自然语言处理第四次大作业

孙博一 2020K8009970015

1. 实验目标

请从新浪或其他门户网上收集一定规模的不同类别文档，进行 分类整理，利用不同的特征和不同的分类器实现文本内容分类， 并对不同的方法进行对比实验。

2. 设计思路

使用两种不同的方法对新浪上面的收集的不同类别的文档进行分类
分别使用 IG 和 TF-IDF 进行处理：
具体的实验步骤如下：
2.1 爬取新浪的数据集，并进行分类的标注
2.2 数据预处理：在首先，需要对文档数据进行预处理。这包括文本分词等步骤，以准备好用于特征提取和分类的文本数据。
2.3 特征提取：从文档数据中提取特征。在这里我使用的方法分别是信息增益和特征向量。

1)计算信息增益：使用特征提取得到的向量表示和类别标签，计算每个特征的信息增益。信息增益可以衡量一个特征对于分类任务的重要性，具有较高信息增益的特征可以被认为是更具有分类能力的特征。

2) 构建文档-词频矩阵：使用 TF-IDF 来表示文档的特征向量。首先，将所有文档中的词语组成一个词汇表。然后，对每个文档计算词语在文档中的频率，并乘以逆文档频率，得到 TF-IDF 值。可以使用现有的库（如 scikit-learn）来计算 TF-IDF。

在实际应用中，上面的两种方法是可以同时使用的，其中 TF-IDF 可以作为计算信息增益的基础。在本次实验中，我们呢仅考虑两个模型单独使用的结果比较。
2.4 特征选择：根据信息增益的值对特征进行排序，选择信息增益较高的特征作为用于分类的特征子集。可以根据预先定义的阈值或者选择前 K 个具有最高信息增益的特征。
2.5 分类模型训练：使用选定的特征子集和相应的类别标签训练分类模型。可以选择适合任务的分类算法，如朴素贝叶斯、支持向量机、深度学习等。这里我们使用朴素贝叶斯分类器
2.6 文档分类：使用训练好的分类模型对新的文档进行分类。将文档转换为特征表示，然后使用训练好的模型进行分类预测。

3. 实验流程

3.1.从新浪网上爬取数据，这里我们爬取的类别数据分别是：娱乐、教育、时尚、科技

```
urlent = "https://ent.sina.com.cn/"
urledu = "https://edu.sina.com.cn/"
sports_urls = get_urls(urlsport)
ent_urls = get_urls(urlent)
edu_urls = get_urls(urledu)
fashion_urls = get_urls("https://fashion.sina.com.cn/")
tech_urls = get_urls("https://tech.sina.com.cn/")
```

因为这些数据都是可以使用相似的方法获取的，而其他的数据使用的格式或许并不相同。
我们使用 beautifulsoup 进行文件的爬取工作

```
for text in bes.find_all("a",target="_blank"):
        if not text.get("href"):
            continue
        url1 = text.get("href")
        if url1[-5:] == "shtml" and url1[:5] == "https" and
has_chinese(text.get_text()):
            print(text)
            print(url1)
            urls.append(url1)
```

其中我们特殊关注 target = _blank 并且 herf 中的 html 格式具有如下的形式:

其中最后的五个字符是 shtml，前五个字符是 https，依照这些内容，我们便可以轻松的获取其中的内容了。

我们打开选择出的文件的网页，使用 get_text()获取网页中的信息。

其中部分的信息需要做筛选，因为其中经常出现与新浪有关的标签，这些标签一般出现在网页内容的前面和后面的部分，所以进行筛选的过程是比较轻松的。

筛选完之后，我们将这些文件分别保存在相应的文件夹中的文件中，

```
with open("passage\\edu\\edu%02d.txt"%i, "w", encoding="utf-8") as f:
```

查看文件并观察是否有误

3.2 进行 TF-IDF 的计算

TF（全称 TermFrequency），中文含义词频，简单理解就是关键词出现在网页当中的频次。

IDF（全称 InverseDocumentFrequency），中文含义逆文档频率，简单来说就是该关键词出现在所有文档里面的一种数据集合。

TF-IDF 用来评估字词对于文档集合中某一篇文档的重要程度。TF-IDF 的计算公式为:

TF-IDF = 某文档中某词或字出现的次数/该文档的总字数或总词数 *log（全部文档的个数/（包含该词或字的文档的篇数）+1）

这里我们使用机器学习库 sklearn 的函数进行计算。

首先，生成训练集:

```
def get_corpus():
    corpus = []
    labels = []
    edu_path =
"C:\\Users\\admin\\Desktop\\my_repo\\hw_nlp4\\passage\\edu"
    file_paths = glob.glob(os.path.join(edu_path, '*'))
    file_count = len(file_paths)
    # print(file_paths)
    for file in file_paths:
        with open(file, "r",encoding="utf-8") as f:
            corpus.append(f.read())
            labels.append("edu")
    ……
    return corpus, labels
```

其次，使用 TfidfVectorizer（）函数进行计算。

划分训练集和测试集，最后，使用 bayes 分类器进行估计。
输出最终的结果。

```
corpus, labels = get_corpus()
vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(corpus)

X_train, X_test, y_train, y_test = train_test_split(X, labels,
test_size=0.2, random_state=0)

clf = MultinomialNB().fit(X_train, y_train)

y_pred = clf.predict(X_test)

print(classification_report(y_test, y_pred))
```

同样的，IG 也作为特征提取的一种方法，在下面的代码中使用。
其中最大的不同是，IG 选取了 1000 个特征值作为判断的依据。
同样的，使用贝叶斯分类器进行分类。
最后输出结果。

```
#IG + bayes
vectorizer = CountVectorizer()
X = vectorizer.fit_transform(corpus)

k = 1000  # 选择 K 个特征
selector = SelectKBest(score_func=mutual_info_classif, k=k)
X_selected = selector.fit_transform(X, labels)

X_train, X_test, y_train, y_test = train_test_split(X_selected, labels,
test_size=0.2, random_state=42)

classifier = MultinomialNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print(classification_report(y_test, y_pred))
```

4.结果分析

在本次实验中，进行了如下的两个测试。第一次，我们都拿出类似大小的数据集进行测试（在这里我选用的每个类别的数据大小都是 60 个），进行测试两种方式的性能。
结果如下所示：
Tfidf1 结果：

	precision	recall	f1-score	support
edu	1.00	1.00	1.00	11
ent	1.00	1.00	1.00	15
fashion	1.00	0.92	0.96	13
tech	0.90	1.00	0.95	9
accuracy			0.98	48
macro avg	0.97	0.98	0.98	48
weighted avg	0.98	0.98	0.98	48

lg1 结果:

	precision	recall	f1-score	support
edu	0.92	0.92	0.92	13
ent	1.00	1.00	1.00	11
fashion	1.00	0.92	0.96	12
tech	0.92	1.00	0.96	12
accuracy			0.96	48
macro avg	0.96	0.96	0.96	48
weighted avg	0.96	0.96	0.96	48

我们可以看到，得到的结果都是相对比较优秀的，基本没有什么错误。这就说明了两者在处理相似大小的数据集中，有着相似的性能。
第二次的测试，我们将四个类别中的某一类，只取出 25 个数据，其他的不变，都取出 60 个数据进行训练。这次我们可以发现:
TFIDF 的结果有了一定的偏差，特别是训练集比较少的那一类别中的数据，正确率只有 20%而 IG 依旧表现着比较高的正确率，这是难得的。
Tfidf2

	precision	recall	f1-score	support
edu	0.80	1.00	0.89	12
ent	0.82	1.00	0.90	9
fashion	1.00	0.93	0.97	15
tech	1.00	0.20	0.33	5
accuracy			0.88	41
macro avg	0.90	0.78	0.77	41
weighted avg	0.90	0.88	0.85	41

lg2

	precision	recall	f1-score	support
edu	0.73	1.00	0.84	8
ent	1.00	0.87	0.93	15
fashion	1.00	1.00	1.00	13
tech	1.00	0.80	0.89	5
accuracy			0.93	41
macro avg	0.93	0.92	0.91	41
weighted avg	0.95	0.93	0.93	41

具体的原因上，我认为 IG 更能体现词语之间的重要程度，选择更多与文章相关的词语进行解析，交叉熵能够更好的区分数据。而 TFIDF 只能不是一个不同类别的相互的计算，其中很大一部分依赖于词语的出现程度，在数据集比较少的时候，即便是词语出现次数相对较多，也不能完全肯定能否代表其中的一类数据，所以性能能够稍微差一点。

TF-IDF 假设高频词在整个文档集合中并不一定是重要的，而是需要考虑其在不同文档中的分布情况。而 IG 假设特征与类别之间的关联性是重要的，通过衡量特征在不同类别中的分布情况来选择最有区分性的特征。