中山大学计算机学院

人工智能

本科生实验报告

（2022 学年春季学期）

课程名称：Artificial Intelligence

教学班级	信息与计算科学班	专业（方向）	信息与计算科学
学号	21311359	姓名	何凯迪

一、 实验题目

在给定的文本数据集完成文本情感分类训练，在测试集完成测试，计算准确率。

1. 文本的特征可以使用 TF-IDF 或 TF，对 TF 均使用拉普拉斯平滑技巧（可以使用 sklearn 库提取特征）。
2. 利用朴素贝叶斯完成对测试集的分类，并计算准确率。
3. 需要提交简要报告+代码。
4. 本次实验需要提交伪代码。

二、 实验内容

1. 算法原理

TF-IDF：

TF-IDF 是一种用于信息检索与文本挖掘的常用算法，用于衡量一个词对于一个文档集合中某个文档的重要性或者关联程度。

TF（词频）表示某个词在文档中出现的频率，就是某个词在文档中出现的次数除以文档中总词数。如果一个词在文档中出现的次数越多，那么它对应的 TF 值就越高。

IDF（逆文档频率）表示某个词的普遍重要性或常见程度。它的计算是对整个文档集合进行统计，通过文档集合中包含某个词的文档数目的倒数来计算。如果一个词在文档集合中出现的文档数目越少，那么它对应的 IDF 值就越高。

TF-IDF 的计算公式为：TF-IDF = TF * IDF。TF-IDF 值越高，表示该词在文档中的重要性越高，与文档的关联程度也更强。

拉普拉斯平滑：

拉普拉斯平滑的原理是在概率计算中对所有可能事件的计数加上一个正数常数 k，然后再进行概率估计。这样可以避免出现概率为零的情况。

在应用拉普拉斯平滑之前，我们需要有一个事件发生的频数统计。假设我们有一个离散的随机变量，它可能取值为{a1, a2, ..., an}，而且我们已经观察到了这些值发生的频数{c1, c2, ..., cn}。那么拉普拉斯平滑的计算公式如下：P(ai) = (c(ai) + k) / (N + k * V)

其中，P(ai) 是事件 ai 的概率，c(ai) 是事件 ai 的频数，N 是总的观察次数（所有事件频数的总和），V 是可能的事件取值个数，k 是平滑参数（通常设为 1，即加一平滑）。

朴素贝叶斯算法：

朴素贝叶斯算法是一种基于概率和特征条件独立性假设的分类算法。它是以贝叶斯定理为基础，通过计算给定特征的条件下每个类别的后验概率来进行分类。

特征提取：将待分类的样本数据转换为特征向量表示。

计算先验概率：在训练阶段，统计每个类别的样本数目，并计算每个类别的先验概率。

计算条件概率：对于每个特征，计算在给定类别下的条件概率。将整个条件概率分解为特征的条件概率的乘积。

应用贝叶斯定理：对于一个新的待分类样本，根据特征的条件独立性和贝叶斯定理，计算每个类别的后验概率。

选择分类结果：选择具有最高后验概率的类别作为分类结果。

2. 伪代码

TF-IDF：

```
# 创建 TfidfVectorizer 对象
tv = TfidfVectorizer(use_idf=True)
# 定义文本数据
text = []
# 使用 TfidfVectorizer 进行特征提取
tv_fit = tv.fit_transform(text)
# 获取特征词汇表
feature_names = tv.get_feature_names()
# TF-IDF 特征矩阵
tfidf_matrix = tv_fit.toarray()
# 对 TF-IDF 特征矩阵进行归一化
tfidf_normalized = np.array(tfidf_matrix) / np.sum(tfidf_matrix, axis=1, keepdims=True)
```

拉普拉斯平滑：

```
# 初始化 likelihoods 列表为每个特征在每个类别下的条件概率，初始值为 1
likelihoods = [[1] * num_features for _ in range(num_classes)]
# 初始化 total_counts 列表为拉普拉斯平滑中的分母项，初始值 2
total_counts = [2] * num_classes
# 遍历训练样本
for i in range(num_train_samples):
    # 获取当前样本的标签和特征向量
    label = train_labels[i] - 1
    feature_vector = train_features[i]
    # 累加当前样本的特征总计数到对应类别的 total_counts 中
    total_counts[label] += feature_vector.sum()
    # 累加当前样本的特征向量到对应类别的 likelihoods 中
    likelihoods[label] += feature_vector.toarray()[0]
# 计算每个特征在每个类别下的条件概率（取对数）
for i in range(num_classes):
    # 获取当前类别的总计数
    total_count = total_counts[i]
    # 计算当前类别下每个特征的条件概率（取对数）
    likelihoods[i] = [math.log((count + 1) / (total_count + num_features)) for count in
likelihoods[i]]
```

朴素贝叶斯分类器：

```
# 朴素贝叶斯分类算法（包括拉普拉斯平滑）
function NaiveBayes(train_features, train_labels, test_features):
    num_classes = number of unique(train_labels)
    num_train_samples, num_features = shape of train_features

    # 计算每个类别的先验概率
    priors = 大小为 num_classes,初始设为 0
    for label in train_labels:
        priors[label - 1]递增
    priors = priors / num_train_samples
    # 计算每个特征在每个类别下的条件概率
    likelihoods = 大小为 num_classes*num_features), 初始值为 0
    total_counts = 大小为 num_classes, 初始值为 1
    for i in range(num_train_samples):
        label = train_labels[i] - 1
        feature_vector = train_features[i]
        increment total_counts[label] by sum of feature_vector
        increment likelihoods[label] by feature_vector
```

```
    for i in range(num_classes):
        total_count = total_counts[i]
        for j in range(num_features):
            likelihoods[i][j] = log((likelihoods[i][j] + 1) / (total_count +
num_features))
    # 对测试样本进行分类
    test_labels = []
    for i in range(shape of test_features):
        test_vector = test_features[i]
        scores = 大小为 num_classes 的数组
        for j in range(num_classes):
            prior = priors[j]
            likelihood = sum of likelihoods[j] * feature for each feature, likelihoods[j],
and test_vector
            scores[j] = log(prior) + likelihood
        predicted_label = scores 中最大的 + 1
        append predicted_label to test_labels
    return test_labels
```

3. 关键代码展示（带注释）

```
    # 计算 TF-IDF 特征
def calculate_tfidf_features(train_data, test_data):
    train_texts = [text for _, text in train_data]
    test_texts = [text for _, text in test_data]

    # 使用 TfidfVectorizer 计算 TF-IDF 特征
    vectorizer = TfidfVectorizer()
    train_features = vectorizer.fit_transform(train_texts)
    test_features = vectorizer.transform(test_texts)

    return train_features, test_features

# 实现朴素贝叶斯分类器（包括拉普拉斯平滑）
def naive_bayes(train_features, train_labels, test_features):
    num_classes = len(set(train_labels))
    num_train_samples, num_features = train_features.shape
    # 计算每个类别的先验概率
    priors = [0] * num_classes
    for label in train_labels:
        priors[label - 1] += 1
    priors = [count / num_train_samples for count in priors]
    # 计算每个特征在每个类别下的条件概率
    likelihoods = [[1] * num_features for _ in range(num_classes)]
```

```
    total_counts = [1] * num_classes  # 拉普拉斯平滑中的分母项，初始值为 2
    for i in range(num_train_samples):
        label = train_labels[i] - 1
        feature_vector = train_features[i]
        total_counts[label] += feature_vector.sum()
        likelihoods[label] += feature_vector.toarray()[0]
    # 计算每个特征在每个类别下的条件概率（取对数）
    for i in range(num_classes):
        total_count = total_counts[i]
        likelihoods[i] = [math.log((count + 1) / (total_count + num_features)) for count
in likelihoods[i]]
    # 对测试样本进行分类
    test_labels = []
    for i in range(test_features.shape[0]):
        test_vector = test_features[i]
        scores = [math.log(prior) + sum(likelihood * feature for likelihood, feature in
zip(likelihoods[j], test_vector.toarray()[0]))
                  for j, prior in enumerate(priors)]
        predicted_label = scores.index(max(scores)) + 1
        test_labels.append(predicted_label)
    return test_labels
```

4. 创新点&优化（如果有）

尝试使用了百度的停用词表并选择性的对原版停用表稍作更改，后启用停用词表。

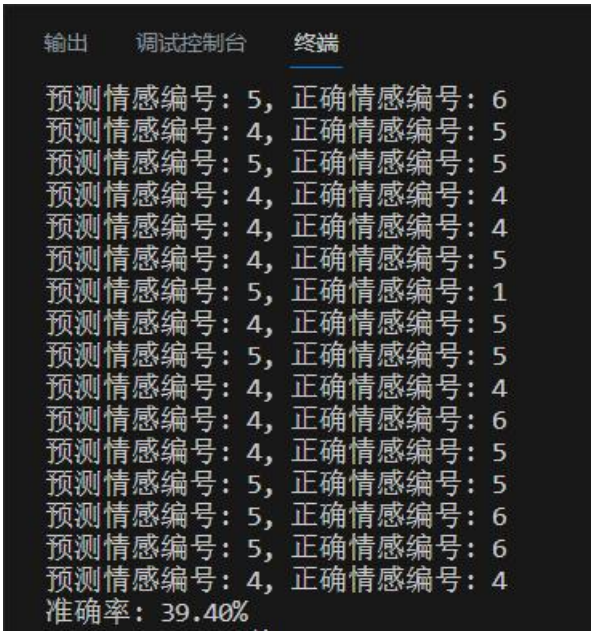
```
# 读取停用词表
def read_stopwords(file_path):
    stopwords = []
    with open(file_path, 'r',encoding="utf-8") as file:
        for line in file:
            word = line.strip()
            stopwords.append(word)
    return stopwords

# 读取停用词表
stopwords = read_stopwords('ban.txt')

# 计算 TF-IDF 特征
train_features, test_features = calculate_tfidf_features(train_data, test_data,
stopwords)
```

（然而，不知为何，我使用停用词表后正确率反而降了 0.6 个百分点...）

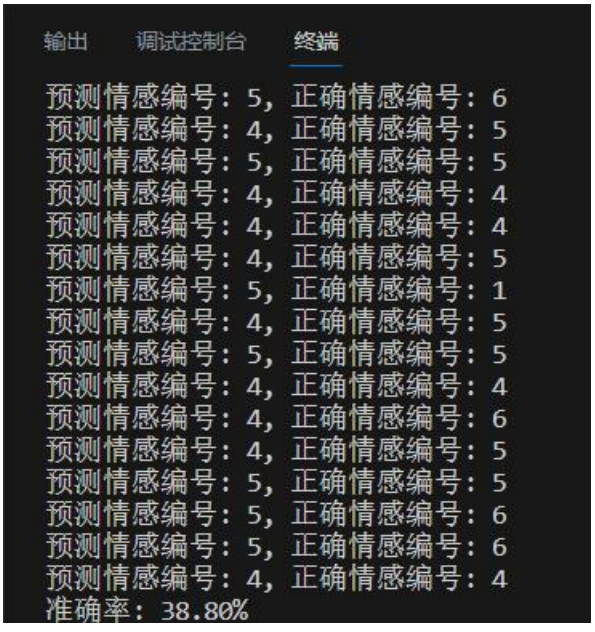
三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

未用停用词表时预测如下：



使用停用词表时预测如下：



四、 参考资料

GitHub - goto456/stopwords: 中文常用停用词表（哈工大停用词表、百度停用词表等）