

山东大学计算机科学与技术学院

数据挖掘实验报告

实验题目: Homework 1: VSM and KNN		学号: 201814852
日期: 2018.10.30	班级: 18 级学硕班	姓名: 周培衍
Email: zhoupeiyan123@hotmail.com		
<p>实验目的:</p> <p>1) 预处理文本数据集, 并且得到每个文本的 VSM 表示。</p> <p>2) 实现 KNN 分类器, 测试其在 20Newsgroups 上的效果。</p> <p>3) 20%作为测试数据集, 保证测试数据中各个类的文档均匀分布。</p>		
<p>硬件环境:</p> <p>Inter Core i7-7700 @ 3.60GHz 四核 + Dell 062KRH + 4G + P3-256</p>		
<p>软件环境:</p> <p>Windows 10 + Git + JetBrains PyCharm Community Edition 2018.2.4</p>		
<p>实验步骤与内容:</p> <p>一、 数据集</p> <p>1) 数据集的获取</p> <p>老师提供的 20 个新闻组数据集 (http://qwone.com/~jason/20Newsgroups/), 它已经成为机器学习技术的文本应用实验的流行数据集, 例如文本分类和文本聚类。</p> <p>2) 数据集的划分---divide_data_set.py</p> <p>根据要求将数据集分为训练用和测试用, 而且各个类中的文档要求均匀分布。即将每个类的 80%分为训练数据集, 20%作为测试数据集。</p> <p>问题: 训练数据集和测试数据集都单独作为一个文件夹, 其下是所有文档, 但是这些文件夹中有命名相同的文件, 需要在复制文件时重命名这些文件。</p> <p>我的解决方法是将某个文件夹下的某个文件以“文件夹编号”+“原文件名”命名。例如第五个文件夹(默认字典排序)的 101666 文件, 其文件名为“05_101666”。</p>		

二、 VSM

1) 准备知识

Git-免费、开源的分布式版本控制系统

依据 Git 版本控制,通过新建一个 repository 即为一个项目,通过 pull, add, commit, push, reset 等命令将本地的文件添加到本地仓库和远程仓库,极大的方便了项目版本的管理和代码的回退。

Python-简易的广泛使用的高级编程语言

Python 的设计哲学强调代码的可读性和简洁的语法,它让开发者能够用更少的代码表达想法。使用的编译器是 Pycharm,黑色背景必备。

2) 分词生成词频词典---fenci.py

通过自带的 os, shutil 库中的函数读文件,将所有的文档分词处理,采用的是 jieba 分词工具结合 nltk 停用词表。

预处理:

1. 必须是全英文,有数字或者非法字符的词不要;
 2. 停用词表,包括很多主语,介词,对于分类文本没有实际意义;
 3. 单词长度,通过观察有单个单词(D)和特别长的编码,限制长度[2, 18];
 4. 若文档词频数少于 10,那么这些单词意义不大,且占用词频词典空间。
- 这里有一点需要注意的,词频数删减超参数的选取需要多试试,如果不增加这个限制的话词频词典有 8 万多条,这对于后面求解向量和 knn 分类会麻烦很多,通过试超参数,设置为 10,过滤后的词频词典有 1.5 万条,这个比较可以接受,但是后面的分类的精确度就会相应减少。

词频词典 {"单词": 155, ...} 155 为词频数。并将其写入词典文档。

3) 生成倒排索引词典及词频最大统计---dict.py

根据之前生成的词频词典构建这两个文档。

通过遍历所有的训练文档,先生成每篇文档内容:

```
word_list {"文档名": "["a", "b", ...] ..."} 
```

计数生成每篇文档词频统计:

```
countlist {"文档名": Counter('car': 12, ...), ...} 
```

遍历词频词典,对于每一个词,遍历每篇文档,将每篇文档出现该词的频数存下来,用于后面计算 tf-idf:

```
my_Dict_Inverted {"单词": {"文档名": 155, "文档名": 255 ...}, ...} 
```

对于每篇文档,找出来词频数最大的,并记录,用于计算 tf:

```
file_Max_Number {"文档名": 15, ...} 
```

以上是对于训练数据集,对于测试数据集方法差不多,不过有一点,遍历的词频词典还是之前训练数据生成的,对于测试数据集文档中存在的词频词典中没有的单词就忽略掉,这样就可以将训练数据集和测试数据集的 my_Dict_Inverted 和 file_Max_Number 字典类型分别以 json 存储到磁盘中。这样的好处是求取向量时只需要读取这两个文件就可以了,不需要再读数据集里的文档了。

4) 应用 tf-idf 公式求文本向量---vsm. py

根据倒排索引词典和词频最大统计结合 tf-idf 公式求解文本向量。
tf 公式采用如下定义：

$$tf(t, d) = \alpha + (1 - \alpha) \frac{c(t, d)}{\max_t c(t, d)}$$

其中刚开始设置的 α 为 0.5，但是计算出来的向量差别很小，分类效果很差（不到 50%），之后将 α 设为 0，即为文档中词频数/文档最大词频数。
idf 公式采用如下定义：

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

即 \log (总文档数/包含该词文档数)，因为测试文档中可能某个词都没有而词频词典中有，所以改善为分子和分母都+1，避免分母为 0。即为：

$$IDF(t) = \log\left(\frac{N + 1}{df(t) + 1}\right)$$

在这里还多了一步处理，将向量模长单位化，采用 numpy 库实现，这样子求余弦相似度时就不用除以两个向量的模长积了。计算出来的训练数据集和测试数据集文本向量表示分别存储到本地。

三、 KNN

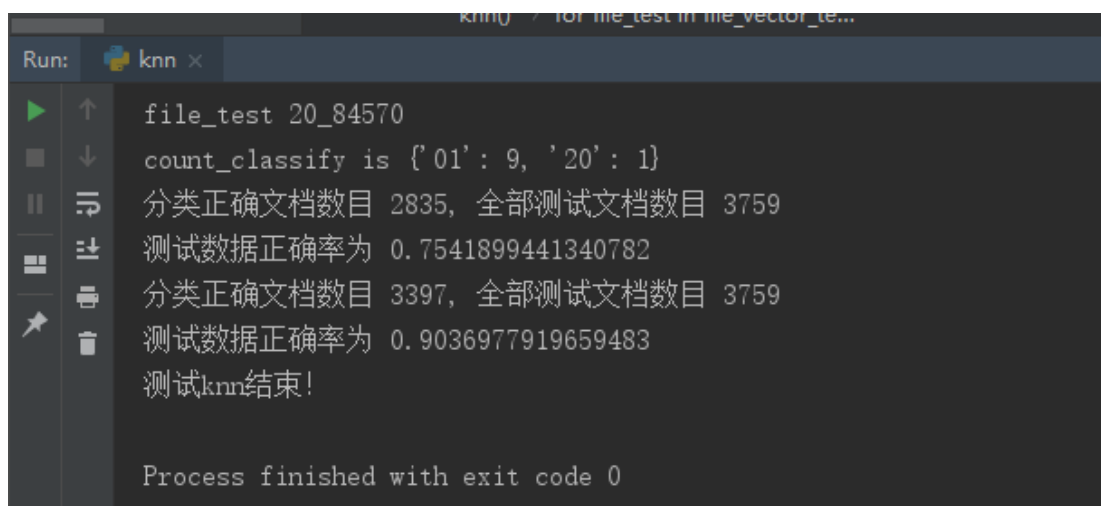
1) 文本向量结合 knn 方法分类---knn. py

加载训练和测试数据集的向量文本表示到内存中，分别将测试的每个文档与训练的所有文档向量依次求 cos 值，优化一点，在每次求出来一个 cos 值后就与最大的 k 个 cos 值比较，存储下来最大的 k 个，时间复杂度从 $O(n \log n)$ 减少至 $O(kn)$ ，最后计算这个长度为 k 的字典计数，找出最多匹配的那一个类别。计算分类正确率。

K=3 时：

```
knn x
file_test 20_84569
count_classify is {'16': 3}
k_list is {'01_53441': 0.3272078634307757, '01_53558': 0.34182016387231}
file_test 20_84570
count_classify is {'01': 3}
分类正确文档数目 2763, 全部测试文档数目 3759
测试数据正确率为 0.7350359138068635
测试knn结束!
```

正确率 73.5%，k=10 时：

A terminal window titled 'Run: knn' showing the execution of a k-NN classification script. The output includes file paths, classification counts for classes '01' and '20', and accuracy calculations for two different test sets. The process ends with 'Process finished with exit code 0'.

```
knn / for file_test in file_vector_te...  
Run: knn ×  
file_test 20_84570  
count_classify is {'01': 9, '20': 1}  
分类正确文档数目 2835, 全部测试文档数目 3759  
测试数据正确率为 0.7541899441340782  
分类正确文档数目 3397, 全部测试文档数目 3759  
测试数据正确率为 0.9036977919659483  
测试knn结束!  
  
Process finished with exit code 0
```

正确率为 75.4%，如果考虑 k=10 时最多分类的三项有一项命中就可以的话，正确率可以达到 90.4%。

结论分析与体会：

通过 homework1 的练习变成体会到 git 的方便，python 代码的简洁，以及很多库的支持使得计算 vsm 和 knn 会方便很多。了解了文本分析和分类的古典的方法，分类正确率还可以有改进的空间，tf-idf 还可以有更好的改进。期待学习并实践新的分类方法。