实验报告

课程名称	内容安全实验			成　　绩		教师签名	
实验名称	文本分类			实验序号	2	实验日期	2021.04.18
姓　　名		学　　号		专　　业		年级-班	
一、实验目的及实验内容 （本次实验所涉及并要求掌握的知识；实验内容；必要的原理分析）							小题分

实验目的：

用 Python 进行词向量化，学习并使用文本分类算法。

实验内容：

1、完成基于 word2vec 模型的文本分类任务；

2、完成基于 Naive Bayesian 的文本分类任务。

要求使用 python 语言编写(或者自选语料库和任务，但要求必须使用 word2vec 和其中一种分类算法完成两次分类任务)

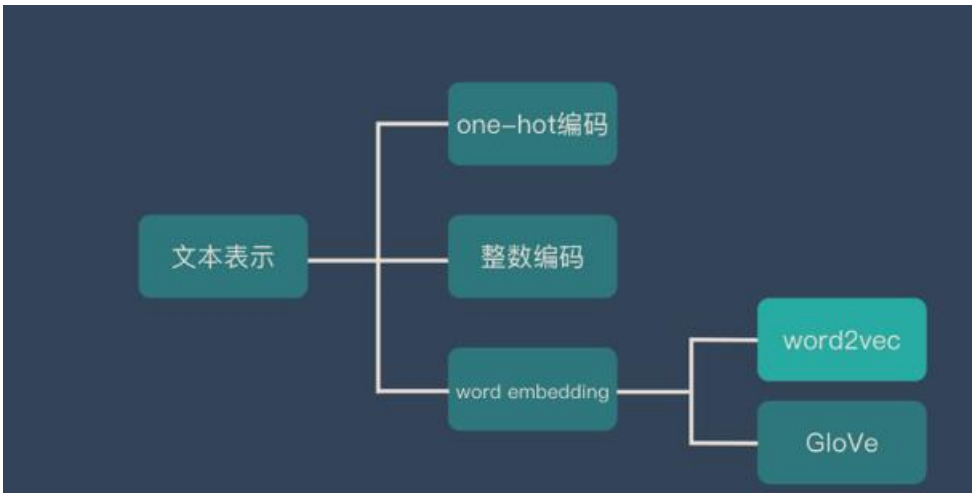
语料库使用群里面提供的素材或者自选。

实验报告中应写出所使用的算法基本原理。

原理分析：

基于 word2vec 模型的文本分类任务

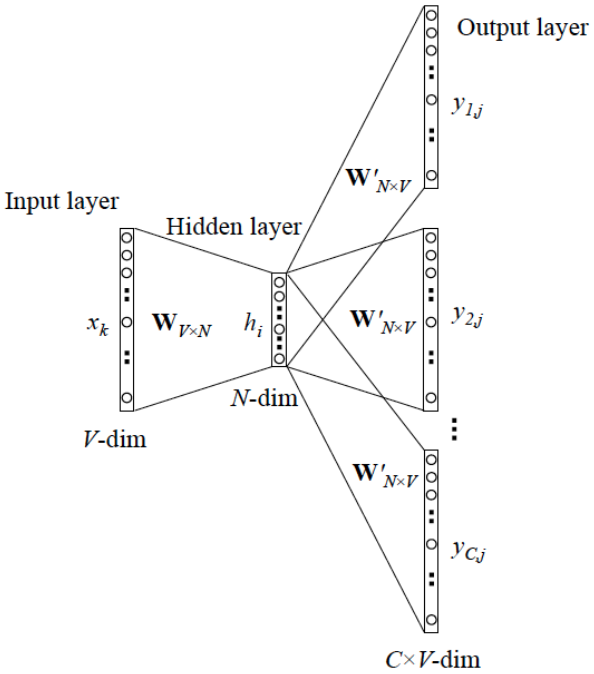
进行文本分类之前，首先要让计算机理解文本，因此需要用一些数学符号或者编码方式来表示文本，该过程被称为文本表示。文本表示的方法有很多种，常用几种方式的如下图所示：



其中 Word2vec 是文本中的词进行向量化表示，以方便计算机对其进行数学运算。词向量化在自然语言处理当中是很重要的，很多任务的第一步就是需要进行词向量化。Word2vec 有两种训练模式，分别是 CBOW(Continuous Bag-of-Words Model)和 Skip-gram (Continuous Skip-gram Model)。CBOW 通过上下文来预测当前值，相当于一句话中扣掉一个词，让你猜这个词是什么；Skip-gram 用当前词来预测上下文，相当于给你一个词，让你猜前面和后面可能

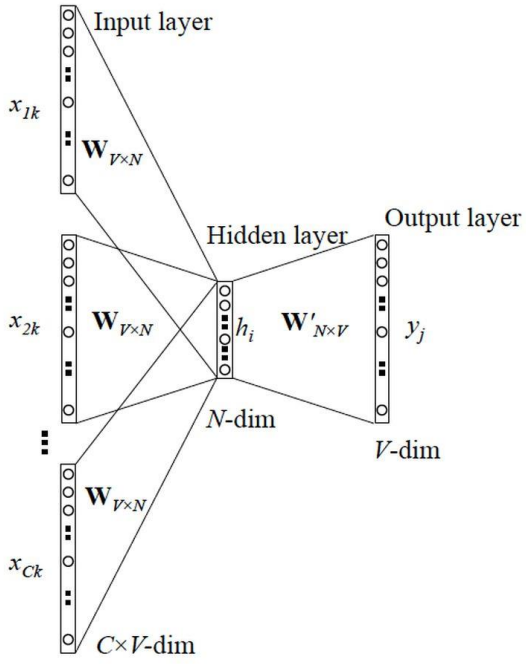
出现什么词。

Skip-gram 的结构如下图所示：



可以看成是 单个 x->单个 y 模型的并联，cost function 是单个 cost function 的累加（取 log 之后）。

而 CBOW 的结构如下图所示：



跟 Skip-gram 的模型并联不同，这里是输入变成了多个单词，所以要对输入处理下（一般是求和然后平均），输出的 cost function 不变。但是不管是哪一种模型，初始化的时候直接为每个词随机生成一个 N 维的向量，并且把这个 N 维向量作为模型参数学习。

使用分词工具（如 jieba）将文本进行清洗及分词处理后，就可以用 gensim 库里的 word2vec 得到词语的向量化表示。

数学定理中，余弦相似度是用向量空间中两个向量夹角的余弦值作为衡量两个个体间差异的大小的度量。余弦值越接近 1 表明两个向量的夹角越接近 0 度，也就是两个向量越相似，夹角等于 0 即两个向量相等；余弦值越接近 0 表明两个向量的夹角越接近 90 度，也就是两个向量的相关性越小，夹角等于 90 即两个向量正交。

经过 word2vec 后，单词实际上已经变成了一个向量，因此若将该思想推广应用到文本领域，余弦相似度即可衡量两个单词的相近程度。更进一步地，如果将一篇文档的所有词向量累加求平均，即可得到一篇文档的特征向量，进而可计算不同文档间的相似度。在此基础上即可进行文本分类，即：

1、将文档分词，得到语料库；

2、训练得到词向量表示；

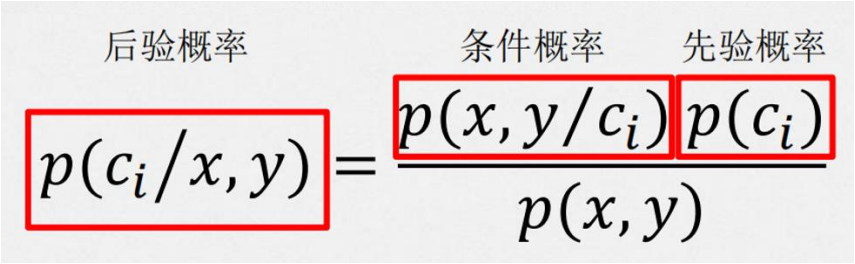
3、对训练集中的每一个类别下的所有文档，将其整合成一篇大的文档，提取训整合后文档的特征向量，即所有词向量求和求平均，将得到的结果作为该类的特征向量；

4、对每一篇测试集中的文档，将该文档中的所有词向量求和求平均，作为该文档的特征向量；

5、将该文档的特征向量与所有类的特征向量分别做余弦相似度计算，取相似度最高的结果对应的类作为该文档的预测分类。

基于 Naive Bayesian 的文本分类任务

朴素贝叶斯的基本思想为，利用先验概率和条件概率估算后验概率，其计算公式如下图所示：



对上图的解释为：先验概率是指，事情还没有发生，那么这件事情发生的可能性的大小，即概率空间中的各个类别的总体分布情况；后验概率是指，事情已经发生，那么这件事情发生的原因是由某个因素引起的可能性的大小。

因此用朴素贝叶斯进行文本分类的步骤为：

1、计算先验概率$P(C_i)$；

2、计算独立条件概率$P(x/C_i)$；

3、计算总条件概率$P(x)=\ \sum \mathrm{i}\ P(C_i)*P(x/C_i)$；

4、计算后验概率$P(C_i/x)$；

5、选取最大的后验概率确定最终的类别。

当然，实际在做时没必要重复造轮子，直接用 nltk 包里的贝叶斯分类器即可。即，真实的步骤为：

1、划分训练集和测试集数据；

2、取 nltk 中的贝叶斯分类器；

3、将训练集数据及对应的标签送入分类器进行训练；

4、调用评估函数，评估分类的准确性。

二、实验环境 （本次实验所使用的器件、仪器设备等的情况）	小题分：

(1)处理器：Intel(R) Core(TM) i5-9300H CPU @ 2.40GHz　2.40 GHz
(2)操作系统环境：Windows 10 家庭中文版 x64 19042.867
(3)编程语言：Python 3.8
(4)其他环境：16 GB 运行内存
(5)IDE 及包管理器：JetBrains PyCharm 2020.1 x64， anaconda 3 for Windows（conda 4.9.0）

三、实验步骤及实验过程分析 （详细记录实验过程中发生的故障和问题，进行故障分析，说明故障排除的过程及方法。根据具体实验，记录、整理相应的数据表格、绘制曲线、波形等）	小题分：

说明：

由于随机数种子设置等情况，本篇实验报告所记录的内容仅为写报告时（2021/04/23）的情况，可能与实际实验时（2021/04/18）结果有出入。

一切以实际运行时所得到的结果为准。

基于 word2vec 实现文本分类

1. 安装并导入工具包：本实验主要使用到的工具包有 gensim 包，jieba 包，numpy 包和 re 包等。

```
# -*- coding: utf-8 -*-
from gensim.models import word2vec, Word2Vec
from tqdm import tqdm
import jieba.analyse
import numpy as np
import os
import re
```

2. 获取词表：把所有类别中的文件使用 jieba 进行中文分词(只获取名词类和动词类)，整合后生成 word2vec 的词表。这里有两种方法：

a. 采用 jieba.analyse.textrank 方法。该方法可以使用基于 TextRank 算法的关键词提取，所以在切词的时候就会过滤掉一部分频繁出现但实际没多大影响的词，如各种语气词等。该方法会破坏词与词之间的上下文关系，因为该方法切出的词返回的结果是按照频次排序的。

b. 采用 jieba.posseg 方法，该方法会按照精确模式切词并返回其词性，返回结果的顺序为行文中出现的先后顺序，随后只保留指定词性的词项即可。该方法须配合停用词表使用，否则会保留一些无意义的词，如语气词等。

这里采用方法 a。因为在切词后会将切好的词项整合到一个语料文件中，这会在一定程度上削弱词与词之间的上下文关系，所以效果上 a、b 可能差别不大，但是 a 更方便。

```
def get_corpus(self):
    for item in self.class_list:
        class_path = self.root_path + '/' + item
        file_list = os.listdir(class_path)
        for file_ in file_list:
            filename = class_path + '/' + file_
            with open(filename, 'r', encoding='utf8') as f:
                cont = f.read()
            cont = "".join(cont.split())
            word_list = jieba.analyse.textrank(cont, topK=None, withWeight=False, allowPOS=('n', 'ns', 'v', 'vn'))
            if len(word_list) == 0:
                continue
            else:
                self.all_txt_list.extend(word_list)

    result = " ".join(self.all_txt_list)
    with open('result.txt', 'w', encoding='utf8') as f:
        f.write(result)
```

3. 训练模型：使用已经生成的词表训练我们的 word2vec 模型并保存，通常维度设为 200，最小词频设置为 1。

直接用 gensim.models 里的 word2vec 方法即可。

```
def get_model(self):
    if os.path.exists('my_model.model'):
        self.model = Word2Vec.load('my_model.model')
        return
    sentences = word2vec.Text8Corpus("result.txt")
    model = word2vec.Word2Vec(sentences, size=200, min_count=1)
    model.save('my_model.model')
    self.model = model
```

4. 统计词频：我们选取每个类别 70%的文件作为训练集，统计词频(同样只获取名词和动词)，剩余 30%的文件作为测试集。

```
def train_frequency(self, rate_train=0.7):
    print('\n对训练集词频统计')
    for c in tqdm(self.class_list):
        all_words = {}
        class_path = self.root_path + '/' + c
        file_list = os.listdir(class_path)
        nFile = len(file_list)
        for name in file_list[:int(nFile * rate_train)]:
            file_path = class_path + '/' + name
            with open(file_path, 'r', encoding='utf8') as f:
                cont = f.read()
                cont = re.sub(re.compile(r'原文网址:.*'), '', cont)
                cont = "".join(cont.split())
            word_list = jieba.analyse.textrank(cont, topK=None, withWeight=False, allowPOS=self.allPOS)
            if len(word_list) != 0:
                for word in word_list:
                    if word in all_words.keys():
                        all_words[word] += 1
                    else:
                        all_words[word] = 1
        self.class_all_words[c] = all_words
```

5. 获取特征向量：对于每个类别前 k 个高频作为关键词，通过训练好的 word2vec 模型获取关键词的向量求平均即为对应类别的特征向量。

```
def average_class_vector(self, keyword_num):
    """
    获取每个类的k个关键词的平均向量
    """
    average_class_dic = {}
    for c in self.class_all_words:
        all_words_list = sorted(self.class_all_words[c].items(), key=lambda item: item[1], reverse=True)
        shape_ = self.model.wv[all_words_list[0][0]].shape
        total = np.zeros(shape_[0])
        limit = min(keyword_num, len(all_words_list))
        for t in range(limit):
            total += self.model.wv[all_words_list[t][0]]
        average_class_dic[c] = total / limit
    return average_class_dic
```

6. 准确率计算：分别获取测试集中每个文件的前 k 关键词(如果不够 k 个就全部获取)求得特征向量，然后分别与每个类别的特征向量求相似度，从而预测该文件为相似度最高的那个类别，并于真是标签对比，求得最终的准确率。

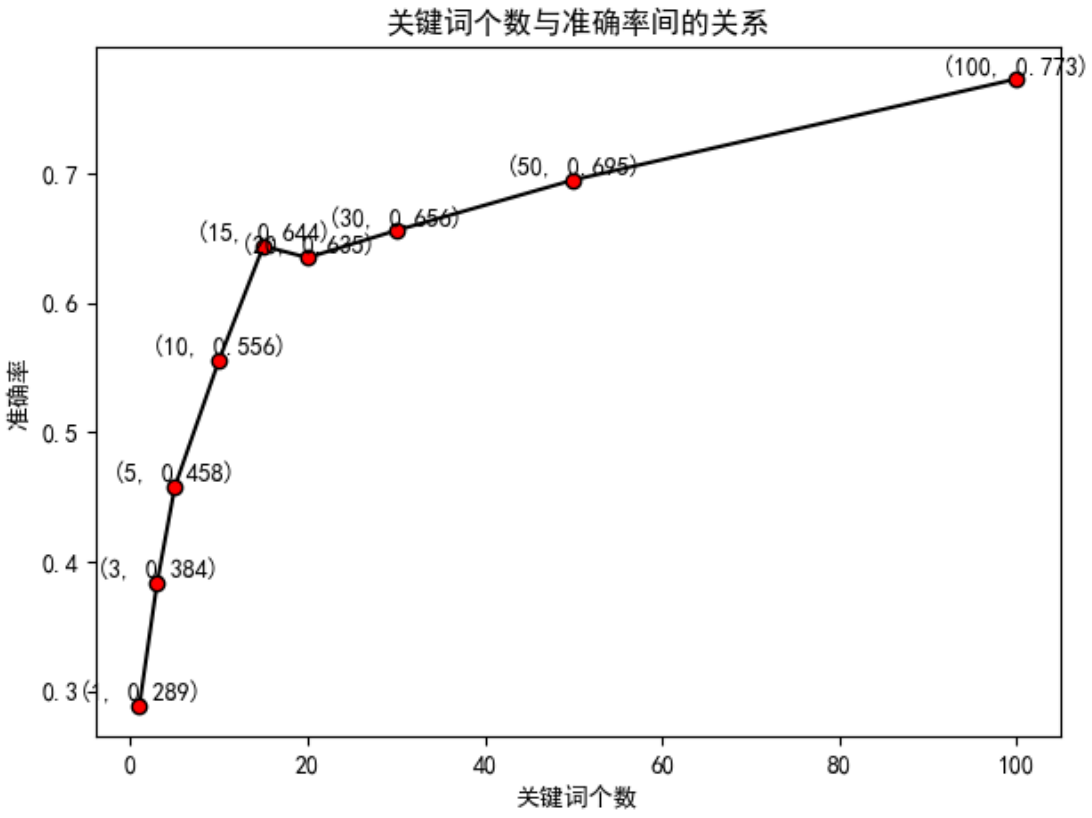
```
    # 取30%作测试集
    for name in file_list[int(nFile * rate):]:
        file_path = class_path + '/' + name
        test_data_words = {}
        with open(file_path, 'r', encoding='utf8') as f:
            txt = f.read()
            txt = re.sub(re.compile(r'原文网址:.*'), '', txt)
        txt = "".join(txt.split())
        word_list = jieba.analyse.textrank(txt, topK=None, withWeight=False, allowPOS=('n', 'ns', 'v', 'vn'))
        for word in word_list:
            if word in test_data_words.keys():
                test_data_words[word] += 1
            else:
                test_data_words[word] = 1
        test_words_list = sorted(test_data_words.items(), key=lambda item: item[1], reverse=True)
        if len(test_words_list) == 0:
            continue
        shape_ = self.model.wv[test_words_list[0][0]].shape
        total = np.zeros(shape_[0])
        limit = min(keyword_num, len(test_words_list))
        for t in range(limit):
            total += self.model.wv[test_words_list[t][0]]
        average_test_vector = total / limit
        pre = predict(average_class_dic, data=average_test_vector)
        if pre == c:
            true += 1
        else:
            false += 1
return true / (true + false)
```

运行情况：

```
C:\ProgramData\Anaconda3\python.exe C:/Users/CandyMonster/Desktop/EXPCLASS/内容安全/exp2/w2v.py
Building prefix dict from the default dictionary ...
Dumping model to file cache C:\Users\CANDYM~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.739 seconds.
Prefix dict has been built succesfully.
  0%|          | 0/7 [00:00<?, ?it/s]对训练集词频统计
100%|██████████| 7/7 [00:29<00:00,  4.27s/it]
计算准确率
  0%|          | 0/7 [00:00<?, ?it/s]Keyword_num: 1
100%|██████████| 7/7 [00:13<00:00,  1.93s/it]
Keyword_num: 3
100%|██████████| 7/7 [00:13<00:00,  1.91s/it]
Keyword_num: 5
100%|██████████| 7/7 [00:13<00:00,  1.88s/it]
Keyword_num: 10
100%|██████████| 7/7 [00:13<00:00,  1.88s/it]
Keyword_num: 15
100%|██████████| 7/7 [00:13<00:00,  1.92s/it]
  0%|          | 0/7 [00:00<?, ?it/s]Keyword_num: 20
100%|██████████| 7/7 [00:13<00:00,  1.91s/it]
Keyword_num: 30
100%|██████████| 7/7 [00:13<00:00,  1.88s/it]
  0%|          | 0/7 [00:00<?, ?it/s]Keyword_num: 50
100%|██████████| 7/7 [00:13<00:00,  1.97s/it]
  0%|          | 0/7 [00:00<?, ?it/s]Keyword_num: 100
100%|██████████| 7/7 [00:14<00:00,  2.02s/it]

Process finished with exit code 0
```

实验结果：



随着关键词个数增多，可以获得的特征更多，预测准确率更高，最高达到了 77.3%，但是当关键词获得太多时，有可能会形成冗余的特征，从而导致准确率下降。

基于 Naive Bayesian 的文本分类

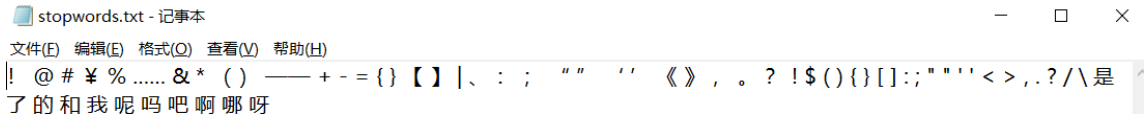
1. 安装并导入工具包：本实验主要使用到的工具包有 jieba 包和 nltk 包。

```
import os
import jieba
import nltk
```

2. 统计词频：我们对整个语料库统计词频形成词表，并选取每个类别 70%的文件作为训练集，30%的文件作为测试集。

```
for label in self.label_list:
    c_path = self.root_path + '/' + label
    if os.path.isdir(c_path):
        file_list = os.listdir(c_path)
        self.train_set[label] = []
        self.test_set[label] = []
        nFile = len(file_list)
        for filename in file_list:
            file = c_path + '/' + filename
            with open(file, 'r', encoding='utf8') as f:
                cont = f.read()
                cont = "".join(cont.split())
            word_cut = jieba.lcut(cont, cut_all=False)
            word_cut_ = []
            for word in word_cut:
                if word in self.stopwords:
                    # 停用词，一些标点符号和语气词等
                    continue
                else:
                    word_cut_.append(word)
                    if word in self.all_words.keys():
                        self.all_words[word] += 1
                    else:
                        self.all_words[word] = 1
            # 70%作为训练集，剩下30%作为测试集
            if file_list.index(filename) < int(rate * nFile):
                self.train_set[label].append(word_cut_)
            else:
                self.test_set[label].append(word_cut_)
```

3. 去除停用词：通常前 N 个高频词是一些没有实际意义的词或不能够反应文本特征的重要词，由于这些词过于常用，从而不能很好的体现文本特征，例如你，的，呢，了，我等。



4. 获取文本特征：使用去除停用词的词表在每一个文本中检索，如果文本存在词表中的词，那么就认为这个词属于这个文本的一个特征。

```
def words_dict(self, delete_n):
    n = 0
    word_features_list = []
    all_words_list = sorted(self.all_words.items(), key=lambda item: item[1], reverse=True)
    for t in range(delete_n, len(self.all_words), 1):
        if n > 1000:
            break
        else:
            n += 1
            word_features_list.append(all_words_list[t][0])
    return word_features_list

@staticmethod
def doc_features(doc, word_features):
    doc_words = set(doc)
    features = {}
    for word in word_features:
        features['contains{}'.format(word)] = (word in doc_words)
    return features
```

5. 贝叶斯分类器训练：把文本特征和真实类别标签输入到分类器中训练。

```
def Classfication_Bayes(self, word_features):
    train_data = []
    for label in self.train_set.keys():
        for vec in self.train_set[label]:
            feature = self.doc_features(vec, word_features)
            tmp = (feature, label)
            train_data.append(tmp)

    test_data = []
    for label in self.test_set.keys():
        for vec in self.test_set[label]:
            feature = self.doc_features(vec, word_features)
            tmp = (feature, label)
            test_data.append(tmp)

    classifier = nltk.NaiveBayesClassifier.train(train_data)
```

6. 准确率计算：把测试集的文本特征和真实标签输入到训练好的分类器中，求得准确率。

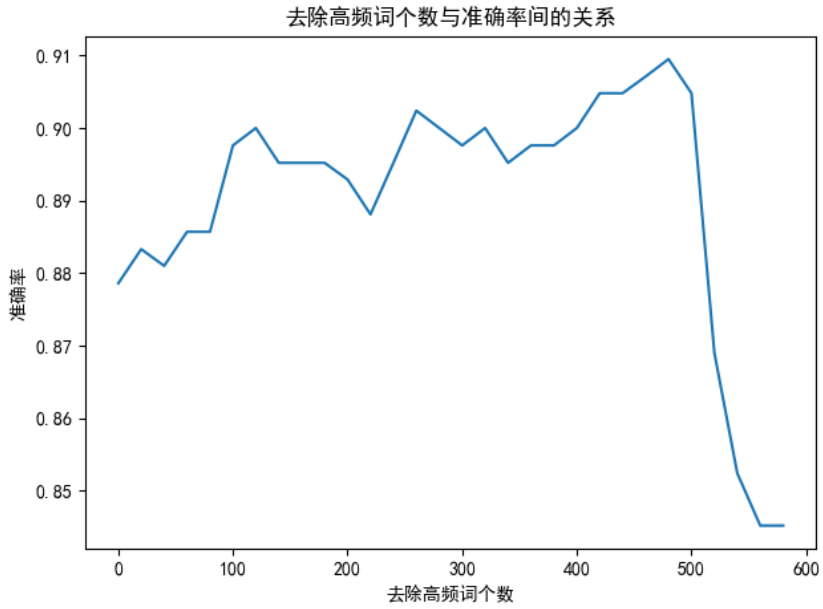
```
test_acc = nltk.classify.accuracy(classifier, test_data)
return round(test_acc, 4)
```

运行情况：

```
C:\ProgramData\Anaconda3\python.exe C:/Users/CandyMonster/Desktop/EXPCLASS/内容安全/exp2/ml/ml_main.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\CANDYM~1\AppData\Local\Temp\jieba.cache
Loading model cost 0.520 seconds.
Prefix dict has been built succesfully.
100%|██████████| 30/30 [02:34<00:00,  5.16s/it]
best accuracy:  0.9095
The corresponding number of removed high-frequency words:  480

Process finished with exit code 0
```

实验结果：



由于停用词表较小，所以仍然保留有很多的无用词。随着去除高频词个数增多，可以去掉

的无用词更多，预测准确率更高，最高达到了 90.95%，但是当去除的高频词太多时，有可能会去掉特征词，从而导致准确率下降。

四、实验结果总结
（对实验结果进行分析，完成思考题目，总结实验的新的体会，并提出实验的改进意见）

小题分：

本次文本分类实验相对来讲比较简单，使用前人已经造好的轮子就可以很方便地完成实验。然而重要的并不是实验结果，而是实验原理，因此通过实验明白实验原理对日后的发展有很大用处。

有很多参数都会对实验结果产生较大的影响，如高频词的选取、停用词的选取等。

自然语言处理的工具有很多，各种封装好的分类算法也都很完善，nltk、sklearn 等都很方便使用。

正则表达式在对提取到的数据进行过滤清洗的时候会起到至关重要的作用，编写合适的正则表达式可以更有效地得到便于训练的数据。

文本分类相关的工作有很多且都很有成效。文本分类对内容安全具有很重要的意义。