kNN实验报告-许晓康

2018年10月15日 星期一 下午6:32

第一步，进行数据预处理 dataPreprocessing.py

预处理工具使用的是nltk，全称natural language toolkit 是一个基于python的自然语言处理工具集。

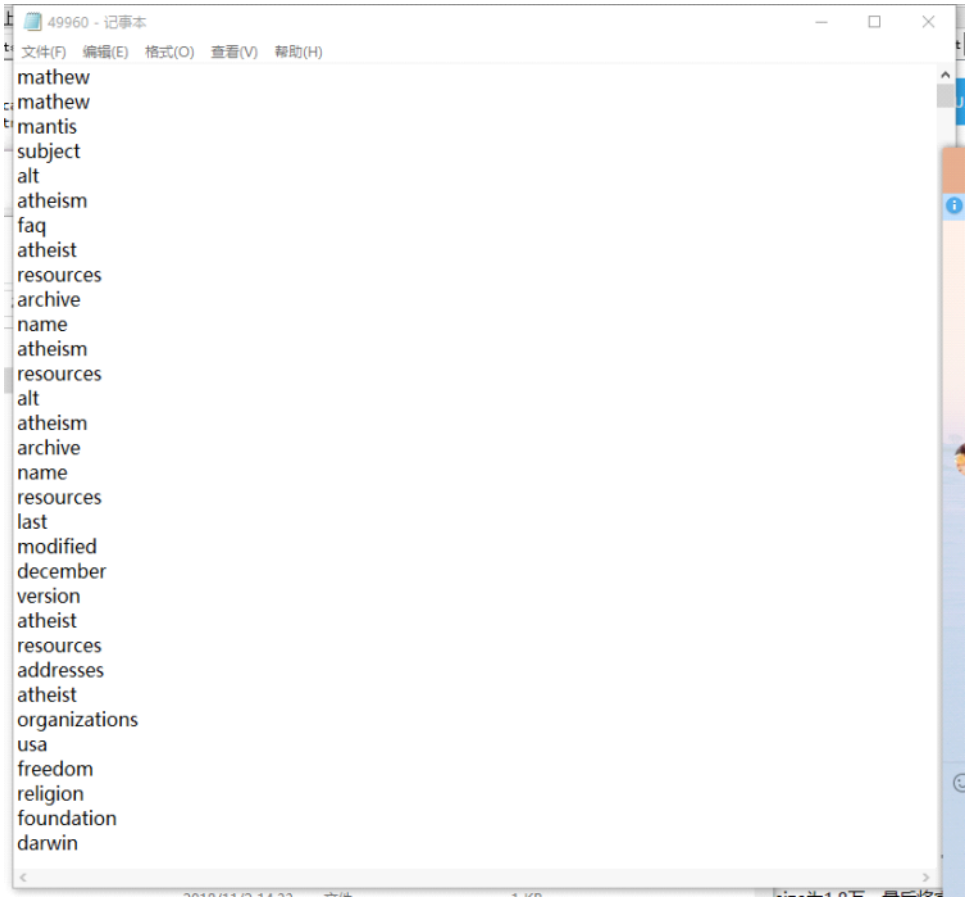
1）使用tokenize(content)方法对句子进行分词

2）去除停用词（停用词使用nltk中的英文停用词）

3）去除单个字母与数字

4）去掉长度小于2的单词

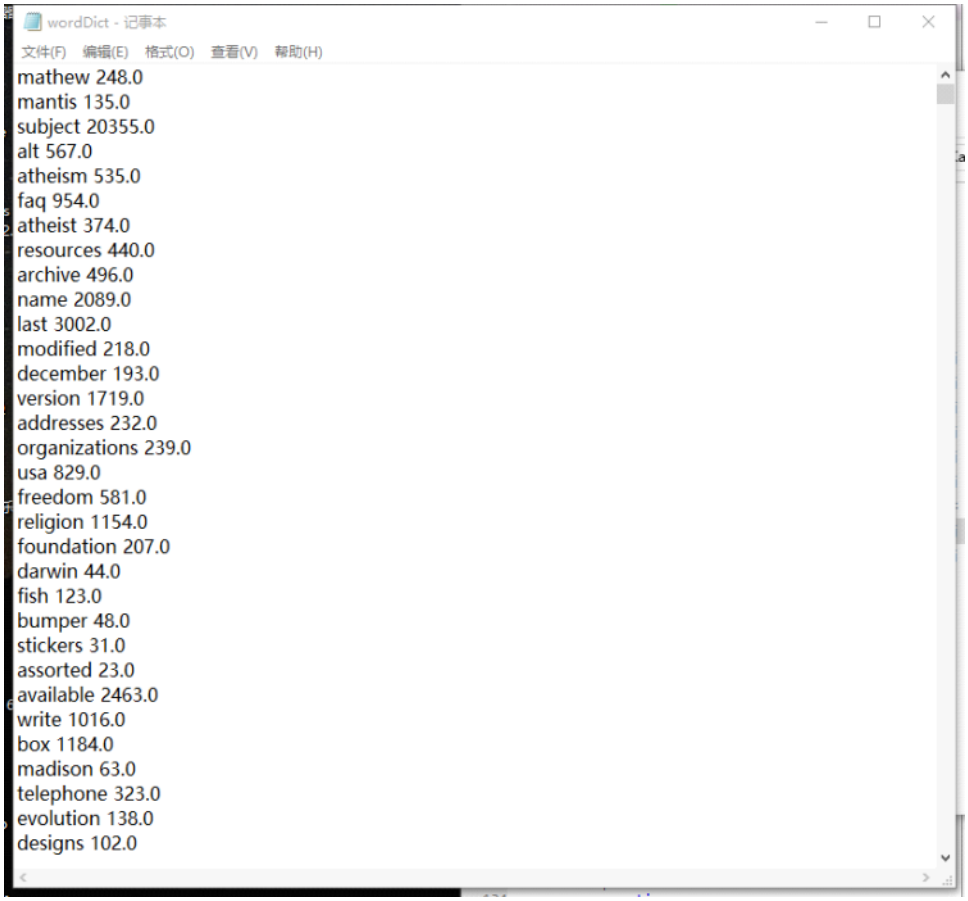
进行数据预处理后的文档如下图所示



第二步，建立字典，过滤文件，划分数据集 partitionData.py

1）方法build_word_map()：

首先对所有文件进行遍历，建立字典，遍历同时统计每个单词出现的频率，设定阈值，将小于13的单词排除，最初设定阈值为5时，字典size大概接近4万，后来调整到13，字典size为1.8万（18725），最后将字典wordDict保存到wordDict.txt文件中，如下图所示



2）方法filter_file()：

对所有文件进行过滤，如果在字典中出现的单词则保留，否则丢弃

3）方法partition_data()：

划分数据集，按照80%训练集 20%测试集的标准，按照文件列表的顺序划分为训练集和测试集

第三步，计算idf，tf*idf computeTFIDF.py

1）方法computeIDF()：

读取所有的文件，在读取过程中记录文档总数目，并且为每个单词建立一个集合set()，统计出现该单词的文档。然后计算每个单词的idf值idf = log(docNum/(wordDocNum+1))/log(10)并保存到文件中

2）方法computeTFIDF():

读取训练集和测试集的文档，在每个文档中，计算每个单词的tf值，与idf值相乘作为单词的权重保存到文件中。所有的训练集保存到一个文本文档中，所有的测试集保存到一个文本文档中，文档格式都相同，其中每一行代表一个文档的数据，格式如下[类名 文档名 单词1 权重 单词2 权重...]，文档之间以换行符区分。

训练集生成的TFIDF文档

测试集生成的TFIDF文档

第四步，knn算法预测分类 knn.py

1）方法compute_similarity(trainData,testData):

计算测试文档和训练文档之间相似性，首先取两个文档共有的单词构成向量，然后计算两个向量之间的cosine值，将cosine值作为两文档之间的相似度返回。

2）方法classify(k,test,trainMap):

根据设定的k值，计算当前测试文档与训练文档集中每一个文档的相似性，然后进行排序取相似度最高的前k个类别用以计算当前测试文档的预测分类，计算方法采用加权计算，权重为相似度，选择加权之后最高的分类作为测试文档的预测分类返回结果。

3）方法test():

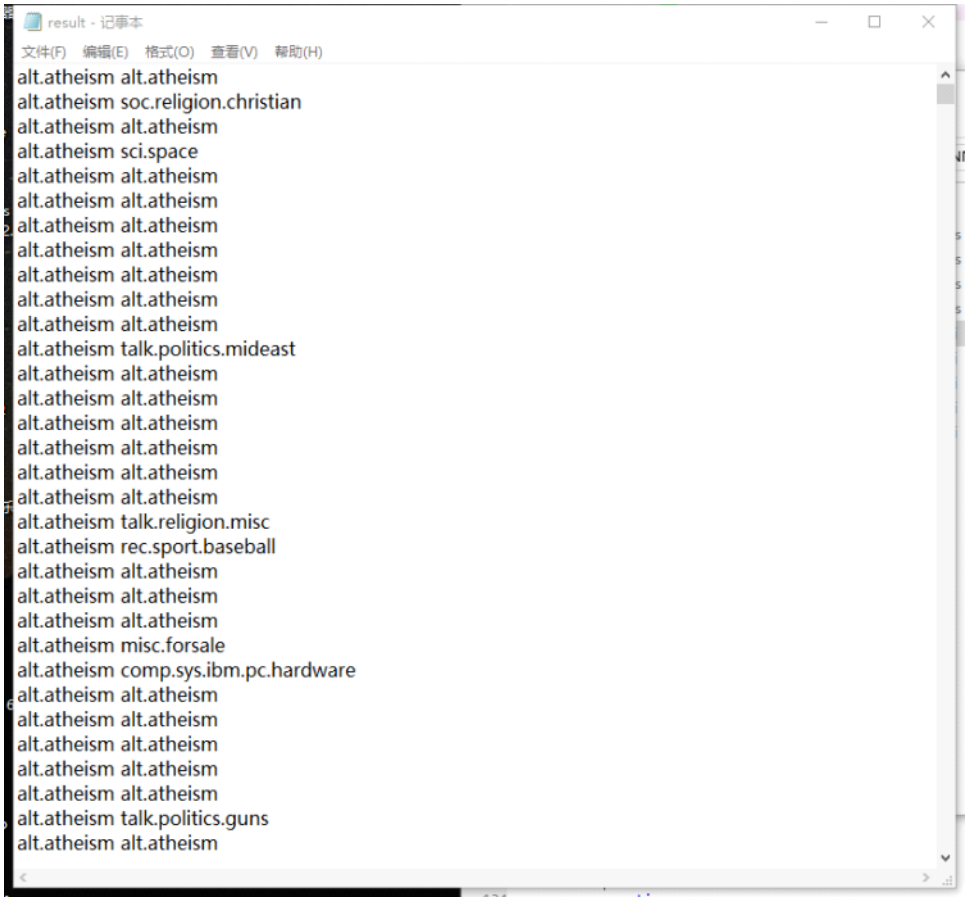
测试knn算法，读取测试集文档和训练集文档，对测试集中的每一条文档调用classfy方法计算文档的预测分类，然后与文档的原始分类相比，统计正确分类的结果，计算knn算法的准确率，并将文档的原始分类 预测分类写入文件中。

4）方法analyse()：

对test()方法分类的结果进行进一步分析，统计不同类别的分类准确率，并写入文件保存。

实验结果及分析：

预测结果记录在result.txt文件中，第一列为文档原始分类，第二列为文档预测分类，如下图所示



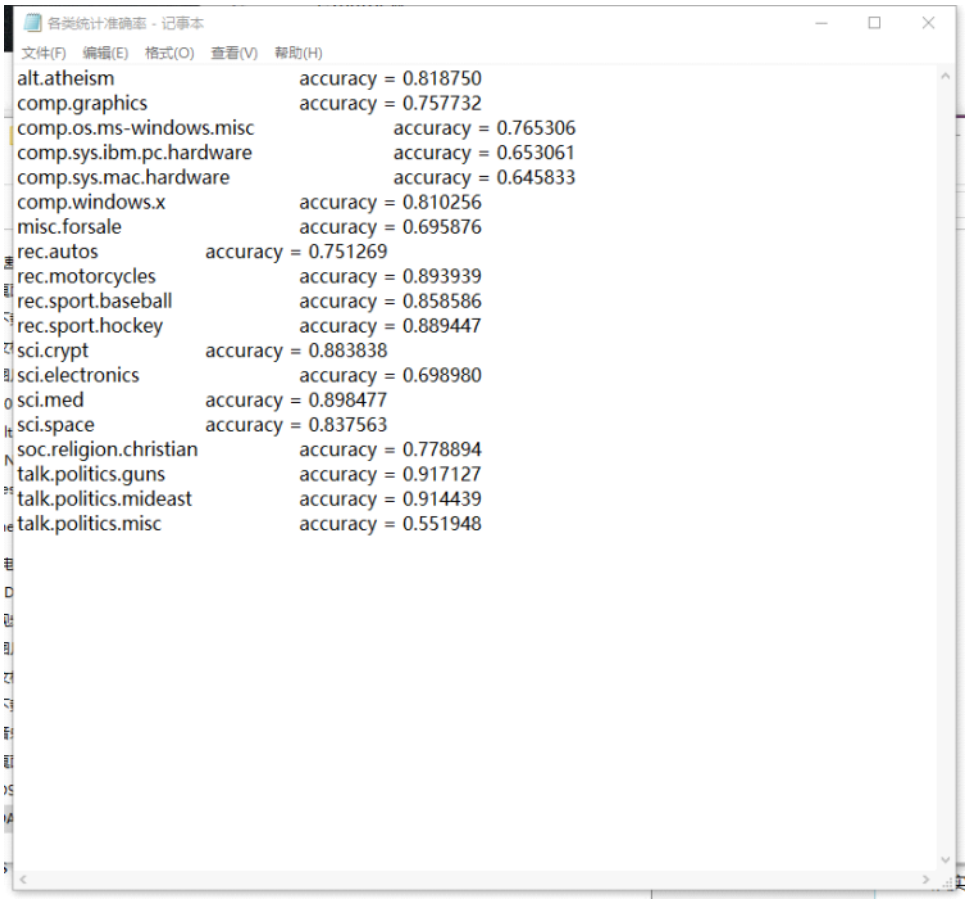
k值进行了多次调整，选取了7/15/20/25/30进行实验，统计不同k取值时的预测准确率，统计结果如下

k取值与准确率

k	acc
7	0.745

15	0.769
20	0.782
25	0.791
30	0.788

从实验取的k值来看在k=25时预测准确率最高，但是也不是很高，所以我对预测结果进行了进一步分析，对每个类的预测结果单独分析，用analyse（）方法基于之前预测的结果计算每个类的预测准确率，保存在 各类统计准确率.txt中，结果如下图所示



可以观察到大多数类中预测准确率比较高，但是其中 misc.forsale sci.electronic talk.politics.misc的准确率偏低，从而拉低了整体的预测准确率。

总结：

通过在20news-18828数据集上实现knn算法，从最初的数据预处理到构建SVM模型，最后实现knn算法，然后改进算法。但仍然存在不足之处，算法的预测准确率不是很高，而且效率低下，预测准确率只有0.791我想进一步要做的是改进数据预处理过程和文档构建向量的过程，如何更好地用一个向量表示当前文档，如何获取到文档的主题。每次执行knn算法耗时比较长，算法效率比较低，可能要改进一下向量表示，提高运算速度。