Data Mining 实验报告

张浩　计算机技术　专硕班　201834886

实验名称

实验（一）　建立 VSM 及 KNN 分类器的实现

实验要求

预处理文本数据集，并且得到每个文本的 VSM 表示。
实现 KNN 分类器，测试其在 20Newsgroups 上的效果。

实验环境

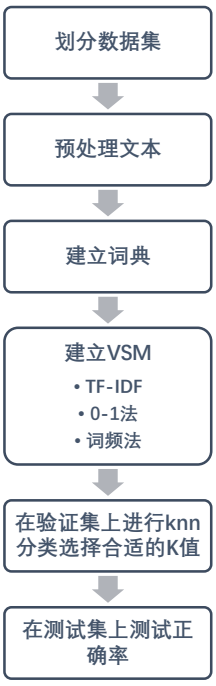
硬件环境:　Intel(R) Core(TM) i5-6260U CPU @1.8GHz 4G RAM
软件环境:　windows 10 1803
编程语言:　python 3.7

实验数据

The 20 Newsgroups dataset is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.
20news-18828.tar.gz - 20 Newsgroups

实验过程

实验流程



划分数据集测试集

将数据集划分为三类

训练集

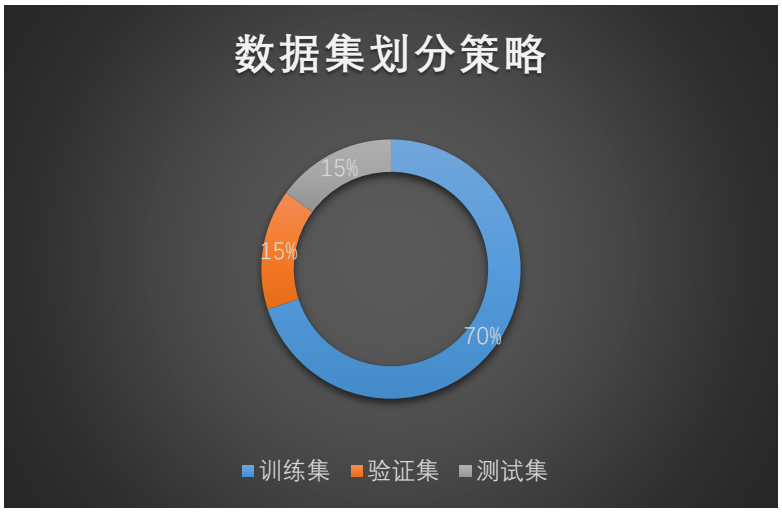
作用：建立 VSM

验证集

作用：调整参数 K 的取值大小，优化 knn 模型

测试集

作用：检验分类器性能



划分数据集时使用其在文件夹名字顺序选取（即选区前 n 个文件）因为文本存放方式与其内容直接不存在直接的关系，所以这样的选择方法可以视为随机选择。

规则化数据集文本

在建立词典以及 VSM 时首先要考虑每个 word 之间尽量不要在意思上重复，所以要提取词干，即单复数、大小写、时态的转化。其次，考虑因素（单复数、大小写、时态、特殊符号）

注意：词干提取时常用方法有

1. **Porter Stemmer 基于 Porter 词干提取算法**
2. **Lancaster Stemmer 基于 Lancaster 词干提取算法**
3. **Snowball Stemmer 基于 Snowball 词干提取算法**

本次实验使用的是 Porter Stemmer 基于 Porter 词干提取算法，在实验时使用不同的算法最终效果也不相同，这个有待进一步实验。

建立词典

建立词典策略的改进

Versions 1.**将所有文档中不在词典的词全部加入词典**（词典中 word 数量太多，约 3w 左右在后续计算过程中会非常慢）

Versions 2.**统计在全部文档中出现次数小于 20 的词**（因为在数据集中单个类最小的文本数量时 628 以 20 为限制条件不会因为数据集过少而 miss 重要信息，而且以 20 为界限可以将词典的数量缩小一半。）

Versions 3.**统计在全部文档中出现次数小于 3 的词**，使用 Versions 2 的统计方法计算速度太慢，跑了两天才跑完。所以这种方法效率很低。如果把范围缩小，只考虑当前文档建立词典的速度会快很多，而且最终词典数量相似。

Versions 4.**统计在全部文档中出现次数小于 3 的词且单词长度大于 2 小于 14 的词**，据观察，之前建立的词典中会有很多垃圾信息（例如 zhanghao122312@gmail.edu.com 经过规则化之后为 zhan122312maileducom 但是这些词都是没有意义的，考虑到英语中常用单词长度约为 3—14 所以这样可以过滤很多垃圾信息）

Versions 5. **单词长度大于 2 小于 14 的词在 df<3 的词且非数字的词**，df 小于 3 说明出现该词的文档数小于 3 说明这个词不太在重要，据观察这类词多数为带数字没有意义的词。

计算数据集建立 VSM

① TF-IDF 方法

根据公式计算 TF-IDF

$$tf(t,d) = \begin{cases} 1 + \log c(t,d), \ if \ c(t,d) > 0 \\ 0, \ otherwise \end{cases}$$

$$IDF(t) = \log(\frac{N}{df(t)})$$

$$w(t,d) = TF(t,d) \times IDF(t)$$

注意：计算中存在一个问题，如果使用公式直接计算 IDF 的话，每个文档要遍历一次词典，每个词要统计在所以文档中出现的情况，即出现改词的文档的数量，时间复杂度约为 $O(n^3)$n 为 2w 左右那么计算量是非常大的，所以需要在循环外计算 DF，而在循环内计算 IDF 时直接查表即可，这样可以在很快的时间内计算完成 TF-IDF

其结果为 float 型，为了加快计算速度，我采用了向上取整策略，但是速度并没有提高多少，准确率还下降了，所以此处还是不取整比较好。

② 0-1 法

出现即为 1、未出现为 0。

③ 词频统计法

以出现的次数为 value 值。

规则化验证集文本

规则化方式同上

建立验证集 VSM

计算方式同上，使用 TF-IDF 计算时，我认为数据集的量足够大时可以用训练集的 DF 值计算测试集的 IDF。所以本实验测试集和验证集 IDF 中的 DF 均使用的训练集的。

使用 knn 输出结果 list

欧氏距离

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

余弦相似度

$$cosine(d_i, d_j) = \frac{V_{d_i}^T V_{d_j}}{|V_{d_i}|_2 \times |V_{d_j}|_2}$$

本次实验使用余弦相似度进行判断

统计实验结果优化 k 的取值

实验 TF-IDF 方法，K 从 15-200（步长为 5）依此计算，并统计正确率
实验 0-1 方法，K 从 15-200（步长为 5）依此计算，并统计正确率

规则化测试集文本

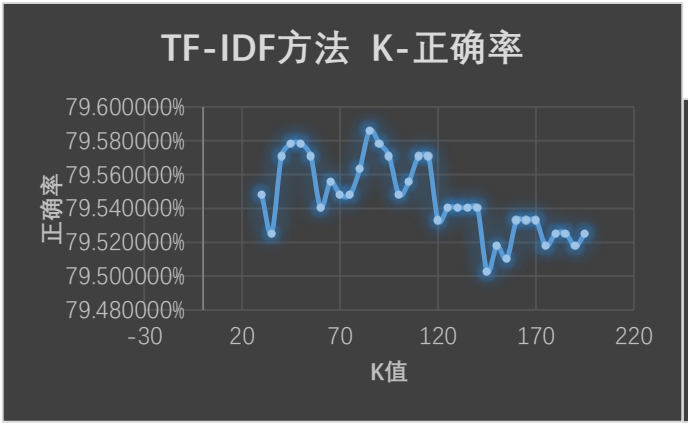
规则化方式同上

建立测试集 VSM

计算方法同上

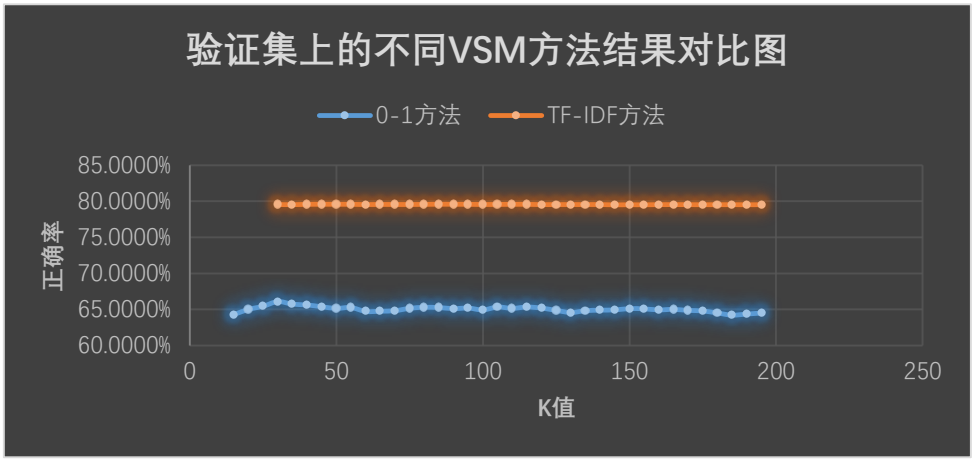
输出 knn 分类器正确率

实验结果分析



验证集结果统计		
	0-1 方法	TF-IDF 方法
平均值	65.0319%	79.5454%
Max 值	66.0866%	79.5860%

测试集上结果		
	0-1 方法（k=30）	TF-IDF 方法（k=85）
正确率	66.47%	79.60%



（实验图中 TF-IDF 值统计了 30-200 的数据）

实验结果来看使用 TF-IDF 方法效果明显好于 0-1 方法，（词频统计方法由于时间关系没来的及做）。使用 TF-IDF 作为向量来衡量文本相似度是比较好的选择，相比于 TF-IDF 空间模型，0-1 型空间模型会 miss 掉许多信息，相比于以出现单词数量作为向量，TF-IDF 考虑到了单词频率对文章的影响。准确率和词典的建立方法有很大关系，为了压缩词典，我将 df 小的直接删除是不妥当的，但是统计词频特别慢，所以实验效果不太好。以及测试集 TF-IDF 的计算可能也是影响因素，在实验中，计算测试集时我使用的时训练集的 DF 值，我认为样本足够大时这个 DF 值应该时可以用训练集的。
实验的不足之处在于词典建立方法比较粗糙，在验证上跑实验时，K 的步长每次增长 5，可能在 k 很小的时候差距很大。但是这样的话从实验结果上是没办法看出来的。

总结

1. 尽量减少 IO 操作，频繁使用的数据常驻内存，在每次处理之后中间数据保存到文件；
2. python 常常内存错误，所以需要及时清理内存中数据，将数据保存或清空；
3. 减少 for 循环中不必要的计算；

附录（主要函数说明）

```
#划分数据集
"""
name:init_set
input:dirs_path    [str]     所有数据根目录
          w [float]       划分为训练集的比例
划分数据集
"""
#       init_set(dirs_path,file_w)
#规则化每个文本
"""
name:dirs_path [str]    根目录路径
遍历所有文件
```

```
input:i [int] 遍历文件数目
"""
Texts,Label =travel_all_file(file_dir_train)

#建立词典
"""
name:texts [list]   所以文件
遍历所有文件构建词典
input:i [int] 遍历文件数目
"""
def travel_all_file_build_dict(texts):
    Dict = vsm.travel_all_file_build_dict(Texts)

#计算 TF-IDF
compute_tf_idf(Dict,Texts,Dict_full)
#规则化测试集
    Texts_test,Label_test = vsm.travel_all_file(file_dir_test)

#计算 test_TF_IDF
compute_tf_idf_test(Dict,Texts_test,Dict_full)

#测试 KNN 性能
judge_dataset(Vectors_TF_IDF, Label,Vectors_TF_IDF_test,Label_test,40)
```