

**数据仓库与数据挖掘课程实验报告**



**题 目 数据挖掘文本分类实验**

**姓 名 于广华、陈腾**

**学 号 2017140464、2017140463**

**学 院 计算机学院**

**任课教师 王晓茹**

**2017年 11月30日**

# 一 背景介绍

文本分类用电脑对文本集(或其他实体或物件)按照一定的分类体系或标准进行[自动](https://baike.baidu.com/item/%E8%87%AA%E5%8A%A8/9374325)分类标记。本实验从中国新闻网爬取了10类（财经、国际、It、健康、军事、能源、汽车、体育、文化、娱乐）共180多万篇新闻，经过分词、取名词、去掉停用词、计算tfidf降低维度、然后生成分类器的输入数据，采用朴素贝叶斯作为baseline，还用了svm和libsvm分类器来对这100万篇文章进行分类。然后输出分类结果和一些其他评估数据。

# 二 实验步骤



上图为本次试验的处理流程，爬数据，筛选文章，分词，去停用次，取名次，计算每类的tfidf降低维度，筛选分词结果，生成输入数据，训练

## 2.1 构建语料库

本次实验，我们使用的预料库是我们利用python编写爬虫程序在中国新闻网（http://www.chinanews.com/）爬取的数据集。新闻分类共有十类，分别是：财经、国际、互联网、健康、军事、能源、汽车、体育、文化和娱乐。

实验里我们使用了scrapy框架，这个框架可以条理清晰的为我们搭建好爬虫项目，步骤如下：

1）首先打开cmd，输入命令scrapy startproject newsPa 回车创建爬虫项目newsPa；

2）接下来cd /d newsPa，输入命令scrapy genspider –t basic newsPa chinanews.com回车，创建爬虫文件newsPa.py；

3）通过编写getBetweenDay函数，获取从2009年12月至今所有的日期，生成时间列表date\_list；

4）按照每天的日期，根据分析网页信息，确定访问的网址格式：

Total="http://www.chinanews.com/scroll-news/gj/{0}/{1}/news.shtml".format(year,day)



图 网址分析图

5）取得网页里二级网页网址信息，



图 xpath获取图

list = selector.xpath("//div[@class='dd\_bt']/a/@href").extract()

6）依次读取<p>标签里的内容，将其写入文件：

f = codecs.open(filename,'a','utf-8')

for i in range(len(list)-1):

f.write(list[i])

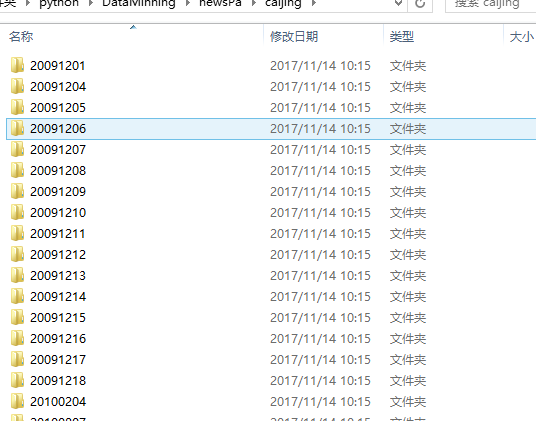
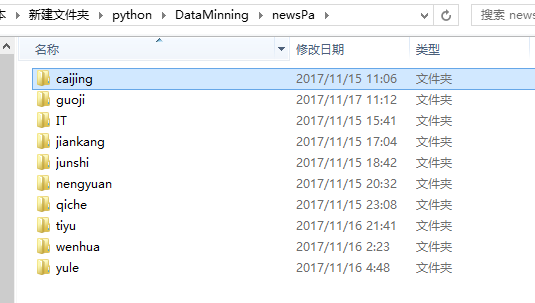
f.close()

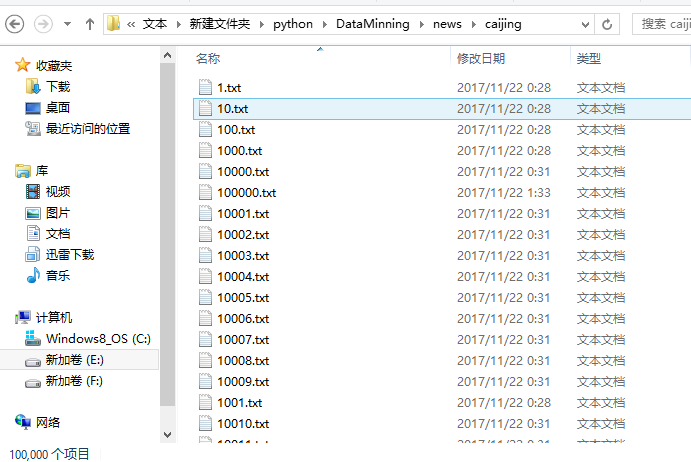
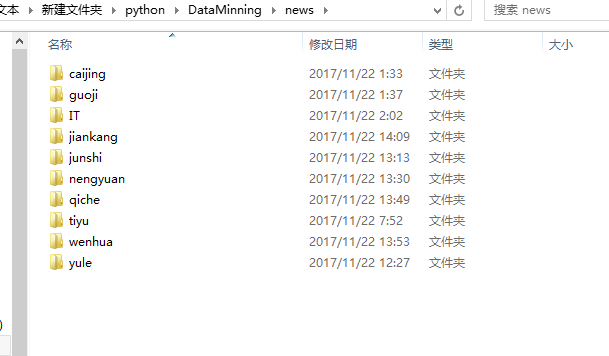
4）输入scrapy crawl newsPa.py命令运行爬虫程序，爬取结果。

## 2.2 数据预处理

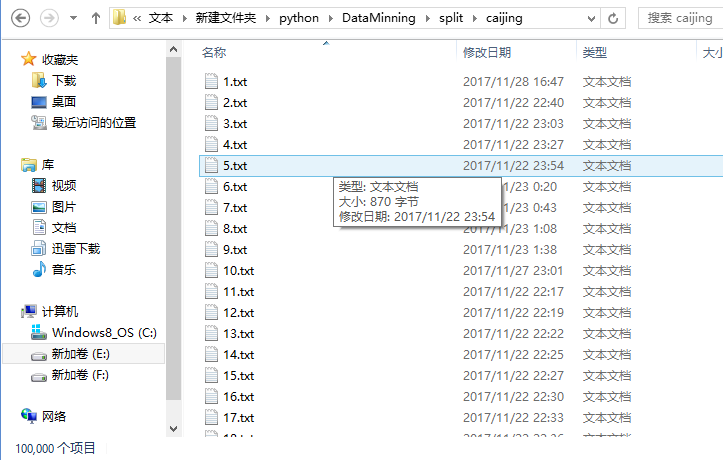
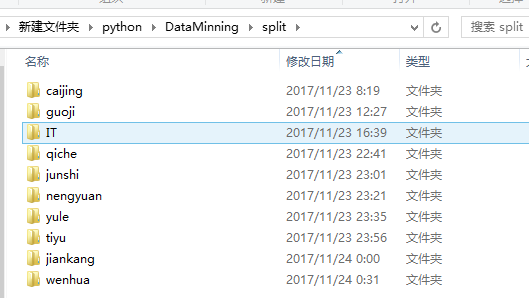
总的数据处理与训练的流程如下：

1.爬下来的文件10类，按照日期生成的文件夹newspa，先去掉字数不符合要求和空文件，挑选出每类10万个放在news里

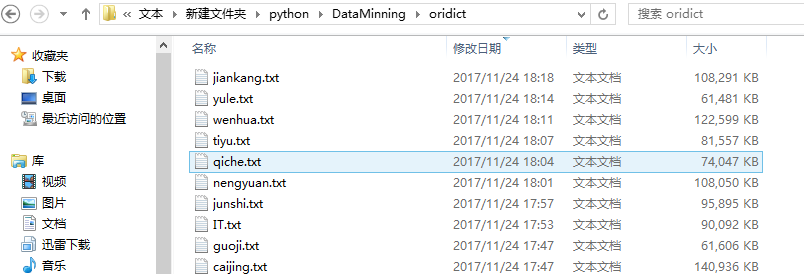




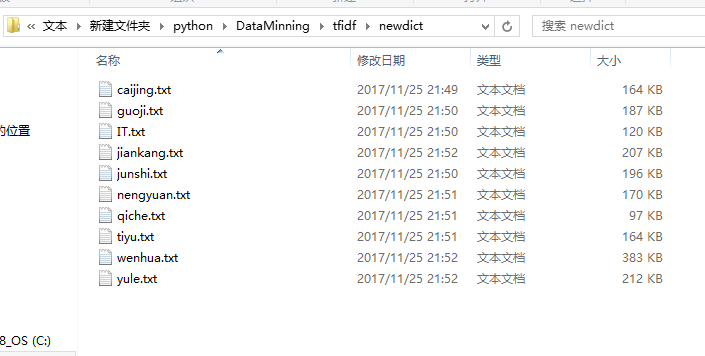
2.从news里读取每类10万个文章并分词，取名词，去停用词放在split文件夹里



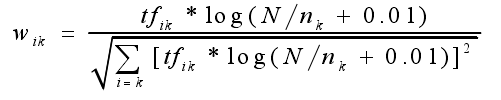
3.把所有类的词语按照类别放在oridict文件夹的一个txt文件里，生成每类的字典，不去重，直接放进去，如图：



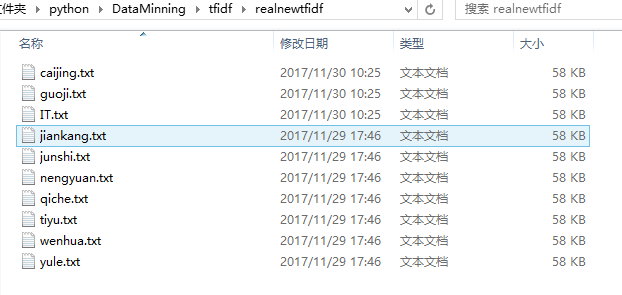
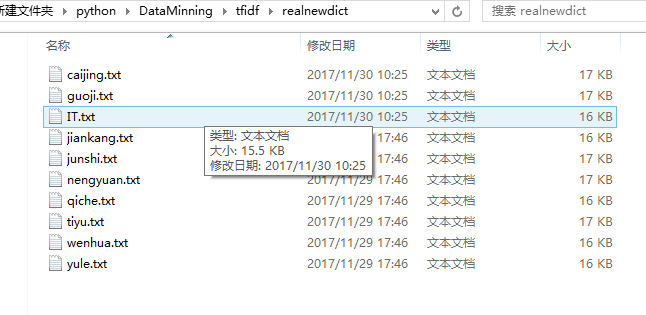
4.把每类的oridict去重并计算每个词语的tfidf，把去重后的词语放在newdict文件夹里，每类的tfidf数值放在newtfidf文件夹里



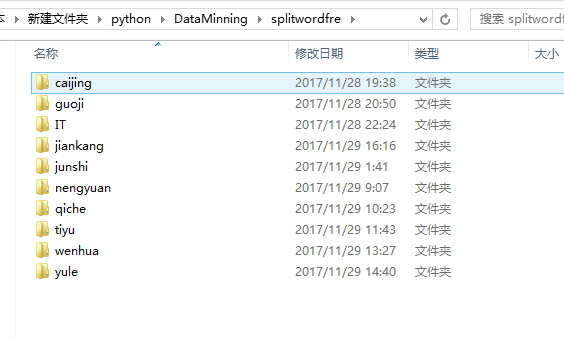
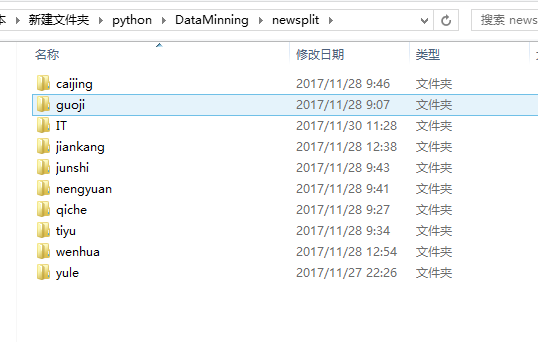
其中，TF-IDF计算公式为：



5.每类的tfidf排序 同时词语跟着换位置，选出每类2000个tfidf数值最大的词语加入总字典，生成dict.txt,每类的选出的2000个词语的内容和tfidf数值放在realnewdict和realnewtfidf



6.根据总字典的内容从新处理每篇文章，对其降低维度，从split文件夹里读取每篇文章的分词结果，和字典里的内容对比，去掉不在字典里的 生成newsplit文件夹（去重）,和splitwordfre（不去重）



7.生成输入文件，朴素贝叶斯直接可以直接输入每篇的分词结果txt，输入未去重的splitwordfre文件精度高一些，而且已经降低纬度。

Libsvm生成专门的格式，需要读取每篇的分词结果，把它映射成总字典的维度，词语需要和字典顺序对应，然后把索引改成序号。

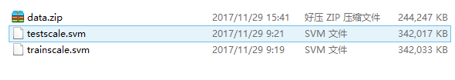
如下格式，最左面表示文章，后面是词语序号和它的tfidf值，训练集不需要类别标号

1 1:-0.963808 2:0.906788 ... 19:-0.197706 20:-0.928853 21:-1

1 1:-0.885128 2:0.768219 ... 19:-0.452573 20:-0.980591 21:-1

... ... ...

1 1:-0.847359 2:0.485921 ... 19:-0.541457 20:-0.989077 21:-1



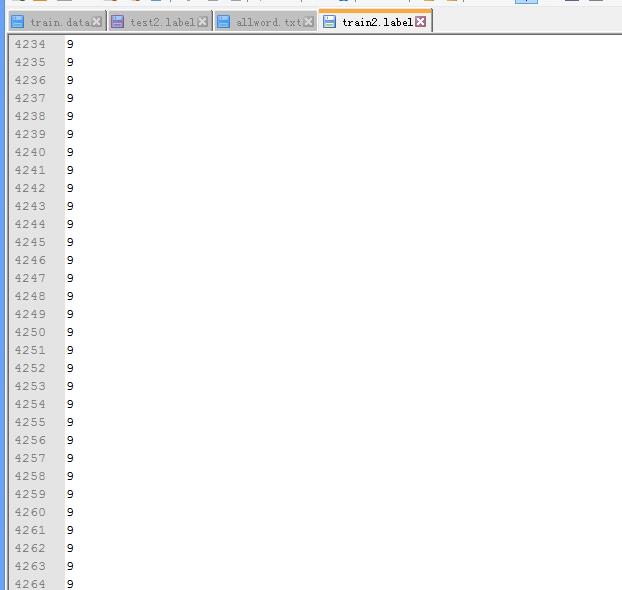
Svm需要生成6个文件，train.data test.data train.label test.label train.map test.map

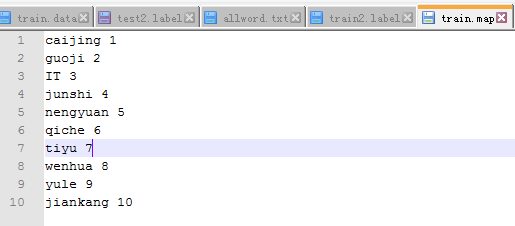
.data 文件每行第一个数字是文章序号 后面是该篇文章的一个词语和他的词频

意思就是每篇文章占用很多行

.label文件是每篇文章的类别，每行对应一篇文章

.map 文件是label序号对应的类别名称





## 2.3 朴素贝叶斯文本分类器

本次实验我们使用的是sklearn里的多项式朴素贝叶斯（MultinomialNB）：

sklearn.naive\_bayes.MultinomialNB(alpha=1.0, fit\_prior=True, class\_prior=None)

因为其主要用于离散特征分类，例如文本分类单词统计，以出现的次数作为特征值，所以很适合用于新闻文本的分类

参数说明：alpha：浮点型，可选项，默认1.0，添加拉普拉修/Lidstone平滑参数；fit\_prior：布尔型，可选项，默认True，表示是否学习先验概率，参数为False表示所有类标记具有相同的先验概率；class\_prior：类似数组，数组大小为(n\_classes,)，默认None，类先验概率。

训练模型：使用

fit(x, y, sample\_weight=None)：根据x、y训练模型

clf = MultinomialNB(alpha=2.0,fit\_prior=True)

clf.fit(x,y)

其中下x和y分别是训练列表和测试列表。

然后根据predict\_proba(X)函数输出测试样本划分到各个类别的概率值，如下：

classifier.predict\_proba(test\_feature\_list)

最后根据score(X, y, sample\_weight=None)：输出对测试样本的预测准确率的平均值，代码如下：

test\_all\_accuracy = classifier.score(test\_feature\_list, test\_class\_list)

## 2.4 SVM文本分类器

SVM 文本分类算法主要分四个步骤：文本特征提取、文本特征表示、归一化处理和文本分类。经过文本预处理、特征提取、特征表示、归一化处理后，已经把原来的文本信息抽象成一个向量化的样本集，然后把此样本集与训练好的模板文件进行相似度计算，若不属于该类别，则与其他类别的模板文件进行计算，直到分进相应的类别，这就是SVM 模型的文本分类方式。

首先将数据载入：

TF, doc2term, term2doc, cate2docs, label = loadOriginData()

然后进行特征的选择：

features, DF = featureSel(doc2term, term2doc, cate2docs)

然后建立SVM模型，为训练做准备：

x, y = buildSVMData(TF, DF, features, len(doc2term), label, cate2docs, doc2term)

然后训练并测试数据：

prob = problem(y, x)

param = parameter('-s 0 -c 4 -B 1')

# 训练

m = train(prob, param)

# 预测test.data

p\_label, p\_acc, p\_vals = predict(y\_test, x\_test, m, '-b 1')

## 2.5 libsvm文本分类器

libSVM的数据格式

Label 1:value 2:value ….

Label：是类别的标识，比如上节train.model中提到的1 -1，你可以自己随意定，比如-10，0，15。当然，如果是回归，这是目标值，就要实事求是了。

Value：就是要训练的数据，从分类的角度来说就是特征值，数据之间用空格隔开

比如: -15 1:0.708 2:1056 3:-0.3333

需要注意的是，如果特征值为0，特征冒号前面的(**序号**)可以不连续。如：

-15 1:0.708 3:-0.3333

表明第2个特征值为0，从编程的角度来说，这样做可以减少内存的使用，并提高做矩阵内积时的运算速度。我们平时在matlab中产生的数据都是没有**序号**的常规矩阵，所以为了方便最好编一个程序进行转化。

y, x = svm\_read\_problem(r'E:\Document-Classification\data\trainscale.svm')

读入训练数据，返回分类标签和数据

yt, xt = svm\_read\_problem(r'E:\Document-Classification\data\testscale.svm')

读入测试数据，

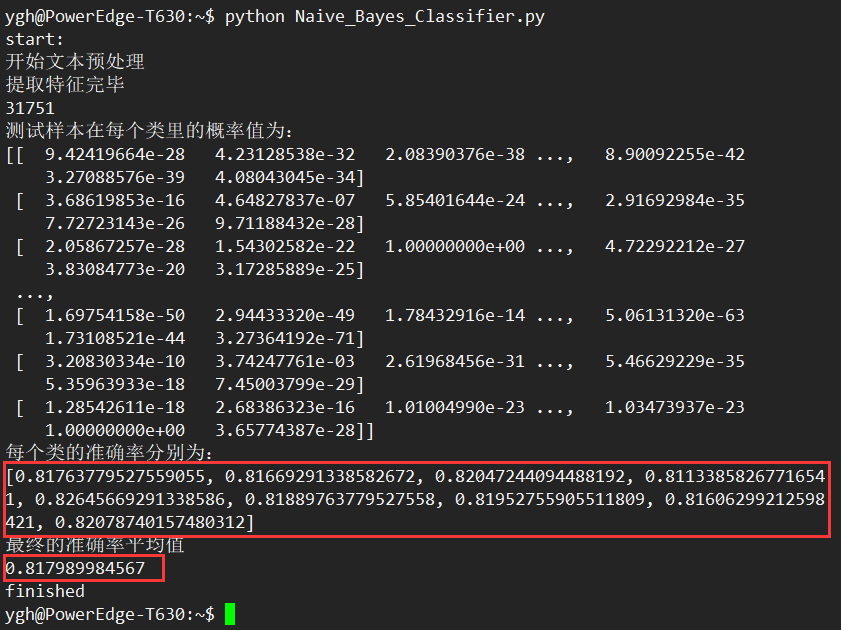
m = svm\_train(y, x )

训练，返回训练模型

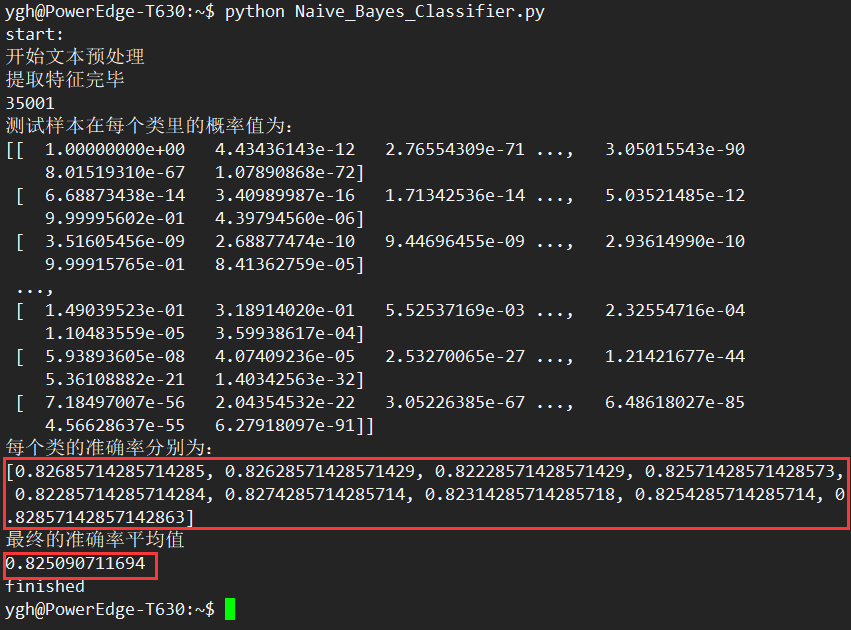
svm\_predict(yt,xt,m)

测试，

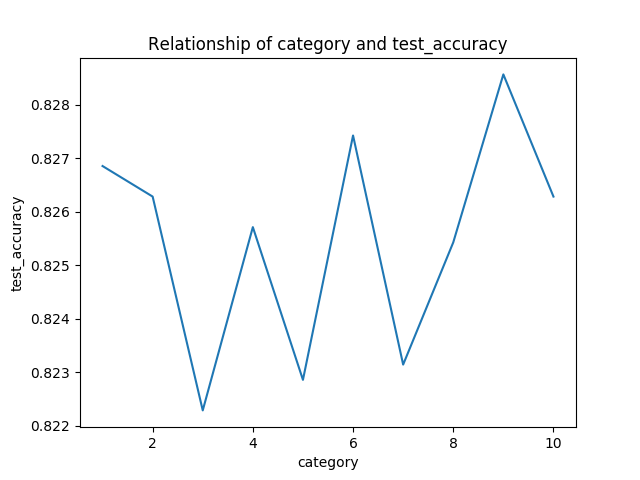
# 三 实验结果



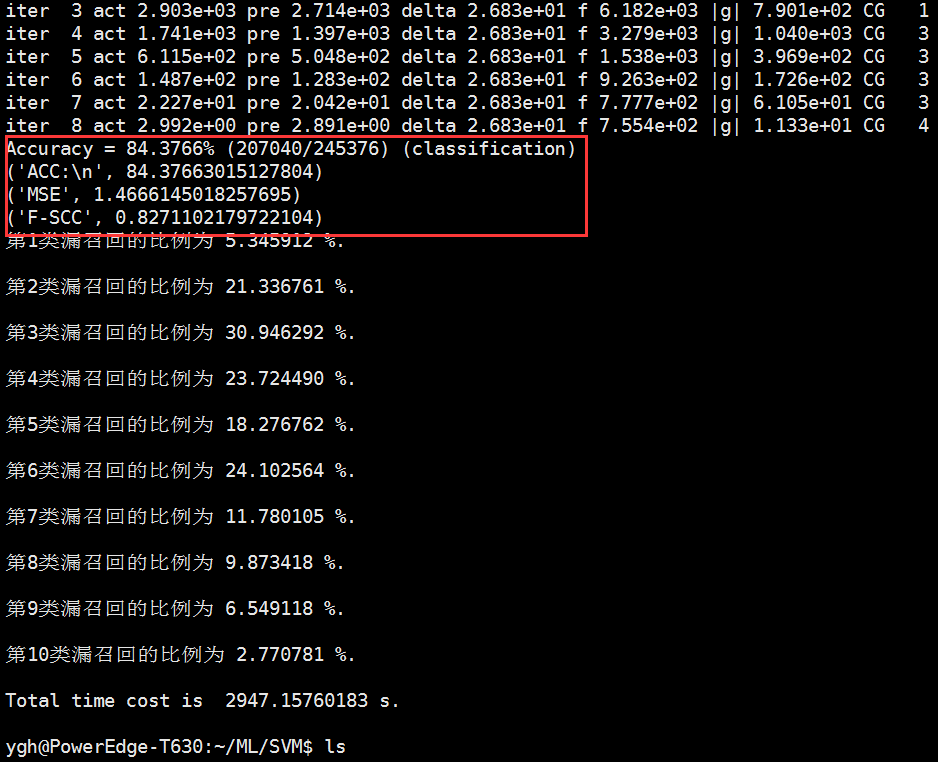
上图为bayes不带词频的每类的准确率和平均的准确率



上图为bayes带词频的准确率，相比不带词频的，准确率提高了1%



上图为每一类的准确率，从左到右一次为1-10类



上图为svm的准确率，召回率和f-score

# 四 总结与展望

本次文本分类实验遇到了很多困难，首先爬虫，很多网站反爬虫，最后找到了中国新闻网站，但是前些年的数据很多有问题，个别类别网页布局不同，然后数据预处理每一步都运行很多时间，比如分词，计算tfidf，降低维度都花了很久；然后根据降低维度的结果从新处理每篇文档去掉不在字典里的词语，，不过这几部分都可以开多个线程；最慢的是生成输入数据，因为这次不可以开多线程，而且要遍历所有文档，把它扩充成字典大小，变换索引，然后在去掉空的，大概需要12小时。而且有时候有一条数据出了问题，需要从新生成，然后因为降低维度以后有可能有的文章的所有词语都不出现在字典里，这样生成的输入数据会crash。

分类模型，贝叶斯没出现太大问题，libsvm跑了很久而且输入数据出了很多次问题，svm会出现内存溢出。

总的来说通过这次实验对数据挖掘里面文本分类的流程有了进一步的了解，并亲自动手实践了一下，从爬去数据到处理数据再到分类。尤其是对以前不了解的降低维度的方法有了进一步的认识。

为今后数据挖掘的道路打下了一点基础，希望以后越挖越开心。

# 五 附录

## 5.1 爬虫源码

=================================================================================

# -\*- coding: utf-8 -\*-

import scrapy

from scrapy.http import Request

from scrapy.selector import Selector

import os

import codecs

import time

import datetime

mkpath = "C:\\Users\\Y-GH\\Desktop\\NEWS-PA\\newsPa\\guoji\\"

#txtid = 0

#base\_dir = mkpath

class NewsSpider(scrapy.Spider):

name = 'news'

allowed\_domains = ['chinanews.com']

start\_urls = ['http://www.chinanews.com/']

def parse(self, response):

date\_list = getBetweenDay("20140101")

for day in date\_list:

year = day[0:4]

dayday = day[4:8]

#global base\_dir

#base\_dir = mkpath+day+"\\"

#mkdir(base\_dir)

#global txtid

time.sleep(0.1)

total = "http://www.chinanews.com/scroll-news/gj/{0}/{1}/news.shtml".format(year,dayday)

yield Request(total,meta = {"day":day},callback = self.info\_1)

def info\_1(self,response):

selector = Selector(response)

day = response.meta["day"]

base\_dir = mkpath+day+"\\"

mkdir(base\_dir)

txtid = 0

print "===============base\_dir=============="

list = selector.xpath("//div[@class='dd\_bt']/a/@href").extract()

for url in list:

txtid += 1

filename = base\_dir + str(txtid) +'.txt'

#url1 = "http://www.chinanews.com/" + url

yield Request(url,meta = {"filename":filename},callback = self.info\_2)

def info\_2(self,response):

selector = Selector(response)

filename = response.meta["filename"]

print "===============" + filename + "=============="

list = selector.xpath("//div[@class='left\_zw']/p/text()").extract()

#print list

#global txtid

#txtid +=1

#filename = base\_dir + str(txtid) +'.txt'

#print filename

f = codecs.open(filename,'a','utf-8')

for i in range(len(list)-1):

print '=========' + str(i) + "=========="

f.write(list[i])

f.close()

=================================================================================

## 5.2 朴素贝叶斯分类器源码

=================================================================================

coding: utf-8

import os

import time

import random

import sklearn

from sklearn.naive\_bayes import MultinomialNB

import numpy as np

import pylab as pl

import matplotlib.pyplot as plt

def MakeWordsSet(words\_file):

words\_set = set()

with open(words\_file, 'r') as fp:

for line in fp.readlines():

word = line.strip().decode("utf-8")

if len(word)>0 and word not in words\_set: # 去重

words\_set.add(word)

return words\_set

def TextProcessing(folder\_path, test\_size=0.2):

folder\_list = os.listdir(folder\_path)

data\_list = []

class\_list = []

# 类间循环

for folder in folder\_list:

new\_folder\_path = os.path.join(folder\_path, folder) #拼接路径

files = os.listdir(new\_folder\_path)

# 类内循环

j = 1

for file in files:

if j > 1000: # 每类text样本数最多1000

break

with open(os.path.join(new\_folder\_path, file), 'r') as fp:

raw = fp.read()

word\_list = list(raw.split())

data\_list.append(word\_list)

class\_list.append(folder.decode('utf-8'))

j += 1

## 划分训练集和测试集

# train\_data\_list, test\_data\_list, train\_class\_list, test\_class\_list = sklearn.cross\_validation.train\_test\_split(data\_list, class\_list, test\_size=test\_size)

data\_class\_list = zip(data\_list, class\_list)

random.shuffle(data\_class\_list)

index = int(len(data\_class\_list)\*test\_size)+1

train\_list = data\_class\_list[index:]

test\_list = data\_class\_list[:index]

train\_data\_list, train\_class\_list = zip(\*train\_list)

test\_data\_list, test\_class\_list = zip(\*test\_list)

# 统计词频放入all\_words\_dict

all\_words\_dict = {}

for word\_list in train\_data\_list:

for word in word\_list:

if all\_words\_dict.has\_key(word):

all\_words\_dict[word] += 1

else:

all\_words\_dict[word] = 1

# key函数利用词频进行降序排序

all\_words\_tuple\_list = sorted(all\_words\_dict.items(), key=lambda f:f[1], reverse=True) # 内建函数sorted参数需为list

all\_words\_list = list(zip(\*all\_words\_tuple\_list)[0])

return all\_words\_list, train\_data\_list, test\_data\_list, train\_class\_list, test\_class\_list

def words\_dict(all\_words\_list, deleteN, stopwords\_set=set()):

# 选取特征词

feature\_words = []

n = 1

for t in range(deleteN, len(all\_words\_list), 1):

if n > 8000: # feature\_words的维度1000

break

print all\_words\_list[t]

if not all\_words\_list[t].isdigit() and all\_words\_list[t] not in stopwords\_set and 1<len(all\_words\_list[t])<5:

feature\_words.append(all\_words\_list[t])

n += 1

#print "选取特征词"

#print feature\_words

return feature\_words

def TextFeatures(train\_data\_list, test\_data\_list, feature\_words):

def text\_features(text, feature\_words):

text\_words = set(text)

features = [1 if word in text\_words else 0 for word in feature\_words]

return features

train\_feature\_list = [text\_features(text, feature\_words) for text in train\_data\_list]

test\_feature\_list = [text\_features(text, feature\_words) for text in test\_data\_list]

return train\_feature\_list, test\_feature\_list

def TextClassifier(train\_feature\_list, test\_feature\_list, train\_class\_list, test\_class\_list,flag='sklearn'):

if flag == 'sklearn':

## sklearn分类器

classifier = MultinomialNB().fit(train\_feature\_list, train\_class\_list)

# print classifier.predict(test\_feature\_list)

# for test\_feature in test\_feature\_list:

# print classifier.predict(test\_feature)[0],

# print ''

print "测试样本在每个类里的概率值为："

print classifier.predict\_proba(test\_feature\_list)

len\_test = int(len(test\_feature\_list))

len\_list = range(0,len\_test,int(len\_test/10))

test\_accuracy = []

for i in len\_list[:9]:

test\_acc = classifier.score(test\_feature\_list[i:i+int(len\_test/10)], test\_class\_list[i:i+int(len\_test/10)])

test\_accuracy.append(test\_acc)

test\_all\_accuracy = classifier.score(test\_feature\_list, test\_class\_list)

else:

test\_accuracy = []

test\_all\_accuracy = 0

return test\_accuracy,test\_all\_accuracy

def select\_word(folder\_path,folder\_all\_word,test\_size):

folder\_list = os.listdir(folder\_path)

data\_list = []

class\_list = []

# 类间循环

for folder in folder\_list:

new\_folder\_path = os.path.join(folder\_path, folder) # 拼接路径

files = os.listdir(new\_folder\_path)

# 类内循环

j = 1

for file in files:

if j > 7000: # 每类text样本数最多1000

break

with open(os.path.join(new\_folder\_path, file), 'r') as fp:

raw = fp.read()

word\_list = list(raw.split())

data\_list.append(word\_list)

class\_list.append(folder.decode('utf-8'))

j += 1

## 划分训练集和测试集

# train\_data\_list, test\_data\_list, train\_class\_list, test\_class\_list = sklearn.cross\_validation.train\_test\_split(data\_list, class\_list, test\_size=test\_size)

data\_class\_list = zip(data\_list, class\_list)

random.shuffle(data\_class\_list)

index = int(len(data\_class\_list) \* test\_size) + 1

train\_list = data\_class\_list[index:]

test\_list = data\_class\_list[:index]

train\_data\_list, train\_class\_list = zip(\*train\_list)

test\_data\_list, test\_class\_list = zip(\*test\_list)

with open(folder\_all\_word, 'r') as f:

raw = f.read()

all\_words\_list = list(raw.split())

return all\_words\_list, train\_data\_list, test\_data\_list, train\_class\_list, test\_class\_list

if \_\_name\_\_ == '\_\_main\_\_':

print "start:"

print "开始文本预处理"

folder\_path = r'/home/ygh/newsData/splitwordfre/'

#all\_words\_list, train\_data\_list, test\_data\_list, train\_class\_list, test\_class\_list = TextProcessing(folder\_path, test\_size=0.2)

folder\_all\_word = r'/home/ygh/dict.txt'

feature\_words, train\_data\_list, test\_data\_list, train\_class\_list, test\_class\_list = select\_word(folder\_path,folder\_all\_word,test\_size=0.5)

print "提取特征完毕"

# 生成stopwords\_set

#stopwords\_file = r'C:\Users\Y-GH\PycharmProjects\document-classification\Database\stopwords\_cn.txt'

#stopwords\_set = MakeWordsSet(stopwords\_file)

## 文本特征提取和分类

flag = 'sklearn'

deleteNs = range(0, 1000, 20)

test\_accuracy\_list = []

train\_feature\_list, test\_feature\_list = TextFeatures(train\_data\_list, test\_data\_list, feature\_words)

print len(test\_feature\_list)

test\_accuracy,test\_all\_accuracy = TextClassifier(train\_feature\_list, test\_feature\_list, train\_class\_list, test\_class\_list, flag)

test\_accuracy\_list = test\_accuracy

print "每个类的准确率分别为："

print test\_accuracy\_list

print "最终的准确率平均值"

print test\_all\_accuracy

# 结果评价

plt.figure()

plt.plot(deleteNs, test\_accuracy\_list)

plt.title('Relationship of deleteNs and test\_accuracy')

plt.xlabel('deleteNs')

plt.ylabel('test\_accuracy')

plt.savefig('result.png')

print "finished"

=================================================================================

## 5.3 SVM文本分类器源码

# encoding:utf-8

import pandas as pd

import math

from liblinearutil import \*

import time

# 读取数据

def loadOriginData(src='E:\\Document-Classification\\document-classification-master\\train'):

# train.data

dataSrc = r'%s.data' % src

# train.label

labelSrc = r'%s.label' % src

label = pd.read\_table(labelSrc, sep=' ', names=['label'])

# train.map

mapSrc = r'%s.map' % src

# 每个文档拥有的terms

doc2term = {}

# 每个term出现在哪些文档

term2doc = {}

# 每个类别下有哪些docs

cate2docs = {}

# TF值

TF = {}

with open(dataSrc, 'r') as f:

for line in f:

str\_docIdx, str\_wordIdx, str\_cnt = line.split()

docIdx = int(str\_docIdx)

wordIdx = int(str\_wordIdx)

cnt = int(str\_cnt)

# update 数据结构

doc2term.setdefault(docIdx, []).append(wordIdx)

term2doc.setdefault(wordIdx, []).append(docIdx)

TF.setdefault(docIdx, {})[wordIdx] = cnt

# 统计每个类别下有哪些文档

with open(labelSrc, 'r') as f:

for line\_index, line in enumerate(f, 1):

labelVal = int(line.strip())

cate2docs.setdefault(labelVal, []).append(line\_index)

return TF, doc2term, term2doc, cate2docs, label

# 特征选择

def featureSel(doc2term, term2doc, cate2docs):

# CHI衡量的是特征项ti和类别Cj之间的关联程度, A,B, C, D是四个统计量

CHI\_cat2term = {}

# N：total number of documents

N = len(doc2term)

# A + B + C + D = N

# A： term出现在某类别中的文档总数

A = {}

# B: term出现在除某类别外的其他文档数

B = {}

# C: 该类别中不包含term的文档总数

C = {}

# D: 其他类别中不包含term的文档总数

D = {}

DF = {}

# 所有类别

categories = list(cate2docs.keys())

# 停用词词汇表

stopwords = {}

stopwordsSrc = r'E:\\Document-Classification\\document-classification-master\\stopwords.txt'

with open(stopwordsSrc) as f:

for line in f:

stopwords[line.strip()] = True

# 训练数据数据词汇表

vocSrc = r'E:\\Document-Classification\\document-classification-master\\vocabulary.txt'

voc = pd.read\_table(vocSrc, names=['voc'])

# 保存所有的特征

features = set()

# 计算一个类别标签下各个词的CHI

for category in categories:

# 属于第category类的文档为docs

docs = cate2docs[category]

sumVal = 0

for term in term2doc:

# 如果是停用词, 则将CHI置零

if stopwords.get(voc['voc'][term - 1], False):

CHI\_cat2term.setdefault(category, {})[term] = 0

continue

# 属于某类且包含term

AVal = len(set(term2doc[term]).intersection(set(docs)))

# 不属于某类但包含term

BVal = len(term2doc[term]) - AVal

# 属于某类，但不包含term

CVal = len(docs) - AVal

# 不属于某类， 不包含term

DVal = N - AVal - BVal - CVal

CHIVal = N \* (AVal \* DVal - CVal \* BVal)\*\*2 / ((AVal + CVal) \* (BVal + DVal) \* (AVal + BVal) \* (CVal + DVal))

# CHIVal = math.log(AVal \* N / ((AVal + CVal) \* (AVal + BVal)))

A.setdefault((term, category), AVal)

B.setdefault((term, category), BVal)

C.setdefault((term, category), CVal)

D.setdefault((term, category), DVal)

CHI\_cat2term.setdefault(category, {})[term] = CHIVal

DF[term] = AVal + BVal

sumVal += CHIVal

# 选出类别中CHI高于平均值的词

terms = CHI\_cat2term[category]

meanVal = sumVal / len(terms)

for term in terms:

if CHI\_cat2term[category][term] > meanVal:

features.add(term)

# for feature in features:

# print(voc['voc'][feature])

print('There are %d features in VSM model.\n' % len(features))

return features, DF

def buildSVMData(TF, DF, features, N, label, cate2docs, doc2terms):

isFeatures = dict(zip(features, [True] \* len(features)))

categories = list(cate2docs.keys())

# 如果是训练样本， 则计算归一化缩放因子，并返回

# y： label值

y = [0] \* N

# x: 稀疏矩阵

x = []

for i in range(N):

x.append({})

for category in categories:

for doc in cate2docs[category]:

# 给y进行标记类别

y[doc - 1] = label.iat[doc - 1, 0]

scale\_factor = -100

for term in doc2terms[doc]:

if isFeatures.get(term, False): # 如果term是特征

# TF值

TFVal = TF[doc].get(term, 0)

# TF-IDF值

tf\_idf = TFVal \* math.log(N / DF[term])

x[doc - 1][term] = tf\_idf

# 更新特征最大值

if scale\_factor < tf\_idf:

scale\_factor = tf\_idf

alpha = 0

# 按一篇文档中特征词最大的tf-idf, 对该文档中的所有特征词进行归一化

for term in doc2terms[doc]:

if isFeatures.get(term, False): # 如果term是特征

# x[doc - 1][term] = alpha + (1 - alpha) \* x[doc - 1][term] / scale\_factor

x[doc - 1][term] /= scale\_factor

print("Data for SVM has been built.\n")

return x, y

# 计算DF

def getDF(doc2term, term2doc, cate2docs):

DF = {}

for term in term2doc:

DF[term] = len(term2doc[term])

return DF

if \_\_name\_\_ == '\_\_main\_\_':

start = time.time()

# # 主程序

TF, doc2term, term2doc, cate2docs, label = loadOriginData()

# 特征选择

features, DF = featureSel(doc2term, term2doc, cate2docs)

# 读取数据(train.data)

TF, doc2term, term2doc, cate2docs, label = loadOriginData()

# 特征选择

features, DF = featureSel(doc2term, term2doc, cate2docs)

# build SVM model

x, y = buildSVMData(TF, DF, features, len(doc2term), label, cate2docs, doc2term)

# 读取测试数据(test.data)

TF\_test, doc2term\_test, term2doc\_test, cate2docs\_test, label\_test = loadOriginData(src='E:\\Document-Classification\\document-classification-master\\test')

DF\_test = getDF(doc2term\_test, term2doc\_test, cate2docs\_test)

# TF, DF, features, len(doc2term), label, cate2docs, doc2term, scales)

x\_test, y\_test = buildSVMData(TF\_test, DF\_test, features, len(doc2term\_test), label\_test, cate2docs\_test, doc2term\_test)

print("处理数据使用了 %s s时间。\n" % (time.time() - start))

# # 调用 liblinear 库进行分类

prob = problem(y, x)

param = parameter('-s 0 -c 4 -B 1')

# 训练

m = train(prob, param)

# 预测test.data

p\_label, p\_acc, p\_vals = predict(y\_test, x\_test, m, '-b 1')

# 评价

ACC, MSE, SCC = evaluations(y\_test, p\_label)

print('ACC:\n', ACC)

print('MSE', MSE)

print('SCC', SCC)

# 统计每类中错误率

categoriesErrs = {}

for doc\_index, doc\_label in enumerate(y\_test):

if doc\_label != int(p\_label[doc\_index]):

cateogory = label\_test.iat[doc\_index, 0]

categoriesErrs.setdefault(cateogory, []).append(doc\_index + 1)

# with open('outcome.txt', 'wb') as f:

print("错误分类的样本为：\n")

for categoryErr in categoriesErrs:

numOfErr = len(categoriesErrs[categoryErr])

print('第%d类共 %d样本, 被错分的个数为 %d, 比例为 %f %%.\n' % (categoryErr,len(cate2docs\_test[categoryErr]), numOfErr, float(numOfErr)/float(len(cate2docs\_test[categoryErr]))\*100.0))

end = time.time()

print("Total time cost is %s s.\n" % (end - start))

## 5.4 libsvm分类器源码

import sys

path = r"E:\SVM\libsvm-3.22\python"

sys.path.append(path)

from svmutil import \*

y, x = svm\_read\_problem(r'E:\Document-Classification\data\trainscale.svm')#读入训练数据

yt, xt = svm\_read\_problem(r'E:\Document-Classification\data\testscale.svm')#训练测试数据

print "开始训练："

m = svm\_train(y, x )#训练

print "开始测试："

svm\_predict(yt,xt,m)#测试

## 5.5 数据处理源码

import jieba

import jieba.posseg as pseg

import os,re,collections

import sys

import numpy as np

from numpy import nan as Na

import pandas as pd

from pandas import Series,DataFrame

sys.setrecursionlimit(999999999)#增加递归次数

stopwords = {}.fromkeys([ line.rstrip() for line in open('stopwords.txt','r',encoding='utf-8')])

# print(stopwords)

#stopwords = {}.fromkeys(['时代', '新机遇','机遇','意识','人'])

#初始每类的路径列表

#遍历txt文件，分词、取名次、去停用词

def gci(filepath,i):

#遍历filepath下所有文件，包括子目录

files = os.listdir(filepath)

for fi in files:

path = os.path.join(filepath,fi)

if os.path.isdir(path):

gci(path)

else:

sliptword(path,i)

#分词、取名次、去停用词

def sliptword(path,i):

print(path)

strinfo = re.compile('news')#news

writepath=strinfo.sub('split',path) #分词结果写入的路径split

with open(path, "r",encoding='utf-8') as f:

text = f.read()

#print(text)

str = ""

str2=""

result = pseg.cut(text) ##词性标注，标注句子分词后每个词的词性

for w in result:

#print(w.word, "/", w.flag, ", ", end=" ")

if w.flag.startswith('n'):

#print(w.word, "/", w.flag)

if w.word not in stopwords:

# with open(writepath, "a") as f:

# #f.write(w.word+"/"+w.flag+"\n")

# f.write(w.word + "\n")

str = str + w.word+"\n"

str2 =str2+w.word+" "

with open(writepath,"a")as f:

f.write(str)

with open(oriDictPathList[i], "a")as f:

f.write(str2)

def tfidf(oriDictPathList):

import sklearn

from sklearn.feature\_extraction.text import CountVectorizer

# 语料

corpus=[]

for i in range(len(oriDictPathList)):

print(i,"\n")

with open(oriDictPathList[i],'r',encoding='utf-8')as f:

corpus.append(f.read())

print(len(corpus[i]),"\n")

# corpus = [

# 'This is the first document.',

# 'This is the second second document.',

# 'And the third one.',

# 'Is this the first document?',

# ]

# 将文本中的词语转换为词频矩阵

vectorizer = CountVectorizer()

# 计算个词语出现的次数

X = vectorizer.fit\_transform(corpus)

# 获取词袋中所有文本关键词

word = vectorizer.get\_feature\_names()

#print("word:",word)

with open("E:\\新建文件夹\\文本\\新建文件夹\\python\\DataMinning\\tfidf\\allword2.txt",'w',encoding='utf-8') as f:

print(len(word))

s='\n'.join(word)

print(len(s))

f.write('\n'.join(word))

# 查看词频结果

# print("X.toarray():",X.toarray())

#np.set\_printoptions(threshold='nan')

np.set\_printoptions(threshold=np.inf)

# print(X.toarray()[0])

print(1)

for i in range(len(wordfrepath)):

print("i:",i)

s = str(X.toarray()[i])

s = s.lstrip('[')

s = s.rstrip(']')

with open(wordfrepath[i], 'w', encoding='utf-8')as f:

f.write(s)

from sklearn.feature\_extraction.text import TfidfTransformer

# 类调用

transformer = TfidfTransformer()

print("transformer:",transformer)

# 将词频矩阵X统计成TF-IDF值

# !!!!正规化处理一下词频矩阵

tfidf = transformer.fit\_transform(X)

# 查看数据结构 tfidf[i][j]表示i类文本中的tf-idf权重

# print("tfidf.toarray()",tfidf.toarray())

np.set\_printoptions(threshold=np.inf)

print(2)

for i in range(len(tfidfpath)):

print(i)

s = str(tfidf.toarray()[i])

s = s.lstrip('[')

s = s.rstrip(']')

with open(tfidfpath[i], 'w', encoding='utf-8')as f:

f.write(s)

#快排

def parttion(v1,v2, left, right):

key1 = v1[left]

key2 = v2[left]

low = left

high = right

while low < high:

while (low < high) and (v1[high] <= key1):

high -= 1

v1[low] = v1[high]

v2[low] = v2[high]

while (low < high) and (v1[low] >= key1):

low += 1

v1[high] = v1[low]

v2[high] = v2[low]

v1[low] = key1

v2[low] = key2

return low

def quicksort(v1,v2, left, right):

if left < right:

p = parttion(v1,v2, left, right)

print(p)

quicksort(v1,v2, left, p-1)

quicksort(v1,v2, p+1, right)

return v1,v2

def reducedimension(tfidfpath,allwordpath,newtfidfpath,newdictpath):

for i in range(len(tfidfpath)):

with open(tfidfpath[i],'r',encoding='utf-8')as f:

text=f.read()

tfidftemp = text.split()

print("i1", i)

with open(allwordpath,'r',encoding='utf-8')as f:

text=f.read()

allwordlisttemp =text.split()

print("i2", i)

print(len(tfidftemp))

tfidflist = []

allwordlist = []

for j in range(len(tfidftemp)):

k = float(tfidftemp[j])

if k > 9.99999999e-05:

tfidflist.append(k)

allwordlist.append(allwordlisttemp[j])

print(tfidflist)

print(allwordlist)

newtfidflist,newallwordlist=quicksort(tfidflist,allwordlist,0,len(tfidflist)-1)

with open(newtfidfpath[i],'w',encoding='utf-8')as f:

f.write(" ".join(str(newtfidflist)))

print("i3 tfidf",i)

with open(newdictpath[i],'w',encoding='utf-8')as f:

f.write(" ".join(newallwordlist))

print("i3 word",i)

def createdict(newtfidfpath,newdictpath,dictpath):

#dictdataframe = pd.DataFrame()

l=[]

strinfo = re.compile('newdict') # news

strinfo = re.compile('newtfidf') # news

#print(dictdataframe)

for i in range(len(newtfidfpath)):

with open(newtfidfpath[i],'r',encoding='utf-8')as f:

tfidflist=[float(e) for e in f.read().split()[0:2000]]

print(tfidflist)

print(len(tfidflist))

with open(newdictpath[i],'r',encoding='utf-8')as f:

wordlist=f.read().split()[0:2000]

print(wordlist)

print(len(wordlist))

s=Series(tfidflist,wordlist)

l.append(s)

dictdataframe = pd.DataFrame(l)

#存起来

pd.set\_option('max\_colwidth', 20000000)

print(dictdataframe)

print(" ".join(dictdataframe.columns.tolist()))

with open(dictpath,'w',encoding='utf-8')as f:

f.write(" ".join(dictdataframe.columns.tolist()))

def createinputdata(dictpath,splitPathList,csvpath):

# 生成每篇series扩展成总的dict的index

# 生成一个大的dataframe

# 两个dataframe相乘

# k1=Series([1,1],index=['c','d'])

# k2=Series([1.5,2.6,7.6,8.9],index=['a','b','c','d'])

# k3=k1\*k2

# print(k3)

articlelist=[]

with open(dictpath,'r',encoding='utf-8')as f:

dictlist=f.read().split()

dictSeries=Series(np.ones(len(dictlist)).tolist(),dictlist)

print(dictSeries)

for i in range(len(splitPathList)):

files = os.listdir(splitPathList[i])

index=0

listtemp=[]

for fi in files:

path = os.path.join(splitPathList[i], fi)

with open(path,'r')as f:

article=list(set(f.read().strip().split('\n')))

articleSeries = Series(np.ones(len(article)).tolist(), article).unique

#print(article)

articleSeries=dictSeries\*articleSeries

#print(3)

# print(articleSeries.dropna())

articlelist.append(articleSeries)

listtemp.append((articleSeries))

index=index+1

if index%500==0 :

articledataframe = DataFrame(articlelist)

articlelist.clear()

# print(articledataframe)

if index<=50000:

with open("E:\\python\\DataMinning\\inputdata\\articletrain.csv", 'a')as f:

articledataframe.to\_csv(f, header=False)

else:

with open("E:\\python\\DataMinning\\inputdata\\articletest.csv", 'a')as f:

articledataframe.to\_csv(f, header=False)

print("i index",i," ",index)

# tempdataframe = DataFrame(articlelist)

# tempdataframe.to\_csv(csvpath[i])

# articledataframe = DataFrame(articlelist)

# #print(articledataframe)

# articledataframe.to\_csv("E: \\python\\DataMinning\\inputdata\\articledataframe.csv")

def createnewsplit(dictpath,splitPath):

strinfo = re.compile('split') # news

with open(dictpath,'r',encoding='utf-8')as f:

worddict=f.read().split()

#ds = Series(np.ones(len(worddict)).tolist(), worddict)

#print(worddict)

for i in range(len(splitPath)):

files=os.listdir(splitPath[i])

index1=1

for fi in files:

path = os.path.join(splitPath, fi)

#if index1<=500:

# list1 = [line.rstrip('\n') for line in open(path, 'r',encoding='utf-8')]

#else:

# list1 = [line.rstrip('\n') for line in open(path, 'r')]

if index1 >= 0:

try:

list1 = [line.rstrip('\n') for line in open(path, 'r',encoding='utf-8')]

except:

list1 = [line.rstrip('\n') for line in open(path, 'r')]

list2=[e for e in list1 if e in worddict]

list3=list(set(list1))

writepath = strinfo.sub('t5', path) # 分词结果写入的路径split

with open(writepath, 'w', encoding='utf-8')as f:

f.write(" ".join(list3))

print("index1",index1)

index1 = index1 + 1

def svminputdata(dictpath,SplitPathList):

with open(dictpath,'r',encoding='utf-8')as f:

dict=f.read().split()

rindex=list(range(len(dict)))

#rindex=["t"+str(e) for e in range(len(dict))]

sd=Series(np.ones(len(dict)).tolist(),index=dict)

strinfo = re.compile('split') # news

trainindex=1

testindex=1

for i in range(len(SplitPathList)):

index1 = 1

num=i+1

files = os.listdir(SplitPathList[i])

for fi in files:

#print("s1")

path=os.path.join(SplitPathList[i],fi)

try:

with open(path, 'r', encoding='utf-8')as f:

list1 = f.read().split()

except:

with open(path, 'r')as f:

list1 = f.read().split()

list2 = [e for e in list1 if e in dict]

# print(list2)

s = Series(list2)

s = s.value\_counts()

# print(list(s.index))

# print(list(s.values))

s2 = Series(s.values, index=s.index)

# print("s2",s2)

s3 = s2 \* sd

s3 = Series(s3.values, index=rindex)

# s3=s3.fillna(0)

# s3=s3.dropna()

# print("s3",s3)

s4 = s3[s3.notnull()]

# print("s4",s4)

s4index = s4.index

s4values = s4.values

# pint(s4index)

# print(s4values)

if index1<=500:

if trainindex>=0:

str1 = ""

for j in range(len(s4)):

str1 = str1 + str(trainindex) + " " + str(s4index[j]) + " " + str(int(s4values[j])) + "\n"

with open("E: \\python\\DataMinning\\inputdata\\svm\\train1.data", 'a',

encoding='utf-8')as f:

f.write(str1)

with open("E:\\python\\DataMinning\\inputdata\\svm\\train1.label", 'a',

encoding='utf-8')as f:

f.write(str(num) + "\n")

writepath = strinfo.sub('splitwordfre', path)

# with open(writepath, 'w', encoding='utf-8')as f:

# f.write(" ".join(list2))

trainindex += 1

if index1>500 and index1<=1000:

if testindex>=0:

str1 = ""

for j in range(len(s4)):

str1 = str1 + str(int(testindex)) + " " + str(s4index[j]) + " " + str(int(s4values[j])) + "\n"

with open("E: \\python\\DataMinning\\inputdata\\svm\\test1.data", 'a',

encoding='utf-8')as f:

f.write(str1)

with open("E: \\python\\DataMinning\\inputdata\\svm\\test1.label", 'a',

encoding='utf-8')as f:

f.write(str(num) + "\n")

writepath = strinfo.sub('splitwordfre', path)

# with open(writepath, 'w', encoding='utf-8')as f:

# f.write(" ".join(list2))

testindex += 1

if index1==1000:

break

index1+=1

# if trainindex==100:

# break

print("type trainindex testindex articleindex",num," ",trainindex," ",testindex," ",index1)

#print("index1",index1)

# if index1==3:

# break

def libsvminputdata(dictpath, newdictpath,newtfidfpath,newSplitPathList):

with open(dictpath,'r',encoding='utf-8')as f:

dict=f.read().split()

sd=Series(np.ones(len(dict)).tolist(),index=dict)

print(len(dict))

sl=[]

rindex=[float(e) for e in range(len(dict))]

for i in range(len(newdictpath)):

with open(newdictpath[i], 'r', encoding='utf-8')as f:

alldict = f.read().split()

with open(newtfidfpath[i], 'r', encoding='utf-8')as f:

alltfidf = [float(e) for e in f.read().split()]

print(len(alldict))

print(len(alltfidf))

sad = Series(alltfidf, index=alldict)

sl.append(sad)

for i in range(len(newSplitPathList)):

files = os.listdir(newSplitPathList[i])

num=i+1

for fi in files:

#print("s1")

path=os.path.join(newSplitPathList[i],fi)

try:

with open(path, 'r', encoding='utf-8')as f:

list1 = f.read().split()

except:

with open(path,'r')as f:

list1=f.read().split()

#print(list1)

s=Series(np.ones(len(list1)).tolist(),index=list1)

print("s1",len(s))

s2=s\*sl[i]

print("s2",len(s2))

s3=s2\*sd

#s3=Series(s3.values,index=rindex)

print("s3",len(s3))

break

#s3=s3.fillna(0)

#s3=s3.dropna()

#print("s3",s3)

s4=s3[s3.notnull()]

#print("s4",s4)

s4index=s4.index

s4values=s4.values

#print(s4index)

#print(s4values)

str1=""

for j in range(len(s4)):

str1 = str1+str(trainindex) + " " +str(s4index[j])+" "+str(int(s4values[j]))+"\n"

with open("",'a',encoding='utf-8')as f:

f.write(str1)

with open("",'a',encoding='utf-8')as f:

f.write(str(num)+"\n")

break