人工智能实验报告

实验6　文本情感分类

陳日康　信息与计算科学　22336049

一、实验概述

用 K-NN 分类器或朴素贝叶斯法（二选一实现）完成文本信息情感分类训练。在给定文本数据集完成文本情感分类训练，在测试集完成测试，并计算准确率。可使用各种提取文本特征的辅助工具（如 OneHotEncoder、TfidfVectorizer等），不可直接调用机器学习库中的分类器算法，文本的特征可以使用 TF 或 TF-IDF。在本次实验中，我选择了朴素贝叶斯法完成实验。

二、算法原理

1. 朴素贝叶斯

做法是根据贝叶斯定理来估计每个类别的后验概率。依据公式为：

$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)} = \frac{p(x|y)p(y)}{\sum_i p(x|y_i)p(y_i)} \propto p(x|y)p(y)$$

然后，对每个情感的后验概率进行对比，选择概率最大的一个情感作为判断结果。

2. TF / TF-IDF

TF（Term Frequency）和 TF-IDF（Term Frequency-Inverse Document Frequency）是用于文本特征提取和向量化的常用算法。

TF（词频）是指在一个文档中某个词语出现的频率，其原理是一个词在文本中出现的次数越多，它对文本的区分度就越高。计算方法：$TF = (\text{词语在文档中出现的次数})/(\text{文档中总词语数})$。

TF-IDF 算法是在 TF 算法的基础上进一步优化的算法。TF-IDF 算法的原理是，它除了考虑词语在文档中的频率，还考虑了词语在整个语料库中的重要性，如果某个词在一个文档中频繁出现，并且在其他文档中出现次数较少，那么它对这个文档的重要性很高。TF-IDF 就是 TF 和 IDF 的乘积，其中 IDF 是逆文档频率，而 IDF 的计算方式为：$IDF = log(\text{语料库中文档总数}/(\text{包含词语的文档数} + 1))$。

通过 TF-IDF 算法，可以将文本数据转换为特征向量，其中每个维度表示一个词语在文档中的重要性。这样可以利用这些特征向量来训练机器学习模型或进行其他文本处理任务。

3. 拉普拉斯平滑

拉普拉斯平滑用于解决在文本分类中出现概率为 0 的问题，在文本分类中，当某个词在某个类别中没有出现时，它的概率就会变成 0，这会导致整个文本的概率为 0，从而影响分类的准确性。在计算概率时，将每个词出现的次数加上一个常数 k，再将分母加上 k 乘以词汇表的大小。这样，即使某个词在某个类别中没有出现，它的概率也不会变成0，而是会有一个较小的概率。

$$p(x_k|d_j,e_i)=\frac{x_k+\lambda}{\sum_{k=1}^{K}x_k+K\lambda}$$

式中 $\lambda \geq 0$。等价于在随机变量各个取值的频数上赋予一个正数 $\lambda \geq 0$。当 $\lambda = 0$ 时就是极大似然估计。当 $\lambda = 1$ 时称为拉普拉斯平滑。

三、流程图

```
开始
|
|---> 读取并处理训练数据 ("train.txt")
|       |
|       |---> 对每个情感类别进行处理
|               |
|               |---> TF-IDF 向量化和拉普拉斯平滑
|               |
|               |---> 数值扩展
|
|---> 读取并处理测试数据 ("test.txt")
|       |
|       |---> 模型评估
|               |
|               |---> 对每条测试数据进行情感预测
|                       |
|                       |---> 计算每个情感类别的得分
|                       |
|                       |---> 选择得分最高的情感作为预测结果
|                       |
|                       |---> 统计正确预测次数
|
|---> 输出结果
|       |
|       |---> 打印准确率和总运行时间
|
结束
```

四、关键代码展示

`read_data(file)` 从文件中读取数据，并将每一行分割成单词列表。

`laplace` 对输入的数组进行拉普拉斯平滑处理。

`extend(res)` 将结果数组中的值扩大一定的倍数，因为特征提取得到的矩阵中每个元素都过小，影响判断结果。

```
def read_data(file):
    with open(file, 'r', encoding='utf-8') as file_object:
        return [line.strip().split() for line in file_object]  # 按行读取并分割成单词列表

def laplace(res, k):
    row_sums = np.sum(res, axis=1) + k * res.shape[1]  # 计算每行的和并加上平滑因子
    return (res + k) / row_sums[:, None]  # 返回平滑后的结果

def extend(res):  # 扩大结果数值范围
    return res * 10000000000
```

`classify_emotions(set)` 使用 TF-IDF 向量化器将文本集合转换为 TF-IDF 表示，并对结果进行平滑处理和扩展，然后返回处理后的结果和特征名称。

```
def classify_emotions(set):  # 分类情感
    tv = TfidfVectorizer(use_idf=True)  # 创建TF-IDF向量化器
    tv_fit = tv.fit_transform(set)  # 将文本集合向量化
    ft_name = tv.get_feature_names_out()  # 获取特征名称
    res = tv_fit.toarray()  # 转换为数组形式
    res = laplace(res, 0.000114514)
    res_norm = extend(res)
    return res_norm, ft_name
```

`judge(num, arr, word_list, word)` 用于判断情感类别，根据每个类别的后验概率来进行评分，并返回最高得分的情感类别。

```
def judge(num, arr, word_list, word):  # 判断情感类别
    res = 0
    for i in range(len(arr[num])):
        mul = 1
        for token in word_list:
            if token in word[num]:
                index = np.where(word[num] == token)[0][0]
                mul *= arr[num][i][index]  # 后验概率公式
        res += mul
    return res
```

`process_data(file_path)` 处理原始数据，将其组织成一个字典，其中键是情感类别，值是对应的句子列表。

```
def process_data(file_path):
    data_list = read_data(file_path)
    data_list.pop(0)
    emotions = {1: [], 2: [], 3: [], 4: [], 5: [], 6: []}  # 初始化情感字典
    for item in data_list:
        sentence = " ".join(item[3:])  # 合并句子部分
        emotions[int(item[1])].append(sentence)  # 按情感类别存储句子
    return emotions
```

`evaluate_model(test_data, arr, word)` 对模型进行评估，计算模型在测试数据上的准确率。

```
def evaluate_model(test_data, arr, word):  # 评估模型
    right_ans = 0
    for i, test in enumerate(test_data):  # 遍历测试数据
        answer = int(test[1])  # 获得正确答案
        word_list = test[3:]  # 获取测试句子词列表
        max_score = float('-inf')  # 初始化最高分数
        judgement = -1  # 初始化判断结果
        for j in range(6):
            score = judge(j, arr, word_list, word)
            if score > max_score:  # 如果得分最高则更新
                max_score = score
                judgement = j + 1
        if judgement == answer:  # 如果判断正确
            right_ans += 1

        print("No.", i + 1)
        print("Judged:", judgement)
        print("Answer:", answer)
        print("Test case:", word_list)
        print()

    return (right_ans / len(test_data)) * 100
```

五、实验结果展示

详细运行结果可见附件 `result`， 正确率为 34.9%，符合预期。

六、参考资料

1. https://blog.csdn.net/chengyuhaomei520/article/details/123370921?ops_request_misc=%257B%2522request%255Fid%2522%253A%252217157003241680022558 2901%2522%252C%2522scm%2522%253A%252220140713.130102334..%2522%257D&request_id=171570032416800225582901&biz_id=0&utm_medium=distribute.pc_search_result.none-task-blog-2~all~sobaiduend~default-2-123370921-null-null.142^v100^pc_search_result_base8&utm_term=%E6%8B%89%E6%99%AE%E6%8B%89%E6%96%AF%E5%B9%B3%E6%BB%91%E7%AE%97%E6%B3%95&spm=1018.2226.3001.4187
2. https://blog.csdn.net/weixin_46178278/article/details/138034104