



|  |  |
| --- | --- |
| **题目：** | 文本数据的分类与分析 |
| **学院：** | 计算机学院 |

小组成员：

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 学号 | 班级 | 分工 |
| 王煜辉 |  |  | 朴素贝叶斯和SVM分类器的实现 |
|  |  |  | 数据集构造 |
|  |  |  | 数据清洗和分词程序的实现 |

报告日期：2018年12月12日

# 实验目的

1. 掌握数据预处理的方法，对训练集数据进行预处理；
2. 掌握文本建模的方法，对语料库的文档进行建模；
3. 掌握分类算法的原理，基于监督学习的机器学习方法，训练文本分类器；
4. 利用学习的文本分类器，对未知文本进行分类判别；
5. 掌握评价分类器性能的评估方法。

# 实验要求

* 1. 文本类别>=10类。
  2. 训练集文档数>=500000篇，每类平均50000篇；测试集文档数>=500000篇，每类平均50000篇。
  3. 分组实验，组员数量<=3。

# 实验内容

## 3.1文本数据的爬取

为了爬取动态网站我是用的技术是Python3+Selenium+CSS3::selection。Python为网络爬虫提供了很多的库，因此Python是实现网络爬虫的一个很好的选择。Selenium是一个用于Web应用程序测试的工具。Selenium测试直接运行在浏览器中，就像真正的用户在操作一样。它支持的浏览器包括IE、Firefox、Safari、Google Chrome、Opera等浏览器。这个工具的主要功能包括：测试与浏览器的兼容性包括测试你的应用程序看是否能够很好得工作在不同浏览器和操作系统之上。测试系统功能包括创建回归测试检验软件功能和用户需求。支持自动录制动作和自动生成 .Net、Java、Perl等不同语言的测试脚本。本次主要用Selenium模拟用户浏览器行为中的滚动行为，点击下一页新闻，以及点击显示更多新闻的行为。CSS3：：selection选择器是用来匹配被用户选取的部分，比如新闻的链接和内容，日期等等。

本次实验爬取的网站是新浪新闻网，通过爬虫一共爬取了90余万的数据

集，又通过和清华的公开新浪新闻数据集综合了一下，大概有130万数据集存入数据库中。共13类，每类数据10万。分别是china，currentaffair，education，ent，finance，food，health，history，houseproperty，military，sport，stock，technology。之后考虑到相邻新闻之间的类似性，将数据隔行划分成训练集和测试集，每个数据集各5万。

## 3.2数据预处理

数据预处理首先要对每条新闻进行过滤脏数据的操作，利用Python正则表达式过滤脏数据。然后进行提取名词、去除停用词的操作，最终把分词得到的结果保存在TXT文件里，其中分词和词性标注采用的工具是jieba。

sklearn.datasets,base包下的Bunch类可以帮助我们方便地定义数据集的数据结构。我们定义数据集的数据结构为：

bunch = Bunch(target\_name=[],lable=[],filenames=[],contents=[])

其中target\_name为数据集总共有多少类的数据，contents对应每条新闻，filenames对应保存这条新闻的TXT文件路径，lable对应这条数据的标签。读取TXT文件初始化bunch最后把bunch通过pickle保存到磁盘的trainData.dat文件中。用同样的方法得到测试数据集testData.dat。

## 3.3朴素贝叶斯算法

朴素贝叶斯原理：

 P(B|A) = (P(A|B)P(B))/(P(A)) 如果应用到文本分类中，我们假设有类别集合 C = {C\_1,C\_2,C\_3,C\_4,C\_5,C\_6,C\_7,C\_8,C\_9,C\_10}，那么文档D属于类别C\_i的概率就可以使用贝叶斯公式计算： P(C\_i│D)= (P(D│C\_i )P(C\_i ))/P(D) = (P(C\_i))/(P(D))\*P(D|C\_i) 因为对每一个分类来说，P(C\_i)恒等于1/10，P(D)都相等，所以若要比较C1、C2…C10的大小，只需计算P(D|C\_i)即可。 假设文档D的特征集合X有n个特征：X = {x\_1,x\_2…x\_n} ,那么P(D|C\_i)的计算公式是： P(D|Ci) =P(x\_1 |Ci)+P(x\_2 |Ci)…P(x\_n |Ci) 令 P(C\_k |D) =max{P(C\_1 |D),P(C\_2 |D)…P(C\_10 |D)} 那么，我们就判断文档D属于类别C\_k。

首先，将每篇文档转化为特征向量，即特征提取。文本分类中最著名的特征提取方法就是向量空间模型（VSM），即将样本转换为向量的形式。为了能实现这种转换，需要做两个工作：确定特征集和提取特征。特征集其实就是词典。特征权重的计算方式本实验选择TF\*IDF。TF词频，IDF逆向文件频率。TF表示词条在文档d中出现的频率。IDF的主要思想是：如果包含词条t的文档越少，也就是n越小，IDF越大，则说明词条t具有很好的类别区分能力。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取以10为底的对数得到。从trainData.dat文件得到训练数据集bunch，运行响应代码。即可将每篇文档转化为特征向量，然后利用pickle保存得到的特征向量到train\_tfidf.dat文件中。max\_df参数含义：当构建词汇表时，严格忽略高于给出阈值的文档频率的词条，以此实现降维度。sublinear\_tf参数含义：应用线性缩放TF，例如，使用1+log(tf)覆盖tf。对测试数据集testData.dat作同样的处理得到test\_tfidf.dat文件。

然后，MultinomialNB实现了服从多项分布数据的朴素贝叶斯算法，也是用于文本分类(这个领域中数据往往以词向量表示，尽管在实践中 tf-idf 向量在预测时表现良好)的两大经典朴素贝叶斯算法之一。 分布参数由每类 的 向量决定，theta_y = (\theta_{y1},\ldots,\theta_{yn})式中http://sklearn.apachecn.org/docs/img/c87d9110f3d32ffa5fa08671e4af11fb.jpg是特征的数量(对于文本分类，是词汇量的大小) theta_{yi}是样本中属于类http://sklearn.apachecn.org/docs/img/0775c03fc710a24df297dedcec515aaf.jpg中特征http://sklearn.apachecn.org/docs/img/43e13b580daefe5ba754b790dfbd216c.jpg概率(x_i \mid y)。最后对于每类都计算的和，哪类的和结果最大就预测为该类。运行相应的语句，得到MultinomialNBModel.dat文件。

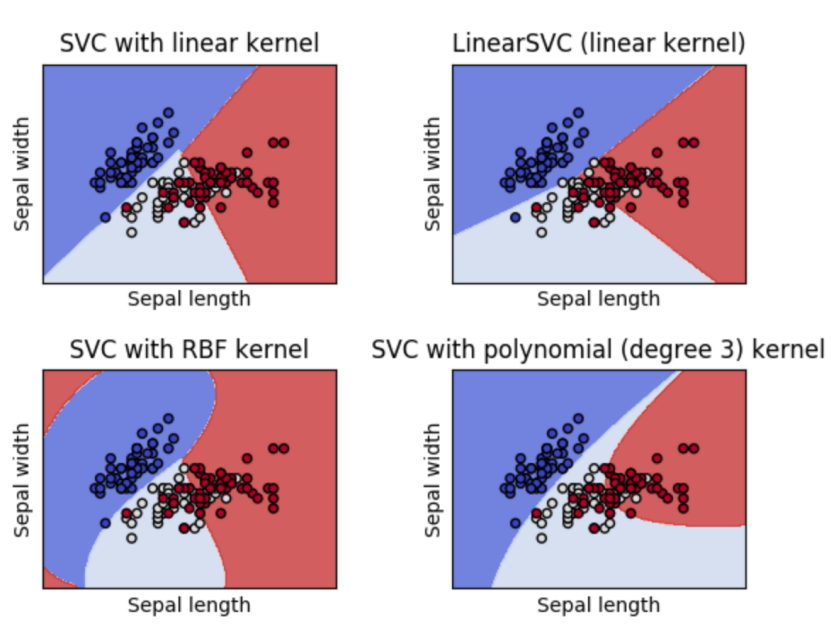
最后，加载朴素贝叶斯模型并得到测试结果。从MultinomialNBModel.dat加载训练好的朴素贝叶斯分类器，从test\_tfidf.dat加载测试数据集的特征向量，运行predicted = clf.predict(testSet.tfidfmetrix)预测分类结果，其中clf是分类器，得到的predicted是一个List ，每个元素对应测试数据集中一条新闻的预测结果。初始化一个大小为10\*10的零矩阵，遍历predicted，如果种类i预测成了种类j，则下标为i，j的矩阵元素加一，以此来统计预测结果。最后用sklearn中的metrics工具包得到模型的精度、召回率和f1值。用metrics工具包下的classification\_report函数得到每个分类结果的精度、召回率和f1值。

## 3.4 SVM算法

SVM算法中的数据预处理、读取TXT文件，将训练数据集加载到内存中、将每篇文档转化为特征向量，即特征提取和加载模型并查看测试结果和朴素贝叶斯算法类似，故此处省略不写。

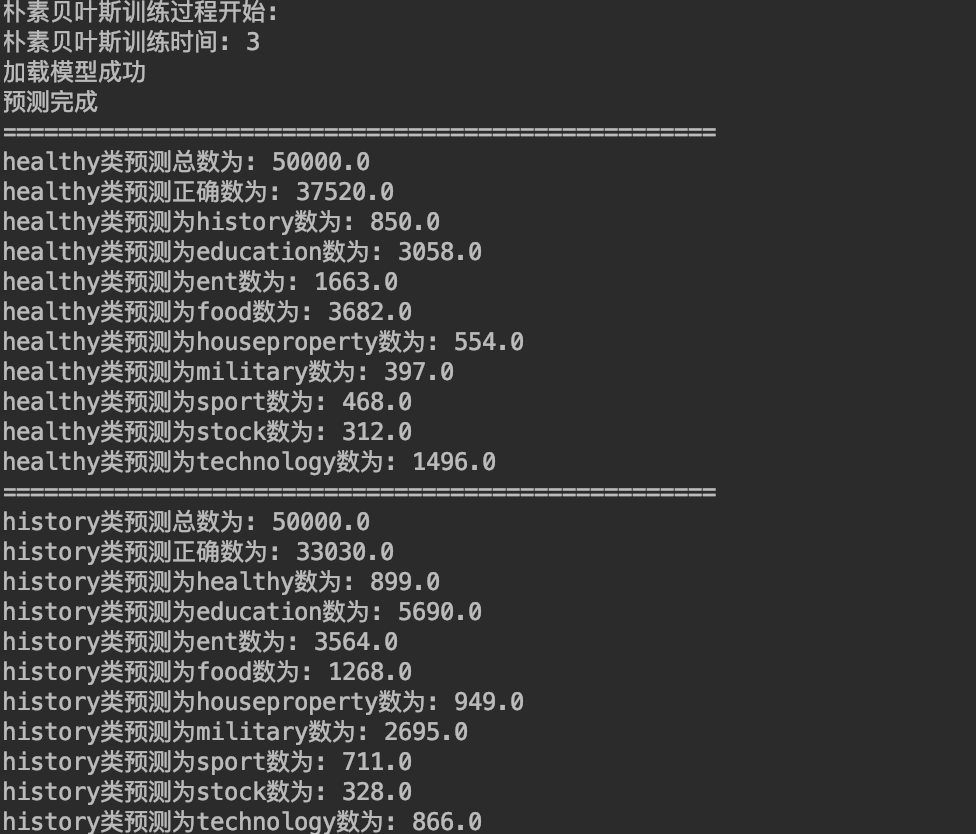
训练SVM模型并且保存得到的结果, svm.SVC函数中最重要的两个参数分别是C和kernel，C是误差项的惩罚参数，一般取值为10的n次幂， n=-5~inf。C越大，相当于惩罚松弛变量，希望松弛变量接近0，即对误分类的惩罚增大，趋向于对训练集全分对的情况，这样会出现训练集测试时准确率很高，但泛化能力弱。C值小，对误分类的惩罚减小，容错能力增强，泛化能力较强。kernel为SVM算法中所用的核函数，采用不同的核函数分类效果如下：

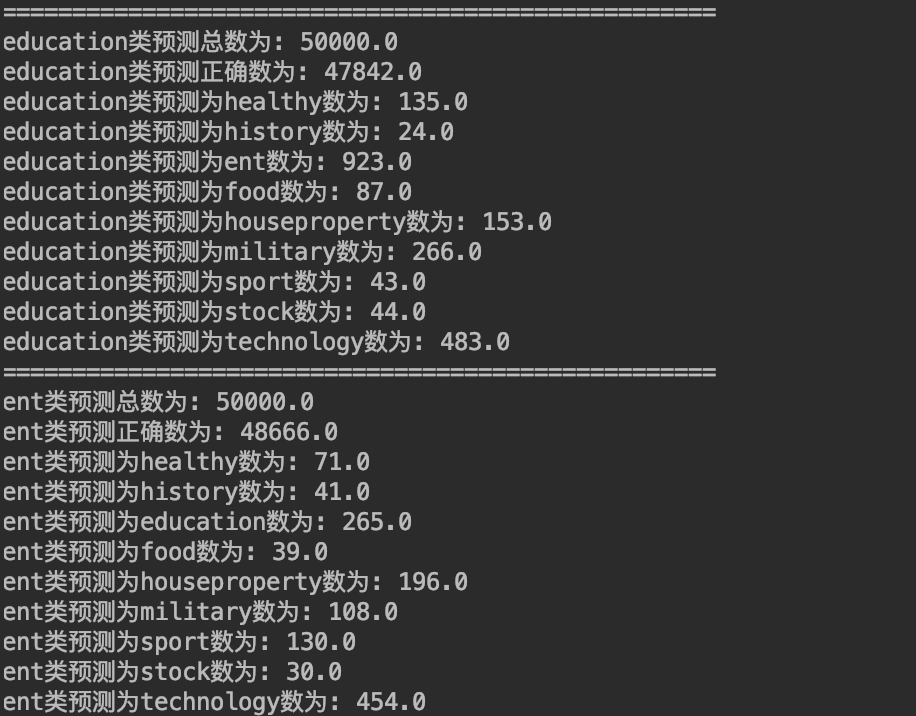
在本实验中我测试了C=0.1、1.0、10.0、100.0和kernel分别为rbf和linear时候的分类效果，发现C=1.0和kernel=linear时候分类效果最好。

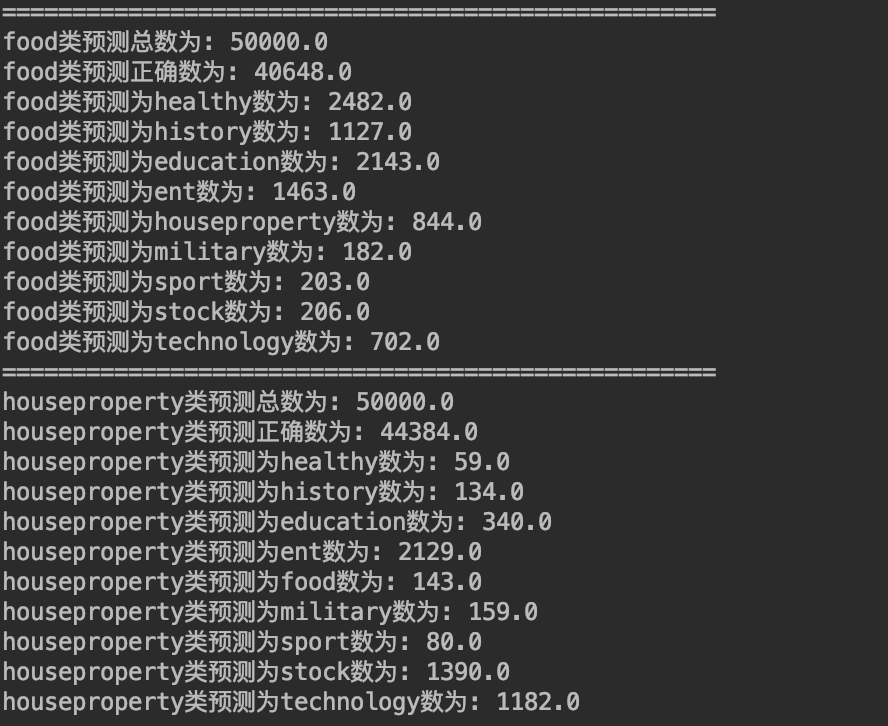


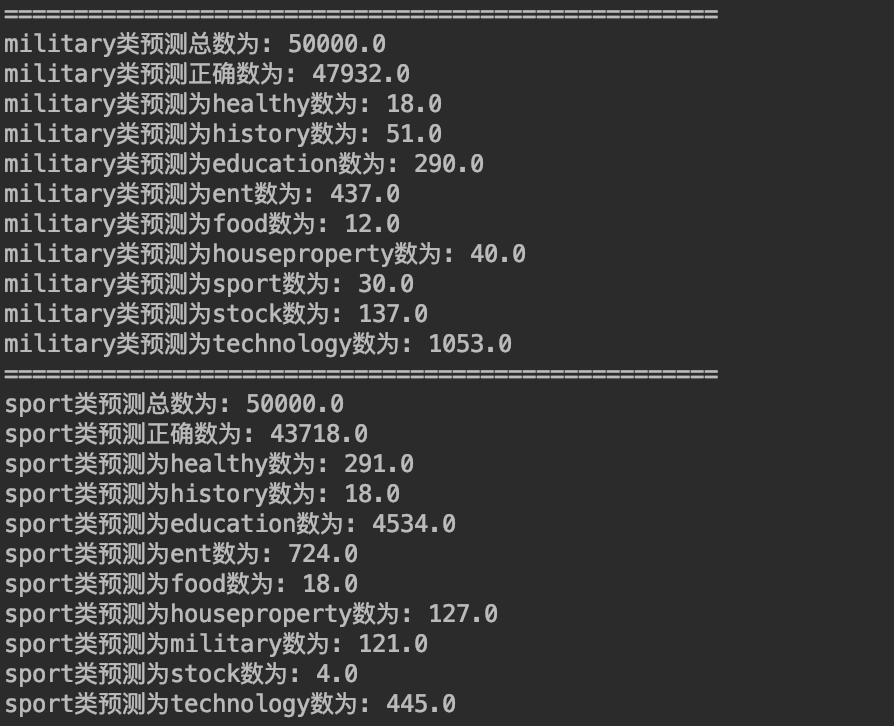
# 实验结果

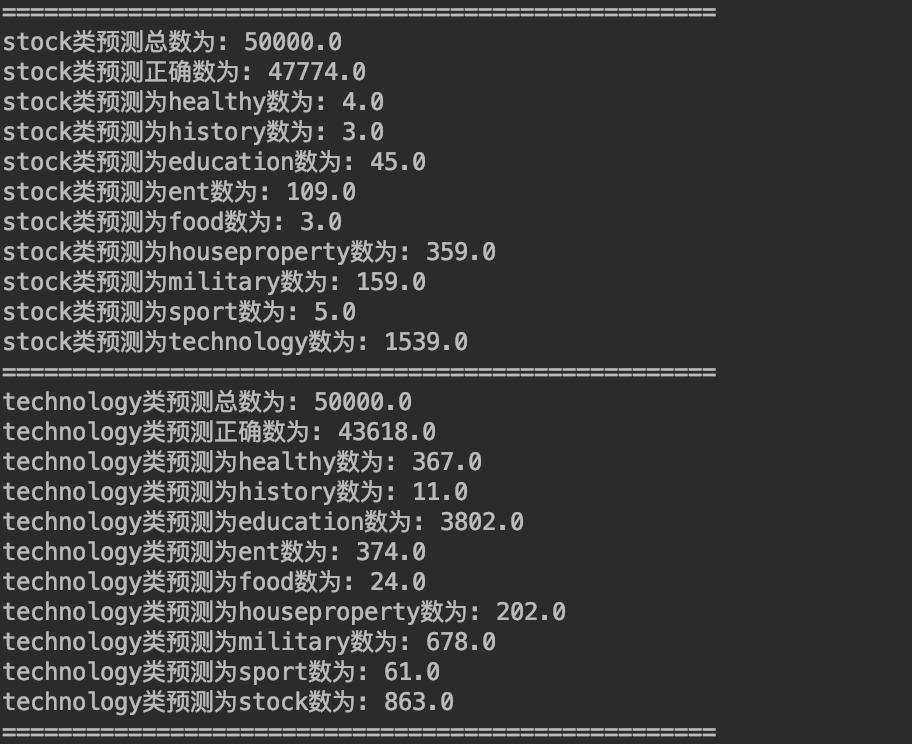
朴素贝叶斯：

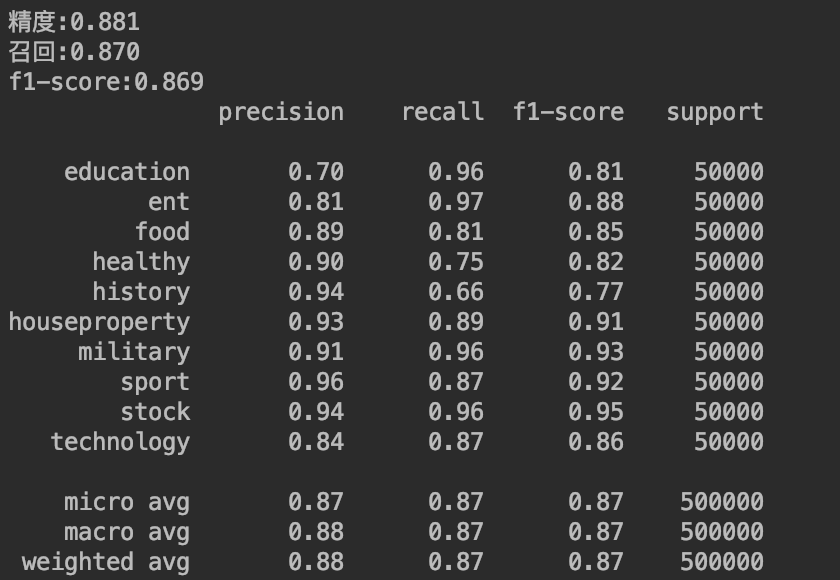


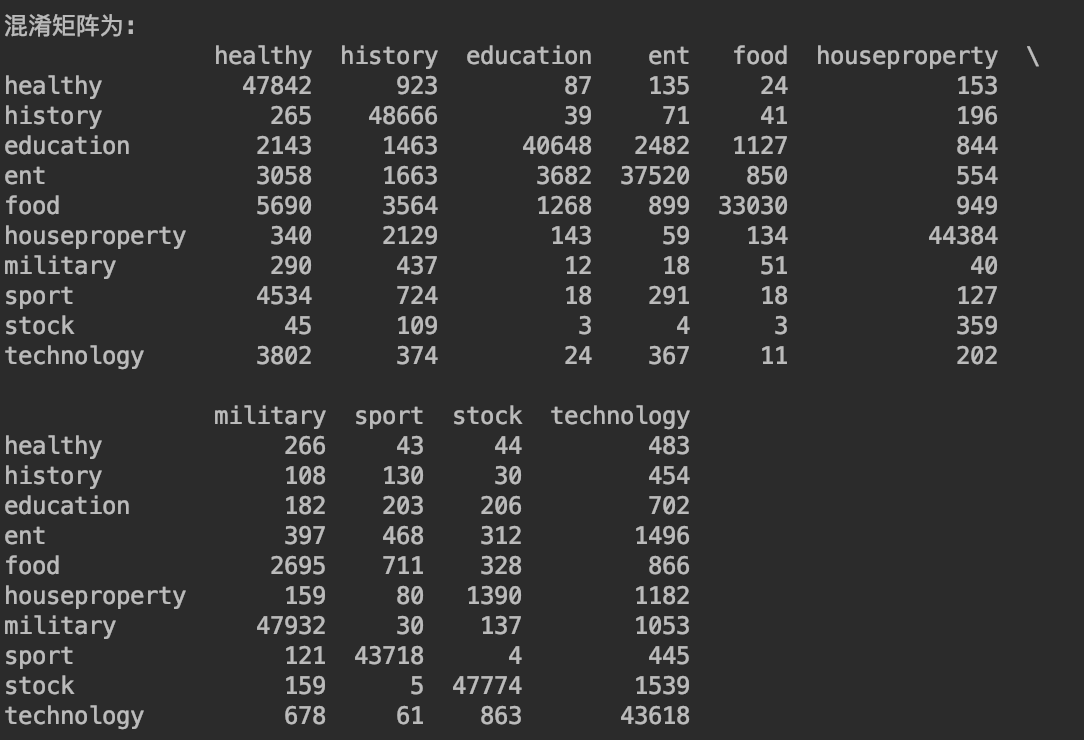




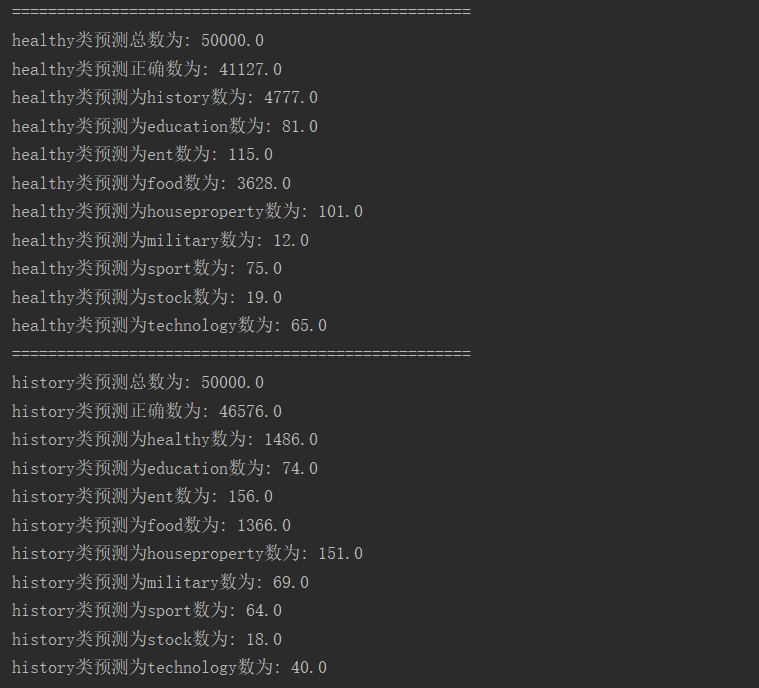
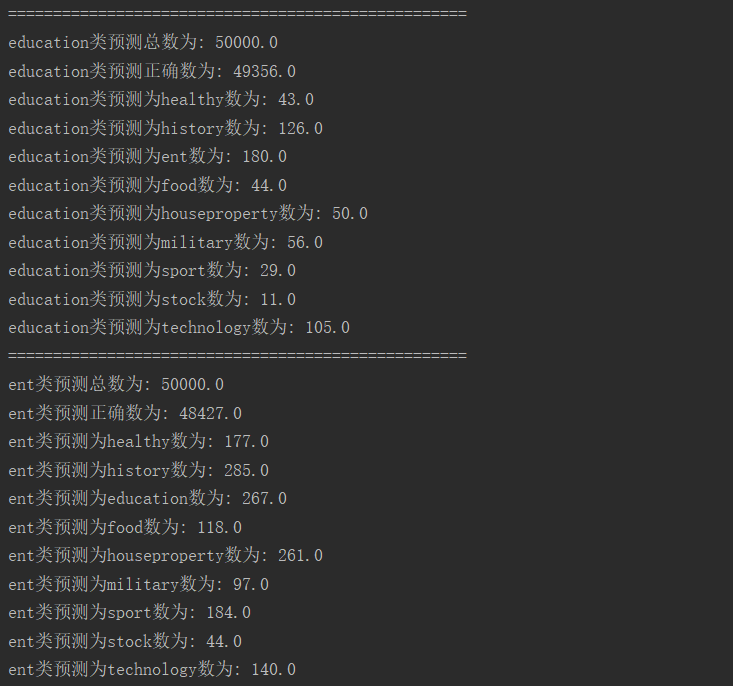


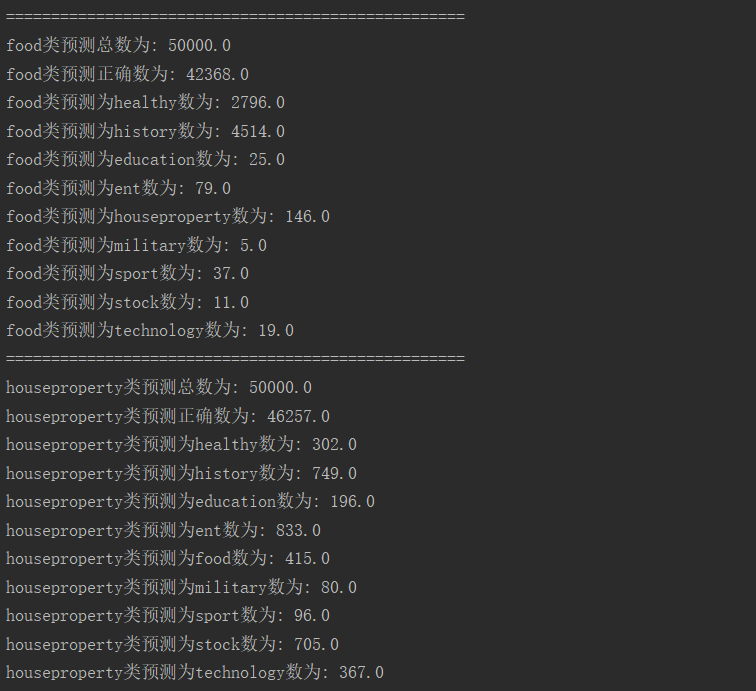


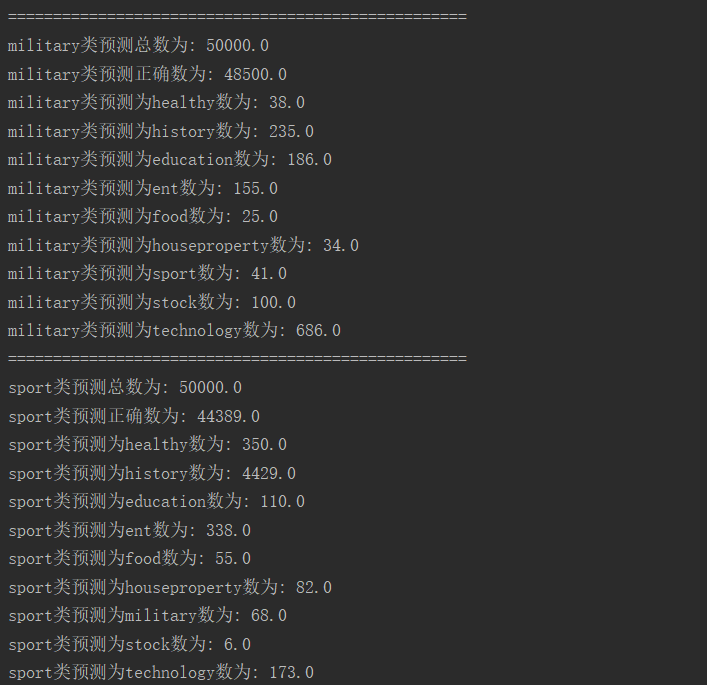


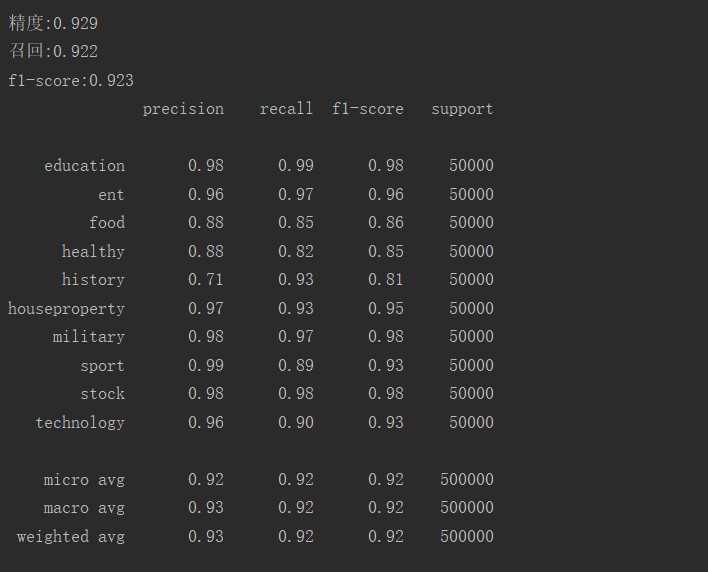
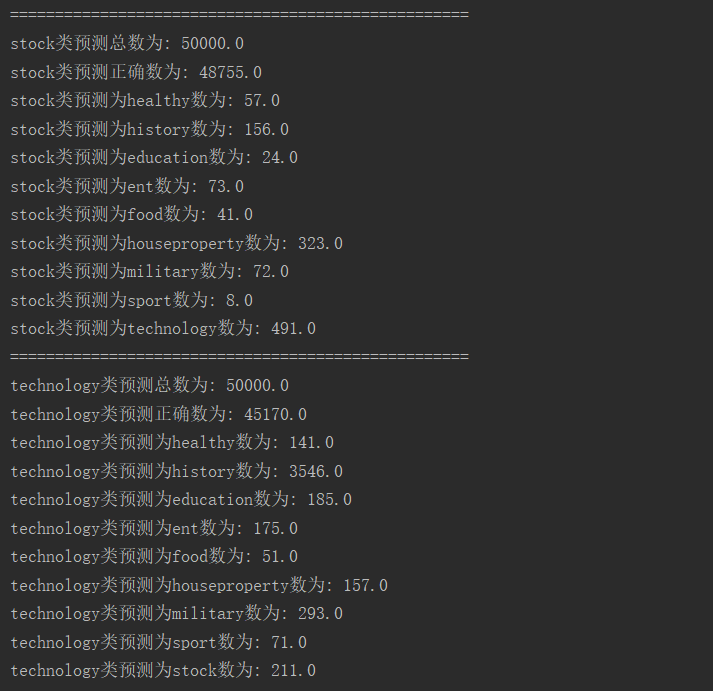


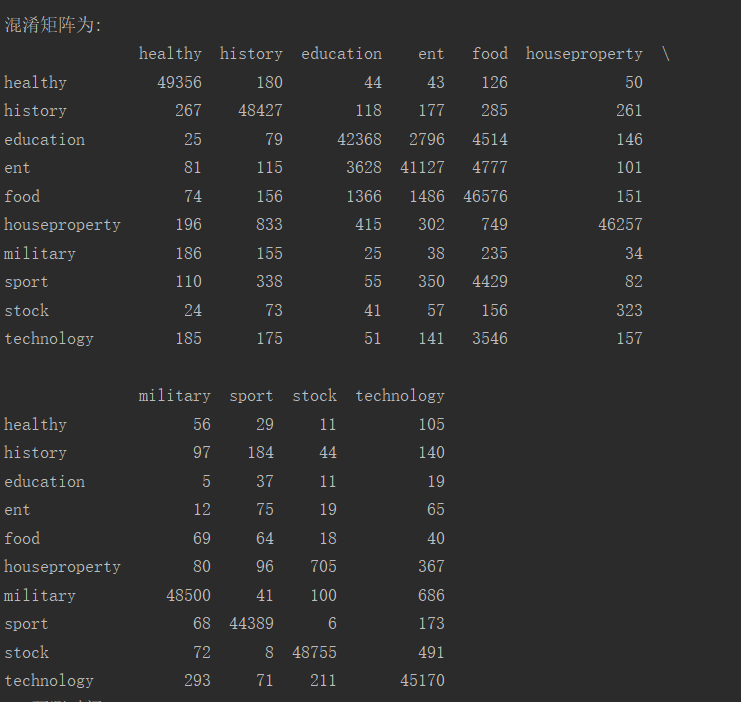
支持向量机：











# 实验过程遇到的问题及解决办法

1. 一开始写程序的时候程序总是超时、效率不高，分析了超时的原因后，决定把程序拆分为两个部分，一个部分爬取新闻的链接，另一个部分爬取新闻的内容，这样就解决了程序超时的原因，同时也挺高了程序的效率。
2. 训练模型过程中程序运行时间太长，模型调优需要通过调整参数来完成，如果每次都从头开始运行程序的话会浪费大量时间，所以将程序运行的每一步比如说数据预处理、生成特征向量、训练得到的模型产生的结果保存，可以避免每次从头开始运行程序而节省大量时间。

# 实验总结

通过本次实验，我们初步了解了通过机器学习来训练一个分类模型的方法，即构造数据集、通过特征提取得到特征向量、通过机器学习算法训练模型、测试模型的泛化能力。了解了文本特征提取的方法有卡方值和TF-IDF值等方法。了解了朴素贝叶斯和SVM的基本原理。了解了特征向量降维和模型调参的基本方法。

网络爬虫代码：

1.爬取新闻的链接。

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.support import expected\_conditions as EC

from selenium.webdriver.support.wait import WebDriverWait

from selenium.common.exceptions import TimeoutException

from pyquery import PyQuery as pq

import pymysql

browser = webdriver.Chrome()

wait = WebDriverWait(browser, 10)

# 进入爬取页面

def search():

try:

url = 'http://smart.huanqiu.com/travel/'

browser.get(url)

wait.until(EC.presence\_of\_element\_located((By.CLASS\_NAME, 'boxCon')))

getDetail()

total = wait.until(EC.presence\_of\_element\_located((By.CSS\_SELECTOR, '.pageBox > div > a:nth-child(13) ')))

return total.text

except TimeoutError:

return search()

# 得到具体信息

def getDetail():

html = pq(browser.page\_source, parser="html")

content = html.find('.boxCon > .fallsFlow')

uls = content.find('ul').items()

conn = pymysql.connect(host="127.0.0.1", user="root", passwd="123456", db="datamining",

charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor)

cur = conn.cursor()

for ul in uls:

lis = ul('li').items()

for li in lis:

news = {

'title': li.find('h3 > a').text(),

'href': li.find('a').attr('href'),

'time': li.find('h6').text()

}

try:

temp = (news['title'], news['href'], news['time'],)

sql = "insert into technology\_news\_new(title,href,time) values(%s,%s,%s)"

cur.execute(sql, temp)

cur.connection.commit()

# print(news)

except Exception as error:

print(error)

conn.close()

# 爬取下一页

def next\_detail(page\_number):

try:

nextBotton = wait.until(EC.element\_to\_be\_clickable((By.CSS\_SELECTOR, '.pageBox > div > a:last-child')))

nextBotton.click()

wait.until(EC.text\_to\_be\_present\_in\_element((By.CSS\_SELECTOR, '.pageBox > div > em'), str(page\_number)))

getDetail()

except TimeoutException:

next\_detail(page\_number)

def main():

try:

total = search()

total = int(total)

print(total)

for i in range(2, total + 1):

next\_detail(i)

except Exception:

print(Exception)

finally:

browser.close()

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

main()

2.根据新闻链接访问新闻,并将新闻写到数据库中:

import pymysql

import requests

from lxml import etree

conn = pymysql.connect(host="127.0.0.1", user="root", passwd="123456", db="datamining",

charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor)

cur = conn.cursor()

sql = "select href from technology\_news\_new"

cur.execute(sql)

results = cur.fetchall()

for row in results:

try:

response = requests.get(row['href'])

response.encoding = response.apparent\_encoding

root = etree.HTML(response.content.decode('utf-8', 'ignore'))

content = ''.join(root.xpath("//div[@class='la\_con']/p/text()"))

#print(content)

sql = "insert into technology(content) values(%s)"

cur.execute(sql, content)

cur.connection.commit()

except:

print("Error1")

conn.close()

数据清洗和分词：

# -\*- coding: utf-8 -\*-  
import sys  
import re  
import jieba.posseg  
import os  
import MySQLdb  
reload(sys)  
sys.setdefaultencoding('utf-8')  
  
def savefile(dirpath,filepath,content):  
  
 if os.path.exists(dirpath)==False:  
 os.makedirs(dirpath)  
  
 savepath = os.path.join(dirpath,filepath)  
 print(savepath)  
 with open(savepath,"w") as fp:  
 fp.write(content)  
 fp.close()  
  
def function(datalist,path):  
 stopwords = []  
 for word in open('stop\_words\_ch.txt', 'r'):  
 stopwords.append(word.strip())  
  
 db = MySQLdb.connect("localhost", "root", "root", "datamining", charset='utf8')  
 cursor = db.cursor()  
  
 i = 200001  
  
 for item in datalist:  
 sql = "select content from " + item  
 cursor.execute(sql)  
 results = cursor.fetchall()  
  
 for row in results:  
 content = []  
 temptest = re.sub("[@·《》、.%，。？“”（）：(\u3000)(\xa0)！… ；▼]|[a-zA-Z0-9]|['月''日''年']", "", row['content'])  
 words = jieba.posseg.cut(temptest)  
 for w in words:  
 if w.word not in stopwords and w.flag == 'n':  
 content.append(w.word)  
  
 forpath = path + "/" + item.replace("\_even","").replace("\_odd","")  
 subpath = str(i) +".txt"  
 savefile(forpath,subpath," ".join(content))  
 i = i+1  
 conn.close()  
  
if \_\_name\_\_=="\_\_main\_\_":  
   
test = ['education\_odd', 'ent\_odd','food\_odd', 'healthy\_odd','history\_odd', 'houseproperty\_odd','military\_odd', 'sport\_odd','tock\_odd', 'technology\_odd']  
path = ""  
function(test, path)  
# train = ['education\_even,ent\_even,food\_even,healthy\_even,history\_even,houseproperty\_even,military\_even,sport\_even,stock\_even,technology\_even]  
# path = ""  
# function(train, path)

朴素贝叶斯分类器代码：

# -\*- coding: utf-8 -\*-  
import os  
import datetime  
import sys  
import pickle  
from sklearn.feature\_extraction.text import TfidfVectorizer  
from sklearn.datasets.base import Bunch  
from sklearn import svm  
from sklearn.metrics import classification\_report  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn import metrics  
from sklearn.feature\_selection import SelectKBest  
from sklearn.feature\_selection import chi2  
import numpy as np  
import pandas as pd  
from sklearn.metrics import confusion\_matrix  
  
reload(sys)  
sys.setdefaultencoding('utf-8')  
  
  
class TF\_IDF:  
 data\_path = "/Users/wangyuhui/Desktop/dataminingData/train/"  
 test\_path = "/Users/wangyuhui/Desktop/dataminingData/test/"  
 result\_path = "/Users/wangyuhui/Desktop/dataminingResult1/"  
 stop\_words\_path = "/Users/wangyuhui/Desktop/dataminingData/stop\_words\_ch.txt"  
 #class\_path = "../test\_res/"  
 stop\_words = []  
 class\_code = {"healthy":0,"history":1,"education":2,"ent":3,  
 "food":4,"houseproperty":5,"military":6,"sport":7,  
 "stock":8,"technology":9}  
  
 def \_\_init\_\_(self):  
 self.ConfusionMatrix = np.zeros([len(TF\_IDF.class\_code),len(TF\_IDF.class\_code)])  
 TF\_IDF.stop\_words = self.read\_file(TF\_IDF.stop\_words\_path).strip().split("\n")  
  
 def printList(self, mylist):  
 for item in mylist:  
 print item,  
 print  
  
 # 写文件  
 def save\_file(self, path, content):  
 with open(path, 'w') as f:  
 f.write(content)  
  
 # 读文件  
 def read\_file(self, path):  
 with open(path, 'r') as f:  
 return f.read()  
 #读取bunch文件  
 def readbunchobj(self,path):  
 with open(path, "rb") as file\_obj:  
 bunch = pickle.load(file\_obj)  
 return bunch  
  
 def writebunchobj(self,path, bunchobj):  
 with open(path, "wb") as file\_obj:  
 pickle.dump(bunchobj, file\_obj)  
  
 # 加载分词结果文件  
 def loadSegmentation(self,path):  
 begintime = datetime.datetime.now()  
 fileDocs = os.listdir(path)  
 print fileDocs  
 for item in fileDocs:  
 if item.startswith("."):  
 fileDocs.remove(item)  
 print fileDocs  
 bunch = Bunch(target\_name=[], label=[], filenames=[], contents=[])  
 bunch.target\_name.extend(fileDocs)  
 # 获取每个目录下所有的文件  
 for mydir in fileDocs:  
 class\_path = path + mydir + "/" # 拼出分类子目录的路径  
 file\_list = os.listdir(class\_path) # 获取class\_path下的所有文件  
 for file\_path in file\_list: # 遍历类别目录下文件  
 if file\_path.endswith("txt") == False:  
 continue  
 fullname = class\_path + file\_path # 拼出文件名全路径  
 bunch.label.append(mydir)  
 bunch.contents.append(self.read\_file(fullname)) # 读取文件内容  
 bunch.filenames.append(mydir+"/"+file\_path)  
 self.writebunchobj(TF\_IDF.result\_path+"trainData.dat",bunch)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "训练bunch:contents长度、label长度:",len(bunch.contents),len(bunch.label)  
 print "训练数据保存完成,所花费时间为",span.seconds  
 def loadTestData(self, path):  
 begintime = datetime.datetime.now()  
 fileDocs = os.listdir(path)  
 print fileDocs  
 for item in fileDocs:  
 if item.startswith("."):  
 fileDocs.remove(item)  
 print fileDocs  
 bunch = Bunch(target\_name=[], label=[], filenames=[], contents=[])  
 bunch.target\_name.extend(fileDocs)  
 # 获取每个目录下所有的文件  
 for mydir in fileDocs:  
 class\_path = path + mydir + "/" # 拼出分类子目录的路径  
 file\_list = os.listdir(class\_path) # 获取class\_path下的所有文件  
 for file\_path in file\_list: # 遍历类别目录下文件  
 if file\_path.endswith("txt") == False:  
 continue  
 fullname = class\_path + file\_path # 拼出文件名全路径  
 bunch.label.append(mydir)  
 bunch.contents.append(self.read\_file(fullname)) # 读取文件内容  
 bunch.filenames.append(mydir + "/" + file\_path)  
 self.writebunchobj(TF\_IDF.result\_path + "testData.dat", bunch)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "测试bunch:contents长度、label长度:", len(bunch.contents), len(bunch.label)  
 print "测试数据保存完成,所花费时间为",span.seconds  
  
 def calculateTFIDF(self,train\_tfidf\_path,bunch\_path,tfidf\_path):  
 begintime = datetime.datetime.now()  
 bunch = self.readbunchobj(bunch\_path)  
 tfidfspace = Bunch(target\_name=bunch.target\_name, label=bunch.label, filenames=bunch.filenames, tfidfmetrix=[],  
 vocabulary={})  
  
 if train\_tfidf\_path is not None:  
 trainbunch = self.readbunchobj(train\_tfidf\_path)  
 tfidfspace.vocabulary = trainbunch.vocabulary  
 vectorizer = TfidfVectorizer(stop\_words=TF\_IDF.stop\_words, sublinear\_tf=True, max\_df=0.5,  
 vocabulary=trainbunch.vocabulary)  
 tfidfspace.tfidfmetrix = vectorizer.fit\_transform(bunch.contents)  
  
 else:  
 vectorizer = TfidfVectorizer(stop\_words=TF\_IDF.stop\_words, sublinear\_tf=True, max\_df=0.5)  
 tfidfspace.tfidfmetrix = vectorizer.fit\_transform(bunch.contents)  
 tfidfspace.vocabulary = vectorizer.vocabulary\_  
 self.writebunchobj(TF\_IDF.result\_path + "myvocabulary.dat", vectorizer.vocabulary\_)  
 # ch2 = SelectKBest(chi2, k=130000)  
 # train\_X = ch2.fit\_transform()  
  
 self.writebunchobj(tfidf\_path, tfidfspace)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 if train\_tfidf\_path is not None:  
 print "生成测试tf-idf矩阵,所花费时间为",span.seconds  
 else:  
 print "生成训练tf-idf矩阵,所花费时间为",span.seconds  
  
 def metrics\_result(self,actual, predict):  
 print('精度:{0:.3f}'.format(metrics.precision\_score(actual, predict, average='weighted')))  
 print('召回:{0:0.3f}'.format(metrics.recall\_score(actual, predict, average='weighted')))  
 print('f1-score:{0:.3f}'.format(metrics.f1\_score(actual, predict, average='weighted')))  
 print(classification\_report(testSet.label, predict))  
  
 def trainProcess(self,trainSet):  
 print "朴素贝叶斯训练过程开始:"  
 begintime = datetime.datetime.now()  
 # 训练分类器：输入词袋向量和分类标签，alpha:0.001 alpha越小，迭代次数越多，精度越高  
 clf = MultinomialNB(alpha=0.001).fit(trainSet.tfidfmetrix, trainSet.label)  
 with open(TF\_IDF.result\_path+"MultinomialNBModel.dat", "wb") as file\_obj:  
 pickle.dump(clf, file\_obj)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "朴素贝叶斯训练时间:",span.seconds  
  
 def SVMProcess(self, trainSet):  
 print "SVM训练过程开始:"  
 begintime = datetime.datetime.now()  
 clf = svm.SVC(C=1.0, cache\_size=800, class\_weight=None, coef0=0.0, decision\_function\_shape='ovr', degree=3,  
 gamma='auto', kernel='linear', max\_iter=-1, probability=False, random\_state=None, shrinking=True,  
 tol=0.001, verbose=False)  
 clf.fit(trainSet.tfidfmetrix, trainSet.label)  
 with open("/Users/wangyuhui/Desktop/svmResult/svmModel.dat", "wb") as file\_obj:  
 # with open(TF\_IDF.result\_path + "svmModel.dat", "wb") as file\_obj:  
 pickle.dump(clf, file\_obj)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "SVM训练时间:", span.seconds  
 test\_tfidf\_path = TF\_IDF.result\_path + "test\_tfidf\_svm.dat"  
 testSet = TF\_IDF.readbunchobj(test\_tfidf\_path)  
 self.predictProcess("/Users/wangyuhui/Desktop/svmResult/svmModel.dat",testSet)  
  
 def predictProcess(self,modelPath,testSet):  
 if modelPath is None:  
 print "模型路径未知"  
 return  
 clf = self.readbunchobj(modelPath)  
 errorFile = "MultinomialNBerror.txt"  
 if modelPath.find("svm") != -1:  
 errorFile = "SVMerror.txt"  
 print "加载模型成功"  
 # 预测分类结果  
 predicted = clf.predict(testSet.tfidfmetrix)  
 outPutList = []  
 for flabel, file\_name, expct\_cate in zip(testSet.label, testSet.filenames, predicted):  
 self.ConfusionMatrix[TF\_IDF.class\_code[flabel]][TF\_IDF.class\_code[expct\_cate]] += 1  
 if flabel != expct\_cate:  
 outPutList.append(file\_name + ": 实际类别:" + flabel + " -->预测类别:" + expct\_cate)  
 self.save\_file(TF\_IDF.result\_path + errorFile, "\n".join(outPutList))  
 print("预测完成")  
 self.showPredictResult()  
 self.metrics\_result(testSet.label, predicted)  
 print "混淆矩阵为:"  
 # 显示所有列  
 pd.set\_option('display.max\_columns', None)  
 # 显示所有行  
 pd.set\_option('display.max\_rows', None)  
 print pd.DataFrame(confusion\_matrix(testSet.label, predicted),  
 columns=["education", "ent", "food", "healthy", "history", "houseproperty", "military",  
 "sport", "stock", "technology"],  
 index=["education", "ent", "food", "healthy", "history", "houseproperty", "military",  
 "sport", "stock", "technology"])  
  
 def showPredictResult(self):  
 for i in range(len(self.ConfusionMatrix)):  
 keyWord = ""  
 for key in TF\_IDF.class\_code:  
 if TF\_IDF.class\_code[key] == i:  
 keyWord = key  
 break  
 print "==================================================="  
 print keyWord + "类预测总数为:", np.sum([self.ConfusionMatrix[i]])  
 print keyWord + "类预测正确数为:", self.ConfusionMatrix[i][i]  
 for j in range(len(self.ConfusionMatrix[0])):  
 if j == i:  
 continue  
 predictKey = ""  
 for key in TF\_IDF.class\_code:  
 if TF\_IDF.class\_code[key] == j:  
 predictKey = key  
 break  
 print keyWord+"类预测为"+predictKey+"数为:", self.ConfusionMatrix[i][j]  
 print "==================================================="  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 TF\_IDF = TF\_IDF()  
 # 步骤1  
 # TF\_IDF.loadSegmentation(TF\_IDF.data\_path)  
 # TF\_IDF.loadTestData(TF\_IDF.test\_path)  
  
 # 步骤2  
 # train\_tfidf\_path = TF\_IDF.result\_path+"train\_tfidf.dat"  
 # train\_bunch\_path = TF\_IDF.result\_path+"trainData.dat"  
 # TF\_IDF.calculateTFIDF(train\_tfidf\_path=None,bunch\_path=train\_bunch\_path,tfidf\_path=train\_tfidf\_path)  
  
 # test\_tfidf\_path = TF\_IDF.result\_path+"test\_tfidf.dat"  
 # test\_bunch\_path = TF\_IDF.result\_path+"testData.dat"  
 # TF\_IDF.calculateTFIDF(train\_tfidf\_path=train\_tfidf\_path, bunch\_path=test\_bunch\_path, tfidf\_path=test\_tfidf\_path)  
  
 # 步骤3  
 # train\_tfidf\_path = TF\_IDF.result\_path + "train\_tfidf.dat"  
 # trainSet = TF\_IDF.readbunchobj(train\_tfidf\_path)  
 # TF\_IDF.trainProcess(trainSet)  
  
 # 步骤4  
 test\_tfidf\_path = TF\_IDF.result\_path + "test\_tfidf.dat"  
 testSet = TF\_IDF.readbunchobj(test\_tfidf\_path)  
 MultinomialNBPath = TF\_IDF.result\_path + "MultinomialNBModel.dat"  
 TF\_IDF.predictProcess(MultinomialNBPath,testSet)  
  
 # 步骤2  
 # train\_tfidf\_path = TF\_IDF.result\_path + "train\_tfidf\_svm.dat"  
 # train\_bunch\_path = TF\_IDF.result\_path + "trainData.dat"  
 # TF\_IDF.calculateTFIDF(train\_tfidf\_path=None,bunch\_path=train\_bunch\_path,tfidf\_path=train\_tfidf\_path)  
  
 # test\_tfidf\_path = TF\_IDF.result\_path + "test\_tfidf\_svm.dat"  
 # test\_bunch\_path = TF\_IDF.result\_path + "testData.dat"  
 # #TF\_IDF.calculateTFIDF(train\_tfidf\_path=train\_tfidf\_path, bunch\_path=test\_bunch\_path, tfidf\_path=test\_tfidf\_path)  
  
 # 步骤3  
 # train\_tfidf\_path = TF\_IDF.result\_path + "train\_tfidf\_svm.dat"  
 # trainSet = TF\_IDF.readbunchobj(train\_tfidf\_path)  
 # TF\_IDF.SVMProcess(trainSet)  
  
 # 步骤4  
 # test\_tfidf\_path = "/Users/wangyuhui/Desktop/svmResult1/test\_tfidf\_svm.dat"  
 # testSet = TF\_IDF.readbunchobj(test\_tfidf\_path)  
 # svmPath = "/Users/wangyuhui/Desktop/forest/svmModel.dat"  
 # TF\_IDF.predictProcess(svmPath,testSet)

SVM分类器代码：

# -\*- coding: utf-8 -\*-  
import os  
import datetime  
import sys  
import pickle  
from sklearn.feature\_extraction.text import TfidfVectorizer, CountVectorizer  
from sklearn.datasets.base import Bunch  
from sklearn import svm  
from sklearn.metrics import classification\_report  
from sklearn.naive\_bayes import MultinomialNB  
from sklearn import metrics  
from sklearn.feature\_selection import SelectKBest  
from sklearn.feature\_selection import chi2  
import numpy as np  
import pandas as pd  
from sklearn.metrics import confusion\_matrix  
  
reload(sys)  
sys.setdefaultencoding('utf-8')  
  
  
class TF\_IDF:  
 # data\_path = "/Users/wangyuhui/Desktop/dataminingData/train/"  
 # test\_path = "/Users/wangyuhui/Desktop/dataminingData/test/"  
 # result\_path = "/Users/wangyuhui/Desktop/dataminingResult1/"  
 # stop\_words\_path = "/Users/wangyuhui/Desktop/dataminingData/stop\_words\_ch.txt"  
  
 data\_path = "/Users/wangyuhui/Desktop/dataminingData/train/"  
 test\_path = "/Users/wangyuhui/Desktop/dataminingData/test/"  
 # result\_path = "C:\\Users\\zo\\Desktop\\dataminingResult\\"  
 result\_path = "/Users/wangyuhui/Desktop/svmResult/"  
 stop\_words\_path = "/Users/wangyuhui/Desktop/dataminingData/stop\_words\_ch.txt"  
 #class\_path = "../test\_res/"  
 stop\_words = []  
 class\_code = {"healthy":0,"history":1,"education":2,"ent":3,  
 "food":4,"houseproperty":5,"military":6,"sport":7,  
 "stock":8,"technology":9}  
  
 def \_\_init\_\_(self):  
 self.ConfusionMatrix = np.zeros([len(TF\_IDF.class\_code),len(TF\_IDF.class\_code)])  
 TF\_IDF.stop\_words = self.read\_file(TF\_IDF.stop\_words\_path).strip().split("\n")  
  
 def printList(self, mylist):  
 for item in mylist:  
 print item,  
 print  
  
 # 写文件  
 def save\_file(self, path, content):  
 with open(path, 'w') as f:  
 f.write(content)  
  
 # 读文件  
 def read\_file(self, path):  
 with open(path, 'r') as f:  
 return f.read()  
 #读取bunch文件  
 def readbunchobj(self,path):  
 with open(path, "rb") as file\_obj:  
 bunch = pickle.load(file\_obj)  
 return bunch  
  
 def writebunchobj(self,path, bunchobj):  
 with open(path, "wb") as file\_obj:  
 pickle.dump(bunchobj, file\_obj)  
  
 # 加载分词结果文件  
 def loadSegmentation(self, path):  
 begintime = datetime.datetime.now()  
 fileDocs = os.listdir(path)  
 print fileDocs  
 for item in fileDocs:  
 if item.startswith("."):  
 fileDocs.remove(item)  
 print fileDocs  
 bunch = Bunch(target\_name=[], label=[], filenames=[], contents=[])  
 bunch.target\_name.extend(fileDocs)  
 # 获取每个目录下所有的文件  
 for mydir in fileDocs:  
 class\_path = path + mydir + "/" # 拼出分类子目录的路径  
 file\_list = os.listdir(class\_path) # 获取class\_path下的所有文件  
 for file\_path in file\_list: # 遍历类别目录下文件  
 if file\_path.endswith("txt") == False:  
 continue  
 fullname = class\_path + file\_path # 拼出文件名全路径  
 bunch.label.append(mydir)  
 bunch.contents.append(self.read\_file(fullname)) # 读取文件内容  
 bunch.filenames.append(mydir + "/" + file\_path)  
 self.writebunchobj(TF\_IDF.result\_path + "trainData.dat", bunch)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "训练bunch:contents长度、label长度:", len(bunch.contents), len(bunch.label)  
 print "训练数据保存完成,所花费时间为", span.seconds  
 def loadTestData(self, path):  
 begintime = datetime.datetime.now()  
 fileDocs = os.listdir(path)  
 print fileDocs  
 for item in fileDocs:  
 if item.startswith("."):  
 fileDocs.remove(item)  
 print fileDocs  
 bunch = Bunch(target\_name=[], label=[], filenames=[], contents=[])  
 bunch.target\_name.extend(fileDocs)  
 # 获取每个目录下所有的文件  
 for mydir in fileDocs:  
 class\_path = path + mydir + "/" # 拼出分类子目录的路径  
 file\_list = os.listdir(class\_path) # 获取class\_path下的所有文件  
 for file\_path in file\_list: # 遍历类别目录下文件  
 if file\_path.endswith("txt") == False:  
 continue  
 fullname = class\_path + file\_path # 拼出文件名全路径  
 bunch.label.append(mydir)  
 bunch.contents.append(self.read\_file(fullname)) # 读取文件内容  
 bunch.filenames.append(mydir + "/" + file\_path)  
 self.writebunchobj(TF\_IDF.result\_path + "testData.dat", bunch)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "测试bunch:contents长度、label长度:", len(bunch.contents), len(bunch.label)  
 print "测试数据保存完成,所花费时间为",span.seconds  
  
 def calculateTFIDF(self,train\_tfidf\_path,bunch\_path,tfidf\_path):  
 begintime = datetime.datetime.now()  
 bunch = self.readbunchobj(bunch\_path)  
 tfidfspace = Bunch(target\_name=bunch.target\_name, label=bunch.label, filenames=bunch.filenames, tfidfmetrix=[],  
 vocabulary={})  
  
 if train\_tfidf\_path is not None:  
 trainbunch = self.readbunchobj(train\_tfidf\_path)  
 tfidfspace.vocabulary = trainbunch.vocabulary  
 vectorizer = TfidfVectorizer(stop\_words=TF\_IDF.stop\_words, sublinear\_tf=True, max\_df=0.5,  
 vocabulary=trainbunch.vocabulary)  
 tfidfspace.tfidfmetrix = vectorizer.fit\_transform(bunch.contents)  
  
 else:  
 vectorizer = TfidfVectorizer(stop\_words=TF\_IDF.stop\_words, sublinear\_tf=True, max\_df=0.5)  
 tfidfspace.tfidfmetrix = vectorizer.fit\_transform(bunch.contents)  
 tfidfspace.vocabulary = vectorizer.vocabulary\_  
 self.writebunchobj(TF\_IDF.result\_path+"myvocabulary.dat", vectorizer.vocabulary\_)  
 # ch2 = SelectKBest(chi2, k=130000)  
 # train\_X = ch2.fit\_transform()  
  
 self.writebunchobj(tfidf\_path, tfidfspace)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 if train\_tfidf\_path is not None:  
 print "生成测试tf-idf矩阵,所花费时间为",span.seconds  
 else:  
 print "生成训练tf-idf矩阵,所花费时间为",span.seconds  
  
 def metrics\_result(self,actual, predict):  
 print('精度:{0:.3f}'.format(metrics.precision\_score(actual, predict, average='weighted')))  
 print('召回:{0:0.3f}'.format(metrics.recall\_score(actual, predict, average='weighted')))  
 print('f1-score:{0:.3f}'.format(metrics.f1\_score(actual, predict, average='weighted')))  
 print(classification\_report(testSet.label, predict))  
  
 def trainProcess(self,trainSet):  
 print "朴素贝叶斯训练过程开始:"  
 begintime = datetime.datetime.now()  
 # 训练分类器：输入词袋向量和分类标签，alpha:0.001 alpha越小，迭代次数越多，精度越高  
 clf = MultinomialNB(alpha=0.001).fit(trainSet.tfidfmetrix, trainSet.label)  
 with open(TF\_IDF.result\_path+"MultinomialNBModel.dat", "wb") as file\_obj:  
 pickle.dump(clf, file\_obj)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "朴素贝叶斯训练时间:",span.seconds  
  
 def SVMProcess(self, trainSet):  
 print "SVM训练过程开始:"  
 begintime = datetime.datetime.now()  
 clf = svm.SVC(C=1.0, cache\_size=800, class\_weight=None, coef0=0.0, decision\_function\_shape='ovr', degree=3,  
 gamma='auto', kernel='linear', max\_iter=-1, probability=False, random\_state=None, shrinking=True,  
 tol=0.001, verbose=False)  
 clf.fit(trainSet.tfidfmetrix, trainSet.label)  
 with open(TF\_IDF.result\_path+"svmModel.dat", "wb") as file\_obj:  
 # with open(TF\_IDF.result\_path + "svmModel.dat", "wb") as file\_obj:  
 pickle.dump(clf, file\_obj)  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "SVM训练时间:", span.seconds  
  
 def predictProcess(self,modelPath,testSet,predictedResult):  
 begintime = datetime.datetime.now()  
 if modelPath is None:  
 print "模型路径未知"  
 return  
 clf = self.readbunchobj(modelPath)  
 errorFile = "MultinomialNBerror.txt"  
 if modelPath.find("svm") != -1:  
 errorFile = "SVMerror.txt"  
 print "加载模型成功"  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 begintime = endtime  
 print "加载模型时间:", span.seconds  
 predicted = predictedResult  
 # 预测分类结果  
 if predictedResult is None:  
 predicted = clf.predict(testSet.tfidfmetrix)  
 self.writebunchobj(TF\_IDF.result\_path + "predictedresult.dat", predicted)  
 outPutList = []  
 for flabel, file\_name, expct\_cate in zip(testSet.label, testSet.filenames, predicted):  
 self.ConfusionMatrix[TF\_IDF.class\_code[flabel]][TF\_IDF.class\_code[expct\_cate]] += 1  
 if flabel != expct\_cate:  
 outPutList.append(file\_name + ": 实际类别:" + flabel + " -->预测类别:" + expct\_cate)  
 self.save\_file(TF\_IDF.result\_path + errorFile, "\n".join(outPutList))  
 print("预测完成")  
 self.showPredictResult()  
 self.metrics\_result(testSet.label, predicted)  
 print "混淆矩阵为:"  
 # 显示所有列  
 pd.set\_option('display.max\_columns', None)  
 # 显示所有行  
 pd.set\_option('display.max\_rows', None)  
 print pd.DataFrame(confusion\_matrix(testSet.label, predicted),  
 columns=["healthy", "history", "education", "ent", "food", "houseproperty", "military",  
 "sport", "stock", "technology"],  
 index=["healthy", "history", "education", "ent", "food", "houseproperty","military","sport","stock","technology"])  
 endtime = datetime.datetime.now()  
 span = endtime - begintime  
 print "SVM预测时间:", span.seconds  
  
 def showPredictResult(self):  
 for i in range(len(self.ConfusionMatrix)):  
 keyWord = ""  
 for key in TF\_IDF.class\_code:  
 if TF\_IDF.class\_code[key] == i:  
 keyWord = key  
 break  
 print "==================================================="  
 print keyWord + "类预测总数为:", np.sum([self.ConfusionMatrix[i]])  
 print keyWord + "类预测正确数为:", self.ConfusionMatrix[i][i]  
 for j in range(len(self.ConfusionMatrix[0])):  
 if j == i:  
 continue  
 predictKey = ""  
 for key in TF\_IDF.class\_code:  
 if TF\_IDF.class\_code[key] == j:  
 predictKey = key  
 break  
 print keyWord+"类预测为"+predictKey+"数为:", self.ConfusionMatrix[i][j]  
 print "==================================================="  
  
 def TestDemo(self,vocabulary,clf):  
 healthy = "因素 肝癌 风险 乙型肝炎"  
 food = "孩子 饭菜 家常菜 豆腐"  
 education = "流感 步伐 流感 疑似病例 学生家长 电话 程度 流感 想象 流感 病例 人数 人 流感病毒 学生家长 情况 流感病毒 流感病毒 毒性 病毒 时期 平均温度 摄氏度 温度 病毒传播 高峰期 流感病毒 能力 流感病毒 情况 医疗 部门 疾病 控制能力 病例 病例 医疗 条件 技术 程度 留学生 流感病毒 时期 学生 签证官 学生 成功率 情况 时期 国家 情况 留学生 进程 反观 流感病毒 事件 全球 疫情 国家 全球 疑似病例 国家 病例 数 留学生 国家 情况 学生 开学 个的 事件 政府 流感病毒 流感病毒 疑似病例 患者 大学 学校 校区 学校 患者 校区 校区 病例 个体 案例 公司 公司 办理 名 学生家长 学生 猪 流感 计划 家长 公司 学生家长 计划 家长 入学 关键时刻 孩子 学期 公司 办理 世界 留学生 情况 猪 流感 常识 家长 疫情 地区 留学生 学员 传染性 疾病 常识"  
 ent = "名状 海角 名单 评委 名状 图 影片 名单 电影 名状 项 赢家 海角 剧情片 男配角 新人 项 电影 海水 火焰 剧情片 男女 主角 赤壁 男配角 项 奖项 电影票房 名单 男孩 降风 原著 剧本 男孩 剧情片 女配角 演员 女主角 光芒 女星 首度 有缘 首度 移师 大使 女主角 男主角 入围者 实力 同场竞技 精彩 硬仗 演员 电影 海角 同片 局面 男孩 骗子 众人 短片 格子 群雄 歌坛 大 灌篮 主题曲 原创 电影 歌曲 电影 工作者 海角 电影票房 新台币 电影 新气象 本土 感情 代表 人物 发迹 脱俗 亮眼 国际 男星 光 电影 资深 灯光师 厂 电影 后辈 奖项 名状 共项 男主角 剧情片 视觉效果 美术设计 造型 动作 原著 剧本 原创 电影 音乐 音效 剪辑 摄影 海角 共项 剧情片 演员 男配角 原创 电影 歌曲 音效 摄影 电影 共项 剧情片 男主角 剧本 动作 视觉效果 音效 海水 火焰 共项 男主角 女主角 剧情片 美术设计 摄影 赤壁 共项 男配角 视觉效果 造型 美术设计 男孩 共项 演员 女配角 原著 剧本 剧情片 事儿 共项 男配角 剧本 造型"  
 history = "小白菜 案 刑部 尚书"  
 houseproperty = "外观 图 风格 印花 餐具 外观 涂鸦 花卉 图案 特色 线条 气质 厨房 格调 特质"  
 military = "对岸 空军 装备 战斗机 军迷 陌生 世纪 对岸 战斗机 型号 老式 性能 盟国 天线 空空导弹 全天候 目标 吊舱 水平 机群 大陆 空军 劲敌 力量对比 大陆 空军 麾下 阵容 对岸 风光 无机 窘境 国际形势 实力 对岸 战斗机 可能性 短期内 现实 机群 对岸 量身 套装 服役 本份 对岸 空军 战斗机 时代 媒体 技术 成果 整体 性能 大陆 空军 级别 机型 服役 改进型 战斗机 单发 中型机 问世 军队 地位 小个子 空战 武器 对岸 空军 顶尖 机型 田忌赛马 对岸 马 大陆 空军 马 对岸 航电 有源 相控阵 航电 系统 技术 原型 能力 航电 功率 逊色 空战 武器 装备 头盔 显示器 基本上 射程 性能 机动性 发动机 发动机 空重 发动机 空重 航电 系统 机身 结构 优势 鸭式 布局 能力 稳盘 能力 机 后者 机体 状态 机龄 新机 起跑线 性能 力压 对岸 利器 大陆 人 事实 大度 心理 事实 作者 利刃 刻雨 无痕 内容 军事 官方 公众 军事 官方 栏目 文章 目的 信息 代表 观点 真实性 凡本网 版权所有 作品版权 作者 版权 原作者 人 本网 作者 利用 方式 作品 军事 军 军事 门户"  
 sport = "爵士 胖友 距离 差分 哥 福 分差 爵士队 净输分 全场 爵士队 下半场 队史 纪录 爵士 人 队史 赢球 差 纪录 爵士 主场 专业 战 底线 手感 价格公道 问 分差 数据 小 均分 手下留情 人 数据 罚球 命中率 图 图 腿 打篮球 腿 帅气 后仰 腿 大 木桩 区别 近景 球员 镜头 印象 毛钱"  
 stock = "经济 数据 财经 消息 汇率 跌势 原因 政府 报告 生产者 汇率 元 跌幅 汇率 涨幅 汇率 跌幅 报告 称份 经济学家 报告 称份 生产者 国际 货币 汇率 均告 过度 理由 货币 汇率 原因 股市 投资者 银行 证券公司 外汇 策略师 政府 报告 投资者 信心 重创 投资者 风险 情绪 跌幅 原因 抗议者 政府部门 水平 汇率 涨幅 汇率 涨幅 原因 股市 市场 投资者 利差 交易 交易 投资者 利率 国家 贷款 地区 收益 资产"  
 technology = "电脑网 游戏 奖项 游戏 奖项 游戏 刺客 信条 战神 蜘蛛侠 怪物 猎人 世界 荒野 镖客 游戏 名单 玩家 荒野 镖客 荒野 镖客 荒野 镖客 游戏 言论 荒野 镖客 游戏 玩家 分量 荒野 镖客 星 游戏 游戏 历时 花费 剧情 系统 画面 游戏 机构 玩家 游戏 大作 游戏 游戏机 电视 游玩 程度 电视 画质 性能 游戏 厂商 事情 高端 电视 荒野 镖客 高端 游戏 画面 外媒 荒野 镖客 画面 事情 游戏 原生 效果 外媒 荒野 镖客 游戏 屏幕 绘制 光谱 图 记录 亮度 光谱 颜色 图像 颜色 水平 动态 颜色 基本上 图像 灰色 粉红色 内容 红色 橙色 结论 荒野 镖客 效果 玩家 选项 感觉 画面 星 玩家 效果 游戏 画质 整体 口碑 销量 游戏 笔者 朋友 朋友 游戏 玩家 游戏 总向 感觉 游戏 画质 电视 玩游戏 画质 电视 笔者 游戏 画质 游戏 画质 引擎 游戏机 机能 电视 画质 游戏 老 电视 玩游戏 游戏 画面 电视 锅 电影 同理 劲 片源 体验 锅 片源 电视 力 技术 画面 视频 技术 经历 漫长 视频 画质 分辨率 深灰 色域 空间 亮度 动态 基础 水平 时代 所幸 动态 市面上 平板电视 标准 电视 平板电视 标准 性能 产物 后者 基础 功能 静态 数据 性能 能力 门槛 公司 费用 电视 厂商 标准 创维 差异 技术 硬件 性能 企业 差距 图片 电视 影片 图 电视 影片 图 奇幻 图 整体 亮度 亮度 颜色 画面 面纱 人物 特写 差异 皮肤 颜色 大 逆光 画面 宽容度 人物 脸部皮肤 细节 细节 衣服 纹理 环境 亮度高 动态 实力 天空 云朵 逆光 飞机 暗部 细节 生机 效果 压倒性 优势 亮度 亮度 色彩 灰度 时代 想象 事物 内容 电影 游戏 电视 高标准 产品"  
 content = [healthy,food,education,ent,history,houseproperty,military,sport,stock,technology]  
 vectorizer = CountVectorizer(stop\_words=TF\_IDF.stop\_words,vocabulary=vocabulary)  
 content = vectorizer.fit\_transform(content)  
 #print content.toarray()  
 label = clf.predict(content)  
 print label  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 TF\_IDF = TF\_IDF()  
 # 步骤1  
 # TF\_IDF.loadSegmentation(TF\_IDF.data\_path)  
 # TF\_IDF.loadTestData(TF\_IDF.test\_path)  
  
 # 步骤2  
 # train\_tfidf\_path = TF\_IDF.result\_path + "train\_tfidf\_svm.dat"  
 # train\_bunch\_path = TF\_IDF.result\_path + "trainData.dat"  
 # TF\_IDF.calculateTFIDF(train\_tfidf\_path=None,bunch\_path=train\_bunch\_path,tfidf\_path=train\_tfidf\_path)  
  
 # test\_tfidf\_path = TF\_IDF.result\_path + "test\_tfidf\_svm.dat"  
 # test\_bunch\_path = TF\_IDF.result\_path + "testData.dat"  
 # TF\_IDF.calculateTFIDF(train\_tfidf\_path=train\_tfidf\_path, bunch\_path=test\_bunch\_path, tfidf\_path=test\_tfidf\_path)  
  
 # 步骤3  
 # train\_tfidf\_path = TF\_IDF.result\_path + "train\_tfidf\_svm.dat"  
 # trainSet = TF\_IDF.readbunchobj(train\_tfidf\_path)  
 # TF\_IDF.SVMProcess(trainSet)  
  
 # 步骤4  
 predictedResultPath = TF\_IDF.result\_path + "predictedresult.dat"  
 test\_tfidf\_path = TF\_IDF.result\_path + "test\_tfidf\_svm.dat"  
 testSet = TF\_IDF.readbunchobj(test\_tfidf\_path)  
 svmPath = TF\_IDF.result\_path + "svmModel.dat"  
 predictedResult = TF\_IDF.readbunchobj(predictedResultPath)  
 TF\_IDF.predictProcess(svmPath,testSet,predictedResult)  
  
 # 步骤5  
 vocabulary\_path = TF\_IDF.result\_path + "myvocabulary.dat"  
 vocabulary = TF\_IDF.readbunchobj(vocabulary\_path)  
 svmPath = TF\_IDF.result\_path + "svmModel.dat"  
 clf = TF\_IDF.readbunchobj(svmPath)  
 TF\_IDF.TestDemo(vocabulary,clf)

附：

程序地址：<https://github.com/tianYingDao/TextCategorization>