017697917
后来 0.028965598554340735
假如 0.026102821217101606
ZAL :
太太 0.05627531927572857
露露 0.02639100786806079
一个 0.02441434637731472
没有 0.022718951368287554
自己 0.019893114499557583
```

(不过老实说, 很多都是作者笔下角色的名字)

2.3 模型选用

本报告采用 PyTorch 的模型。主要包含如下步骤:

2.3.1 数据预处理

构建作者列表及计算其数量, 借助字典推导创建字典, 将作者名映射为整数索引, 利于后续数据处理时对作者类别进行数值表示。

```
# 作者列表及相关映射字典初始化
int2author = ['LX', 'MY', 'QZS', 'WXB', 'ZAL']
author_num = len(int2author)
author2int = {author: i for i, author in enumerate(int2author)}

# 数据集初始化及加载
dataset_init = []
data_path = 'dataset/'
for file in os.listdir(data_path):
    if not os.path.isdir(file) and not file.startswith('.'): # 排除隐藏文件和文件夹
        file_path = os.path.join(data_path, file)
        try:
            with open(file_path, 'r', encoding='UTF-8') as f:
                for line in f.readlines():
                    dataset_init.append((line, author2int[file[:-4]]))
        except UnicodeDecodeError:
            print(f"文件 {file_path} 编码错误, 无法读取, 请检查编码。")
```

2.3.2 特征提取

与词频统计一同进行。

```
# 词频统计及特征提取
str_full = [''] * author_num
for sentence, label in dataset_init:
    str_full[label] += sentence

words = set()
for text in str_full:
    words.update(jb.analyse.extract_tags(text, topK=800, withWeight=False))

int2word = list(words)
word_num = len(int2word)
word2int = {word: i for i, word in enumerate(int2word)}

features = torch.zeros((len(dataset_init), word_num))
labels = torch.zeros(len(dataset_init))
for i, (sentence, author_idx) in enumerate(dataset_init):
    feature = torch.zeros(word_num, dtype=torch.float)
    for word in jb.lcut(sentence):
        if word in words:
            feature[word2int[word]] += 1
    if feature.sum():
        feature /= feature.sum()
        features[i] = feature
        labels[i] = author_idx
    else:
        labels[i] = 5 # 无法识别作者标记为 5

dataset = data.TensorDataset(features, labels)
```

2.3.3 模型训练

```
# 数据集划分与加载器创建
valid_split = 0.3
train_size = int((1 - valid_split) * len(dataset))
valid_size = len(dataset) - train_size
train_dataset, test_dataset = torch.utils.data.random_split(dataset, [train_size, valid_size])
train_loader = data.DataLoader(train_dataset, batch_size=32, shuffle=True)
valid_loader = data.DataLoader(test_dataset, batch_size=1000, shuffle=True)

# 模型构建与初始化
model = nn.Sequential(
    nn.Linear(word_num, 512),
    nn.ReLU(),
    nn.Linear(512, 1024),
    nn.ReLU(),
    nn.Linear(1024, 6)
).to(device)

# 损失函数与优化器定义
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)
best_acc = 0
best_model = model.state_dict()
```

这里采用训练数据集和验证集七三分的方式, 较为合理。多轮迭代, 训练集批次训练优化模型参数, 验证集评估准确率, 优则更新最佳模型与准确率, 得优效作者识别模型, 为文本分类任务奠基。

```
# 训练循环
for epoch in range(40):
    valid_acc = 0
    for b_x, b_y in train_loader:
        b_x, b_y = b_x.to(device), b_y.to(device)
        out = model(b_x)
        loss = loss_fn(out, b_y.long())
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        del out
    with torch.no_grad():
        correct_num = 0
        total_num = 0
        for b_x, b_y in valid_loader:
            b_x, b_y = b_x.to(device), b_y.to(device)
            out = model(b_x)
            _, predicted = torch.max(out.data, 1)
            total_num += b_y.size(0)
            correct_num += (predicted == b_y).sum().item()
            del out
        valid_acc = correct_num / total_num
    if valid_acc > best_acc:
        best_acc = valid_acc
        best_model = model.state_dict()
        torch.save({
            'word2int': word2int,
            'int2author': int2author,
            'model': best_model,
        }, 'results/test_model.pth')
    print(f'epoch:{epoch} | valid_acc:{valid_acc:.4f}')
```


2.3.4 参数调整, 得到最好的结果

主要是 topK 的调整, 也即选取的每个作家的惯用词数。经过实验, 大约在 800 个时可以达到最优。

```
epoch:27 | valid_acc:0.9455
epoch:28 | valid_acc:0.9451
epoch:29 | valid_acc:0.9455
epoch:30 | valid_acc:0.9459
epoch:31 | valid_acc:0.9467
epoch:32 | valid_acc:0.9463
epoch:33 | valid_acc:0.9423
epoch:34 | valid_acc:0.9451
epoch:35 | valid_acc:0.9459
epoch:36 | valid_acc:0.9459
epoch:37 | valid_acc:0.9471
epoch:38 | valid_acc:0.9463
epoch:39 | valid_acc:0.9467
```

(训练了四十轮，大概稳定在了 0.946 左右)

3. 结果分析

测试点	状态	时长	结果
		1s	测试完成 一共50个文本，预测正确49个

预测结果令人惊喜，识别对了 49 个样本。这说明了该网络优异的效果。

4. 总结与讨论

Pytorch 架构写成的神经网络虽然比较简单，但是其预测效果比较好。这种优异的预测效果主要源于作家风格识别任务的特点。在这类任务中，模型需要捕捉作家独特的语言使用习惯、句法结构和表达风格等深层特征。此任务的目标聚焦于特定特征的分类，而非复杂的多维度预测，这使得简单的网络架构在这种任务中能够高效发挥作用。