1. **文本分类问题**

随着互联网的发展，非结构化的文本数据急剧增加，在为人们提供更多可用信息的同时，也导致人们更难从中发现自己最感兴趣的信息，也就是说，信息爆炸导致了信息迷航，因此，如何从海量的信息中挖掘出重要的信息具有非常高的研究价值和实际意义 。

其实，说白了就是人想知道这个文档是做什么的。首先给每篇文章一个标签、构建文档的特征，然后通过机器学习算法来学习特征和标签之间的映射关系，最后对未知的文本进行标签的预测。

在理解文本信息的问题上，由于信息量的庞大，如果仅凭人工方式来收集和挖掘文本数据，不仅需要消耗大量的人力和时间，而且也是很难实现的。于是，实现自动文本分类就显得尤其重要，它是文本信息挖掘的基本功能，也成为了处理和组织文本数据的核心技术。

目前我们所说的文本分类是通过机器学习的方式对文本进行分类。文本分类的应用相当广泛。主要有以下几种应用：

1. 垃圾邮件或短信的过滤
2. 社会舆论
3. 情感分析
4. 新闻分类（本文主要内容）

对于网站来说，可以根据你看得较多的新闻类别给你推荐新闻；对于用户来说，则是可以忽略掉不感兴趣的分类，提高了浏览体验。

智能信息推荐，因其可以广泛的应用到各种信息系统中，有效解决面对海量信息的尴尬，以及建立起适应性的个性化的应用系统拥有有效的文本分类方法，以建立智能推荐系统，使其可以根据用户的个人兴趣来定位并推荐相关的新闻资料。

1. **题目分析**

**文本分类问题：** 给定文档p（可能含有标题t），将文档分类为n个类别中的一个或多个

**文本分类方向：** 主要有二分类，多分类，多标签分类

文本分类方法： 传统机器学习方法（贝叶斯，svm等），深度学习方法（fastText，TextCNN等） 在本次实验中使用了 贝叶斯 + scikit-learn

本实验中新闻数据集的来源为：[https://www.sogou.com/labs/resource/list\_news.php](https://www.sogou.com/labs/resource/list_news.php*)（搜狗实验室）内容涉及到

C000008 财经

C000010 IT

C000013 健康

C000014 体育

C000016 旅游

C000020 教育

C000022 招聘

C000023 文化

C000024 军事

等不同的新闻文本

本次实验的数据格式为 txt 文本文件，使用了统一utf-8的编码方式，避免格式出现错误。

**本文的思路：** 本文主要介绍文本分类的处理过程，主要哪些方法，重点关注什么问题，对于不同的场景应该采用什么方法。

分类问题包括学习和分类两个过程，学习过程的目标是根据已知的训练数据构建分类模型，得到分类器；分类过程的任务是利用学习得到的分类器，预测新数据实例的类标号。

1. **实验过程**

**环境：**

Anaconda 集成 python 3.6.5

主要使用了 jieba、numpy、sklearn、wordcloud、matplotlib等库

**数据预处理：**

分词： 中文任务分词必不可少，一般使用jieba库进行分词（pip install jieba），工业界的翘楚。

去停用词：建立停用词字典，目前停用词字典有2000个左右，停用词主要包括一些副词、形容词及其一些连接词。通过维护一个停用词表，实际上是一个特征提取的过程，本质上是特征选择的一部分。（文件中停词为stopwords\_cn.txt）

词性标注： 在分词后判断词性（动词、名词、形容词、副词…），在使用jieba分词的时候设置参数就能获取。

预处理过程详细代码：

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

folder\_list = os.listdir(folder\_path) # 查看folder\_path下的文件

data\_list = [] # 数据集数据

class\_list = [] # 数据集类别

# 遍历每个子文件夹

for folder in folder\_list:

new\_folder\_path = os.path.join(folder\_path, folder) # 根据子文件夹，生成新的路径

files = os.listdir(new\_folder\_path) # 存放子文件夹下的txt文件的列表

j = 1

# 遍历每个txt文件

for file in files:

if j > 500: # 每类txt样本数最多1000个

break

try:

with open(os.path.join(new\_folder\_path, file), 'rb+') as f: # 打开txt文件

raw = f.read()

encoding = detect(raw)['encoding']

raw = raw.decode(encoding).encode('utf8')

word\_cut = jieba.cut(raw, cut\_all=False) # 精简模式，返回一个可迭代的generator

word\_list = list(word\_cut) # generator转换为list

data\_list.append(word\_list) # 添加数据集数据

class\_list.append(folder) # 添加数据集类别

j += 1

except:

pass

data\_class\_list = list(zip(data\_list, class\_list)) # zip压缩合并，将数据与标签对应压缩

random.shuffle(data\_class\_list) # 将data\_class\_list乱序

index = int(len(data\_class\_list) \* test\_size) + 1 # 训练集和测试集切分的索引值

train\_list = data\_class\_list[index:] # 训练集

test\_list = data\_class\_list[:index] # 测试集

train\_data\_list, train\_class\_list = zip(\*train\_list) # 训练集解压缩

test\_data\_list, test\_class\_list = zip(\*test\_list) # 测试集解压缩

all\_words\_dict = {} # 统计训练集词频

for word\_list in train\_data\_list:

for word in word\_list:

if word in all\_words\_dict.keys():

all\_words\_dict[word] += 1

else:

all\_words\_dict[word] = 1

# 根据键的值倒序排序

all\_words\_tuple\_list = sorted(all\_words\_dict.items(), key=lambda f: f[1], reverse=True)

all\_words\_list, all\_words\_nums = zip(\*all\_words\_tuple\_list) # 解压缩

all\_words\_list = list(all\_words\_list) # 转换成列表

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

函数说明:读取文件里的内容，并去重

def MakeWordsSet(words\_file):

words\_set = set() # 创建set集合

with open(words\_file, 'r', encoding='utf-8') as f: # 打开文件

for line in f.readlines(): # 一行一行读取

word = line.strip() # 去回车

if len(word) > 0: # 有文本，则添加到words\_set中

words\_set.add(word)

return words\_set # 返回处理结果

**算法的选择过程：**



引入预训练的 word2vec 模型会给训练带来好处，具体来说：

（1）间接引入外部训练数据，防止过拟合；

（2）减少需要训练的参数个数，提高训练效率

LSTM 需要训练的参数个数远小于 CNN，但训练时间大于 CNN。CNN 在分类问题的表现上一直很好，无论是图像还是文本；而想让 LSTM 优势得到发挥，首先让训练数据量得到保证

将单词在 word2vec 中的词向量加和求平均获得整个句子的语义向量的方法看似 naive 有时真挺奏效，当然仅限于短句子，长度 100 以内应该可以

机器学习方法万千，具体选择用什么样的方法还是要取决于数据集的规模以及问题本身的复杂度，对于复杂程度一般的问题，看似简单的方法有可能会有出其不意的效果，所以本次采用了贝叶斯分类的方法。

由于随机抽取数据集，所以每次运行的结果都不一样。

具体细节如下：

1. 文本向量化：

根据feature\_words将文本向量化

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

def text\_features(text, feature\_words): # 出现在特征集中，则置1

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 # 返回结果

2、新闻分类器

def TextClassifier(train\_feature\_list, test\_feature\_list, train\_class\_list, test\_class\_list):

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

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

return test\_accuracy

1. 文本特征提取：

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

feature\_words = [] # 特征列表

n = 1

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

if n > 1000: # feature\_words的维度为1000

break

# 如果这个词不是数字，并且不是指定的结束语，并且单词长度大于1小于5，那么这个词就可以作为特征词

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

return feature\_words

**算法调优过程：**

由于数据量过于庞大，我在此大大缩小了数据集的规模，从而精度变化较为巨大，通过调整高频词汇的出现次数调整精度，如下所示：

这部分的主要代码：

test\_accuracy\_list = [] #精度集合 为画图做准备

deleteNs = range(0, 1000, 20) # 0 20 40 60 ... 980

for deleteN in deleteNs:

feature\_words = words\_dict(all\_words\_list, deleteN, stopwords\_set)

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

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

test\_accuracy\_list.append(test\_accuracy)

plt.figure()

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

plt.title('去掉N个高频词与最终检测的准确率的关系')

plt.xlabel('去除高频词的个数')

plt.ylabel('测试准确度')

plt.show()

生成词云：

def Ciyun(g):

coloring = np.array(Image.open("love.jpg"))

wc = WordCloud( font\_path='C:/Windows/Font/simfang.ttf',#设置字体

background\_color="white", #背景颜色

max\_words=2000,# 词云显示的最大词数

mask=coloring,#设置背景图片

max\_font\_size=40, #字体最大值

random\_state=42,

).generate(g)

image\_colors = ImageColorGenerator(coloring)

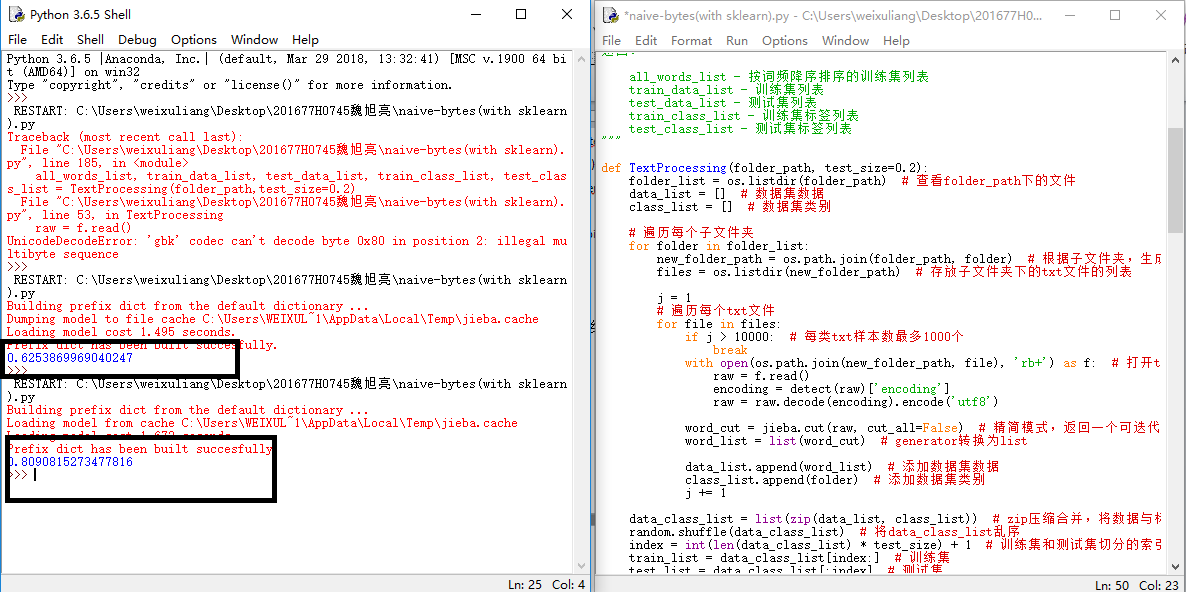
plt.figure(figsize=(64,32))

plt.imshow(wc, interpolation="bilinear")

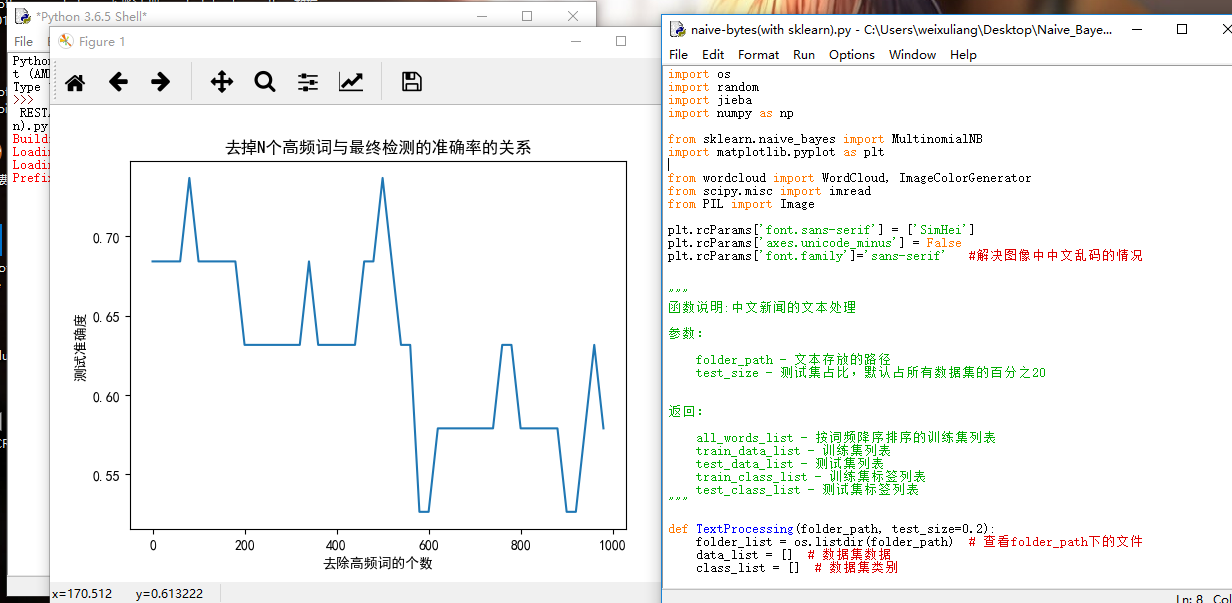
plt.axis('off')

plt.show()

样本数据为1000和10000时对结果的影响，耗费时间也巨长



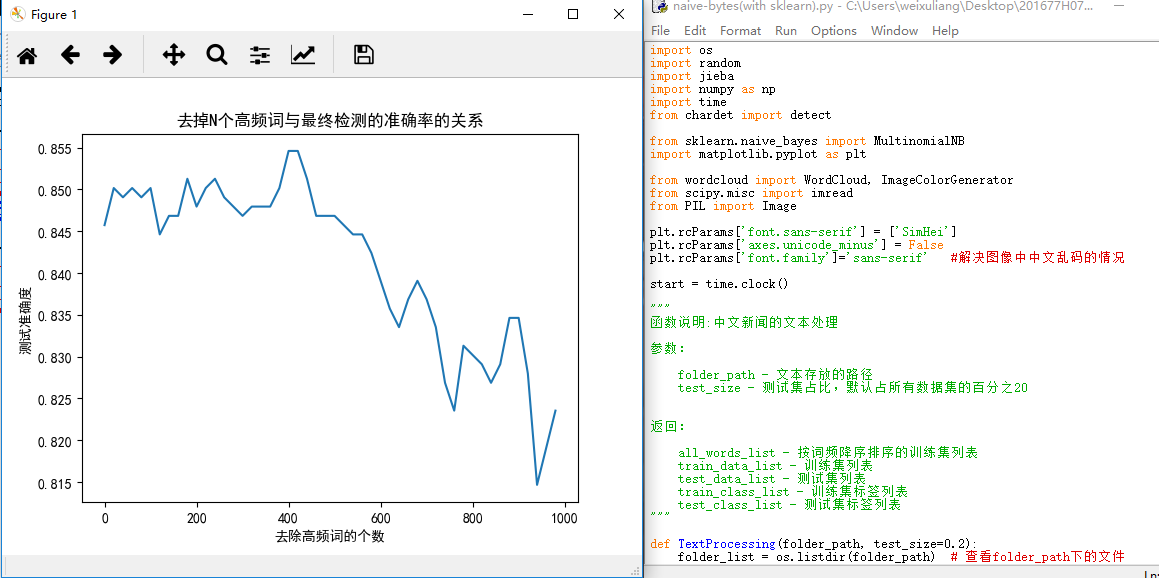
1. **结果分析**



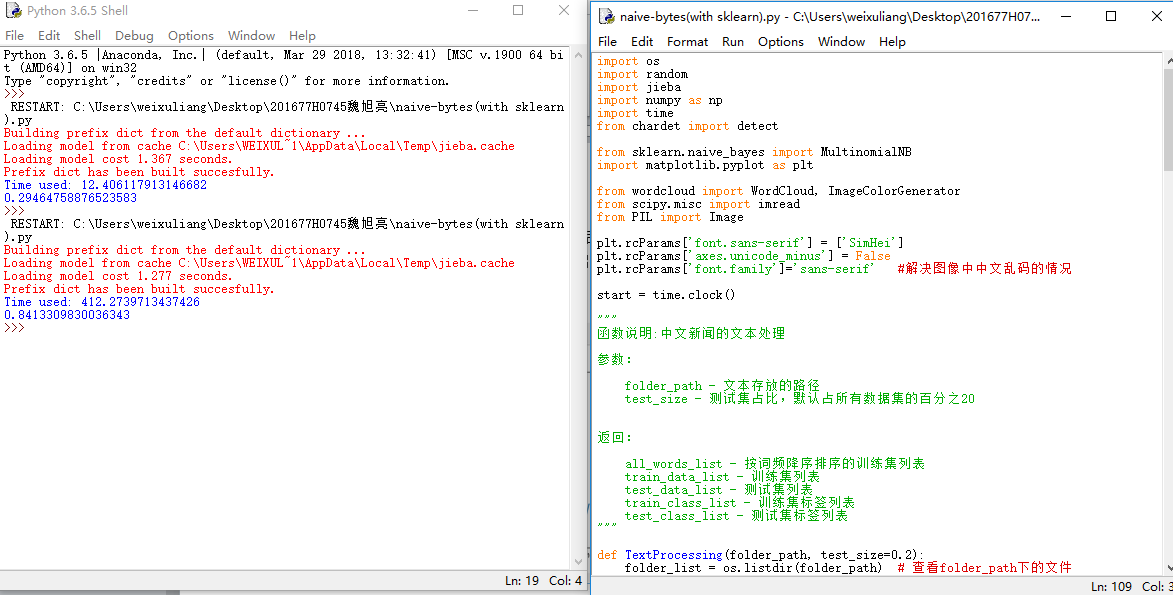
可以看出来随着高频词汇的减少，精确度不断地发生改变，但由于所用数据量太少，故不能明显得到规律

更改大一点的数据集之后我们得到的结果为（之前的数据规模在百位级别，这次在千位），很明显从一开始精度就特别的高，而且趋势比较明显。

十万位的数据量过于庞大，故不做测试



两次实验所消耗时间和精度的对比（第一个数据输出为测试代码，数据集只是数十位，所以精确率极低）



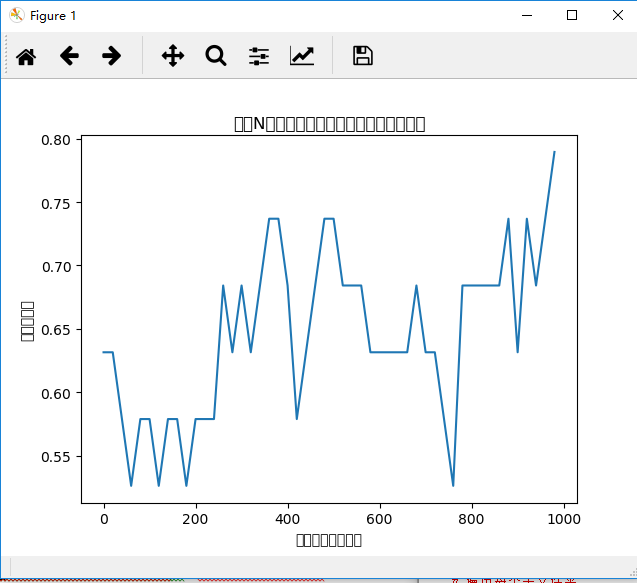
所生成的词云也大相径庭





基于词频生成的词云

**初次运行可能会有下面的错误：**



其中一次的结果

这是因为中文字体的乱码导致的，故我们注明所使用的编码方式，

使用

plt.rcParams['font.sans-serif'] = ['SimHei']

plt.rcParams['axes.unicode\_minus'] = False

plt.rcParams['font.family']='sans-serif' #解决图像中中文乱码的情况

可以解决问题。

1. **结论**

在训练朴素贝叶斯分类器之前，要处理好训练集，文本的清洗还是有很多需要学习的东西。

数据预处理中对于Jieba库的使用和掌握。

根据提取的分类特征将文本向量化，然后训练朴素贝叶斯分类器。

去高频词汇数量的不同，对结果也是有很大的影响的。

**贝叶斯模型**在文本分类中应用十分广泛。

懂得了sickit-learn 中几个常用的分类器以及其在本项目中的效率。

文本一般比较稀疏，最好是将文本数据集转化成稀疏矩阵再做训练。

样品数据越多，最终精确率越高，但耗费时间会越来越长。

文本文档的文件编码问题在python里面是个很大的问题。