实验报告

一、实验目的

1.了解朴素贝叶斯分类的原理，掌握相关概率和统计知识

2.应用朴素贝叶斯概率模型，做文本分类

二、实验任务

对数据进行分词和统计运算 （可选：词项优化方法）

将分词的结果输出到文件 c_学号.txt 中，格式：文件 id:类别

对比 answer.txt 和 c_学号. txt，得到 F 指标

三、使用环境

pycharm

python 3.7

四、实验过程

1.数据集格式

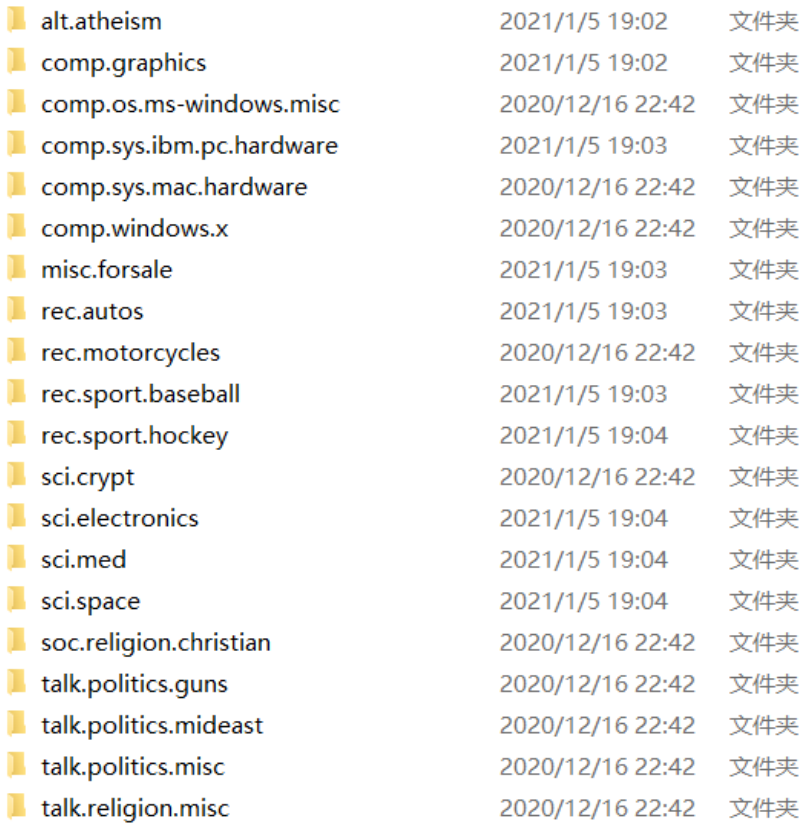
训练集有 20 个类，每个类约 800 个文件；

train 文件夹下的每一个子文件夹对应一个类，子文件夹名即为类名。

测试集共 4000 个文件，模型输出预测结果每个文件的类别。

因此这是一个 20 分类问题。

20 个类别如下图所示：



2.朴素贝叶斯分类器

文档 d 属于类别 c 的概率计算如下：

$$P(c|d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k|c)$$

- n_d 是文档的长度(词条的个数)
- $P(t_k|c)$ 是词项 t_k 出现在类别 c 中文档的概率
- $P(t_k|c)$ 度量的是当 c 是正确类别时 t_k 的贡献
- $P(c)$ 是类别 c 的先验概率

朴素贝叶斯分类的目标是寻找“最佳”的类别

最佳类别是具有最大后验概率(maximum a posteriori -MAP)的类别 c_{map}:

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} \hat{P}(c|d) = \arg\max_{c \in \mathbb{C}} \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k|c)$$

很多小概率的乘积会导致浮点数下溢出，由于 $\log(xy) = \log(x) + \log(y)$，可以通过取对数将原来的乘积计算变成求和计算；log 是单调函数，因此得分最高的类别不会发生改变。实际中常常使用：

$$c_{\text{map}} = \arg\max_{c \in \mathbb{C}} [\log \hat{P}(c) + \sum_{1 \le k \le n_d} \log \hat{P}(t_k|c)]$$

以上是分类的规则，而先验概率和条件概率都必须作为模型参数，通过训练集训练得来。使用极大似然估计时，

$$\hat{P}(c) = \frac{N_c}{N} \qquad \hat{P}(t|c) = \frac{T_{ct}}{\sum_{t' \in V} T_{ct'}}$$

为了避免零概率，提高模型的准确度，需要对其做平滑，即

$$\hat{P}(t|c) = \frac{T_{ct}+1}{\sum_{t' \in V}(T_{ct'}+1)} = \frac{T_{ct}+1}{(\sum_{t' \in V} T_{ct'}) + B}$$

3.代码实现

在 train.py 中实现了主函数，运行 train.py 可以使整个系统运行。主函数按顺序依次建立训练集和测试集的词典、统计词频、训练模型、做预测并输出到结果文件。执行这些过程需要调用 model.py 中的类和函数，因此在该文件开头 import models。

```python
import models

if __name__ == '__main__':
    train_data = models.Prepare("train")
    test_data = models.Prepare("test")

    train_data.create_dic()
    test_data.create_dic()

    train_data.count_tokens()
    test_data.count_tokens()

    train_data.create_news_group()
    test_data.create_news_group()

    print("开始训练...")
    model = models.NaiveBayes(train_data.Count_Path)
    model.nb_multinomial_train(train_data.news_group)
    print("开始预测...")
    model.nb_multinomial_test(test_data.news_group)
    print("运行成功！")

```

models 文件主要实现了 Prepare 和 NaiveBayes 两个类，Prepare 负责字典构建、词频统计，为估计模型参数做准备，NaiveBayes 负责训练模型和预测。

models 文件首先 import 四个包，os 用于数据文件的读取和写入，re 用于正则表达式，nltk 是因为需要下载一个 nltk 的通用英文停用词表，math 是为了使用 log 函数。

```
import os
import re
import nltk
import math
```

在 Prepare 类中，类内变量包括 Main_Path, Dic_Path, Count_Path 和 news_group。Main_Path 是待处理的数据集文件所在的路径，Dic_Path 是生成的词典所在的路径，Count_Path 是词频统计结果所在的路径，news_group 存储了数据集的结构信息，包括文件夹名和对应的文件名。

```
class Prepare:

    def __init__(self, Main_Path):
        self.Main_Path = Main_Path
        self.Dic_Path = Main_Path + "_dic"
        self.Count_Path = Main_Path + "_count"
        self.news_group = {}
```

Prepare 类中还包括三个函数：

create_dic：为每个文件创建词典并写入文件系统

count_tokens：统计词频并写入文件系统

create_news_group：得到文档集结构信息

在 create_dic 中，首先判断词典文件集是否已经存在，避免调试过程中重复创建影响效率。接着下载 nltk 的英文停用词表。遍历传入路径下的每一个文件夹和文件，创建相同的文件目录结构，读取每一个文件的全部内容，对单词做转换小写的操作，去除停用词，最后将得到的单词列表，按每个单词占据一行的格式，写入词典文件集。

```
# 为每个文件创建词典
def create_dic(self):
    # 判断是否已创建完毕
    if not os.path.exists(self.Dic_Path):
        print("start create " + self.Dic_Path)
        os.makedirs(self.Dic_Path)
    else:
        print("{} already exists".format(self.Dic_Path))
        print()
        return None
    nltk.download('stopwords')
    for group_name in os.listdir(self.Main_Path):  # 遍历每一个文件夹
        group_path = self.Main_Path + '/' + group_name
        group_path_dic = self.Dic_Path + '/' + group_name
        if not os.path.exists(group_path_dic):
            os.makedirs(group_path_dic)
        for file_name in os.listdir(group_path):  # 遍历每一类下的每一个文件
            file_path = group_path + '/' + file_name  # 原文件路径
            file_path_dic = group_path_dic + '/' + group_name + "-" + file_name  # 字典文件路径
            fw = open(file_path_dic, 'w')
            content = open(file_path, 'r', encoding='utf-8', errors='ignore').readlines()  # 读取原文件的全部内容
            for line in content:
                word_list = get_words(line)  # 该函数返回去除停用词后一个句子的单词列表
                for word in word_list:
                    fw.write('{}\n'.format(word))
            fw.close()
        print(group_name + "词典文件创建成功")
    print(self.Dic_Path + "下的文件全部创建成功")
    print()
    return None
```

在 count_tokens 中，基于刚刚创建的词典，创建一个相同的目录结构，对每一个文件中的词，做词频统计，并写入文件。文件内容是“词项 词频”为一行，以空格间隔。

```
# 统计词频
def count_tokens(self):
    if not os.path.exists(self.Count_Path):
        print("start create {}".format(self.Count_Path))
        os.makedirs(self.Count_Path)
    else:
        print("{} already exists".format(self.Count_Path))
        print()
        return None
    for group_name in os.listdir(self.Dic_Path):# 计算每个文件的词频并写入文件系统
        group_path_dic = self.Dic_Path + '/' + group_name
        group_path_count = self.Count_Path + '/' + group_name
        if not os.path.exists(group_path_count):
            os.makedirs(group_path_count)
        for file in os.listdir(group_path_dic):
            word_dic = {}
            file_path_dic = group_path_dic + '/' + file
            file_path_count = group_path_count + '/' + file
            for line in open(file_path_dic).readlines():
                word = line.strip('\n')
                if word in word_dic:
                    word_dic[word] += 1
                else:
                    word_dic[word] = 1
            f = open(file_path_count, 'w')
            for word in word_dic:
                f.write('{} {}\n'.format(word, word_dic[word]))
            f.close()
        print(group_name + "统计词频成功")
    print(self.Count_Path + "下的文件全部创建成功")
    return None
```

在 create_news_group 中，将 news_group 字典的内容修改为{文件夹名：对应的文件名列表}：

```
# 得到文档集结构信息
def create_news_group(self):
    for group_name in os.listdir(self.Count_Path):
        self.news_group[group_name] = []
        group_path = self.Count_Path + '/' + group_name
        for file_name in os.listdir(group_path):
            self.news_group[group_name].append(file_name)
    return None
```

在 NaiveBayes 类中，类内变量包括 train_set, test_set, NB_word_P, NB_group_P, Count_Path 和 total_word。train_set 是训练集的文档结构信息，test_set 是测试集的文档结构信息，二者都与 Prepare 类中的 news_group 一致；NB_word_P 保存着模型需要的条件概率（即每个词出现在某一类中的概率），NB_group_P 保存每个类的先验概率，Count_Path 是由训练集做 Prepare 形成的词频统计文档集所在的路径，total_word 保存了整个文档集的词频统计信息。

```
class NaiveBayes:

    def __init__(self, Count_Path):
        self.train_set = {}
        self.test_set = {}
        self.NB_word_P = {}
        self.NB_group_p = {}
        self.Count_Path = Count_Path
        self.total_word = {}
```

NaiveBayes 类中还包括三个函数：

nb_multinomial_train：模型训练

nb_multinomial_test：测试

在 nb_multinomial_train 中，根据 train_set 的文档集结构信息，对之前处理好的训练集的词频统计文件操作，将每个文件的词频统计结果读入 word_count 字典。接下来对 word_count 字典的每一个键值对，将词频记入 NB_word_P[group][word]，同时完成 total_word 的全部文档集的词频统计：

```
def nb_multinomial_train(self, train_set):
    self.train_set = train_set
    for group in self.train_set:
        group_path_count = self.Count_Path + '/' + group
        self.NB_word_P[group] = {}
        for file in self.train_set[group]:
            file_path_count = group_path_count + '/' + file
            word_count = {}
            with open(file_path_count, 'r', encoding='utf-8', errors='ignore') as f:
                while 1:
                    line = f.readline().split()
                    if not line:
                        break
                    if line[0] not in word_count:
                        word_count[line[0]] = int(line[1])
                    else:
                        word_count[line[0]] += int(line[1])
            for word in word_count:
                if word not in self.NB_word_P[group]:
                    self.NB_word_P[group][word] = word_count[word]
                else:
                    self.NB_word_P[group][word] += word_count[word]
                if word not in self.total_word:
                    self.total_word[word] = word_count[word]
                else:
                    self.total_word[word] += word_count[word]
```

group_total 记录了每个类的词频，total 是 total_word 的值的求和，即所有单词的词频。在得到它们之后，group_total[group]) / total 即为先验概率，由于后续处理需要对它取对数，在这里直接先取对数；NB_word_P 在上一步记录的是词频，在这一步做后续处理，计算后验概率并做平滑，即

float(self.NB_word_P[i][word] + 0.01) / (group_total[i] + 0.01 * len(self.NB_word_P[i]))

再取对数，训练模型的准备到此结束。

注：先验概率也可以使用更简单的 某类文档个数/总文档个数 ，但改进后训练效果更好

```
group_total = {}
total = sum(self.total_word.values())
for group in self.NB_word_P:
    group_total[group] = 0
    for word in self.total_word:
        if word not in self.NB_word_P[group]:
            self.NB_word_P[group][word] = 0
        group_total[group] += self.NB_word_P[group][word]
    print("Group {} counts of words are {}".format(group, group_total[group]))
    self.NB_group_p[group] = math.log(float(group_total[group]) / total)
print("counts of words in total training set is {}".format(total))
print()
for i in self.NB_word_P:
    for word in self.NB_word_P[i]:
        self.NB_word_P[i][word] = math.log(
            float(self.NB_word_P[i][word] + 0.01) / (group_total[i] + 0.01 * len(self.NB_word_P[i])))
return None
```

在 nb_multinomial_test 中，test_set 是测试集的文档结构信息，接着把之前准备好的测试集词频统计结果读入 test_news：

```
def nb_multinomial_test(self, test_set):
    self.test_set = test_set
    test_news = {}
    for group in self.test_set:
        group_index_path = 'test_count/' + group
        for news in self.test_set[group]:
            test_news[news] = {}
            news_index_path = group_index_path + '/' + news
            with open(news_index_path, 'r', encoding='utf-8', errors='ignore') as f:
                while 1:
                    line = f.readline().split()
                    if not line:
                        break
                    test_news[news][line[0]] = int(line[1])
```

接下来做预测，将概率 largest_p 首先设置为一个极小值，计算文档出现在某一类的概率，若这个概率大于 largest_p，则替换 largest_p，并且记录类名。使用的计算公式在之前推导过，注意使用过 log，因此乘法变加法，且 xlogy = logy^x。最后将预测结果按格式写入 c_10182100208.txt 文件。

```
    fw = open("D:/大四上/信息检索/pro04/TextClassification/c_10182100208.txt", 'w')
    for news in test_news:
        largest_p = float("-inf")
        prediction = ""
        for group in self.NB_word_P:
            p = self.NB_group_p[group]
            for word in self.NB_word_P[group]:
                if word in test_news[news]:
                    p = p + (self.NB_word_P[group][word] * test_news[news][word])
            if p > largest_p:
                prediction = group
                largest_p = p
        names = news.split("-")
        name = names[len(names) - 1]
        fw.write(name + ":" + prediction + "\n")
    fw.close()
    return None
```

4.程序运行结果

D:\python\python.exe D:/大四上/信息检索/pro04/TextClassification/train.py

D:\python\lib\site-packages\sklearn\feature_extraction\text.py:17: DeprecationWarning: Using or importing the ABCs from 'collections' instead of from 'collections.abc' is deprecated, and in 3.8 it will stop working

from collections import Mapping, defaultdict

start create train_dic

[nltk_data] Downloading package stopwords to

[nltk_data] C:\Users\nicole\AppData\Roaming\nltk_data...

[nltk_data] Package stopwords is already up-to-date!

alt.atheism 词典文件创建成功

```
comp.graphics 词典文件创建成功
comp.os.ms-windows.misc 词典文件创建成功
comp.sys.ibm.pc.hardware 词典文件创建成功
comp.sys.mac.hardware 词典文件创建成功
comp.windows.x 词典文件创建成功
misc.forsale 词典文件创建成功
rec.autos 词典文件创建成功
rec.motorcycles 词典文件创建成功
rec.sport.baseball 词典文件创建成功
rec.sport.hockey 词典文件创建成功
sci.crypt 词典文件创建成功
sci.electronics 词典文件创建成功
sci.med 词典文件创建成功
sci.space 词典文件创建成功
soc.religion.christian 词典文件创建成功
talk.politics.guns 词典文件创建成功
talk.politics.mideast 词典文件创建成功
talk.politics.misc 词典文件创建成功
talk.religion.misc 词典文件创建成功
train_dic 下的文件全部创建成功

start create test_dic
[nltk_data] Downloading package stopwords to
[nltk_data]         C:\Users\nicole\AppData\Roaming\nltk_data...
[nltk_data]       Package stopwords is already up-to-date!
test-all-in-one 词典文件创建成功
test_dic 下的文件全部创建成功

start create train_count
alt.atheism 统计词频成功
comp.graphics 统计词频成功
comp.os.ms-windows.misc 统计词频成功
comp.sys.ibm.pc.hardware 统计词频成功
comp.sys.mac.hardware 统计词频成功
comp.windows.x 统计词频成功
misc.forsale 统计词频成功
rec.autos 统计词频成功
rec.motorcycles 统计词频成功
```

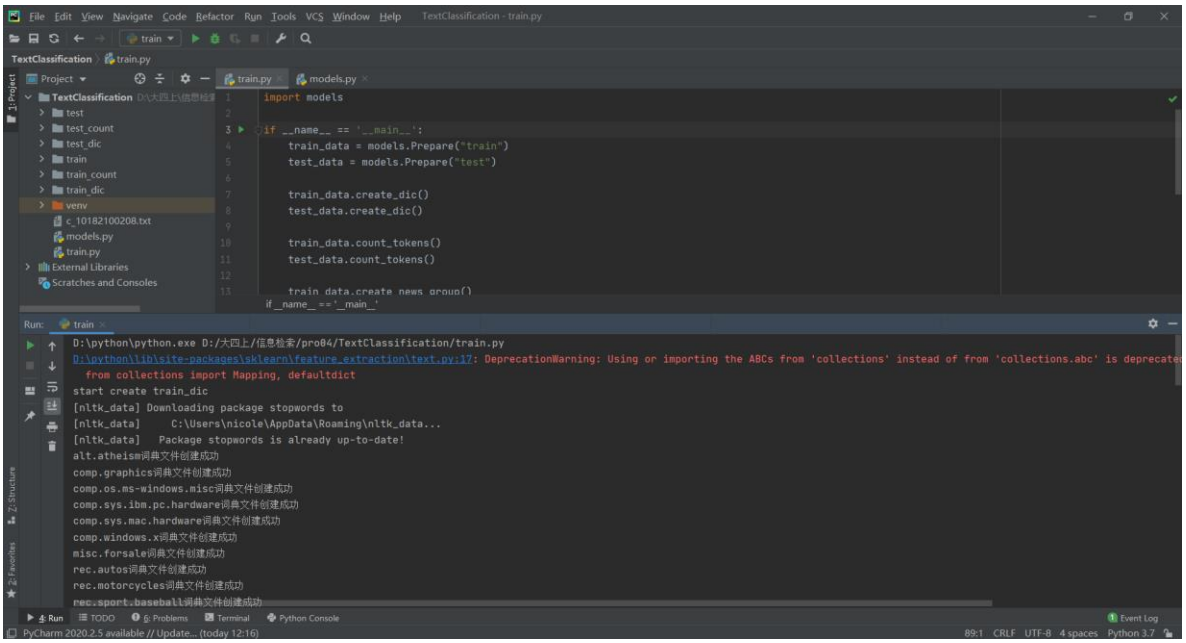
```
rec.sport.baseball 统计词频成功
rec.sport.hockey 统计词频成功
sci.crypt 统计词频成功
sci.electronics 统计词频成功
sci.med 统计词频成功
sci.space 统计词频成功
soc.religion.christian 统计词频成功
talk.politics.guns 统计词频成功
talk.politics.mideast 统计词频成功
talk.politics.misc 统计词频成功
talk.religion.misc 统计词频成功
train_count 下的文件全部创建成功
start create test_count
test-all-in-one 统计词频成功
test_count 下的文件全部创建成功
开始训练...
Group alt.atheism counts of words are 125071
Group comp.graphics counts of words are 123516
Group comp.os.ms-windows.misc counts of words are 298915
Group comp.sys.ibm.pc.hardware counts of words are 86217
Group comp.sys.mac.hardware counts of words are 73780
Group comp.windows.x counts of words are 128157
Group misc.forsale counts of words are 66339
Group rec.autos counts of words are 92788
Group rec.motorcycles counts of words are 86566
Group rec.sport.baseball counts of words are 95785
Group rec.sport.hockey counts of words are 120458
Group sci.crypt counts of words are 145501
Group sci.electronics counts of words are 87858
Group sci.med counts of words are 122349
Group sci.space counts of words are 130478
Group soc.religion.christian counts of words are 150378
Group talk.politics.guns counts of words are 143041
Group talk.politics.mideast counts of words are 202415
Group talk.politics.misc counts of words are 167789
Group talk.religion.misc counts of words are 136099
counts of words in total training set is 2583500

开始预测...
```

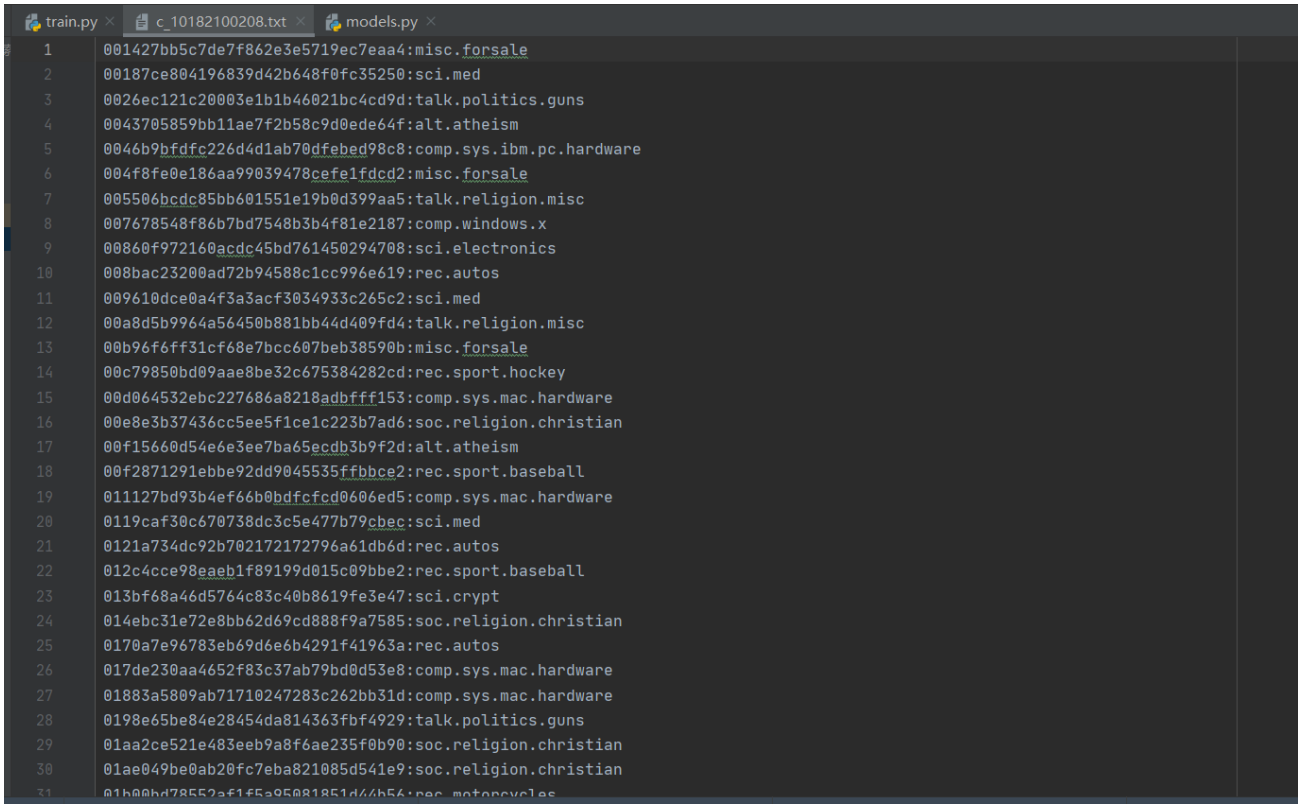
运行成功！

Process finished with exit code 0

运行结束的部分截图：



输出的结果文件：



五、总结

1.用朴素贝叶斯分类器做文本分类，依托于概率统计知识，它是一种简洁、容易理解的模型，但效果可能没有机器学习模型（比如 **SVM**、**softmax** 分类器）或者神经网络模型效果好；应用机器学习相关的方法做文本分类基本都需要做 **NLP** 相关的处理，将句子转化为句向量，在这一方面，可以使用 **word2vec**、**Glove**、**fasttext** 等方法，会大大提高准确率。而朴素贝叶斯进行分词和统计运算即可。

2.为了提高朴素贝叶斯的准确率，主要采用了三种方法：

(1)优化先验概率的统计方法，使用词频而不是文档数

(2)引入 **nltk** 的停用词表，不统计停用词，与去掉高频词效果一样

(3)对后验概率做平滑，防止零概率产生

3.为了防止统计词频的结果全部在内存中导致运行时压力太大，选择将部分结果写入磁盘，需要时再读入

4.为了提高代码可读性和模块化，尝试了使用类和对象，即面向对象编程，将一些函数封装在类内，在主函数里创建对象并调用。