山东大学 计算机科学与技术 学院

信息检索与数据挖掘 课程实验报告

学号：201600130053	姓名：王斌	班级： 16 人工智能

实验题目： 实现朴素贝叶斯分类器，测试其在 20 Newsgroups 数据集上的效果

实验内容：

#Homework 2: NBC

#实现朴素贝叶斯分类器，测试其在 20 Newsgroups 数据集上的效果。

The 20 Newsgroups dataset is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups.

#20news-18828.tar.gz（http://qwone.com/~jason/20Newsgroups/20news-18828.tar.gz）?- 20 Newsgroups; duplicates removed, only "From" and "Subject" headers (18828 documents)

实验环境：

Spyder+python3.6

Win10

实验过程和遇到的问题：

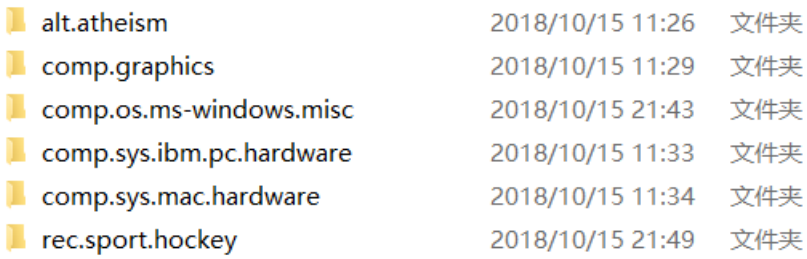
（记录实验过程中遇到的问题，以及解决过程和实验结果。可以适当配以关键代码辅助说明，但不要大段贴代码。）

核心公式：

$$P(\text{“属于某类”}|\text{“具有某特征”}) = \frac{P(\text{“具有某特征”}|\text{“属于某类”})P(\text{“属于某类”})}{P(\text{“具有某特征”})}$$

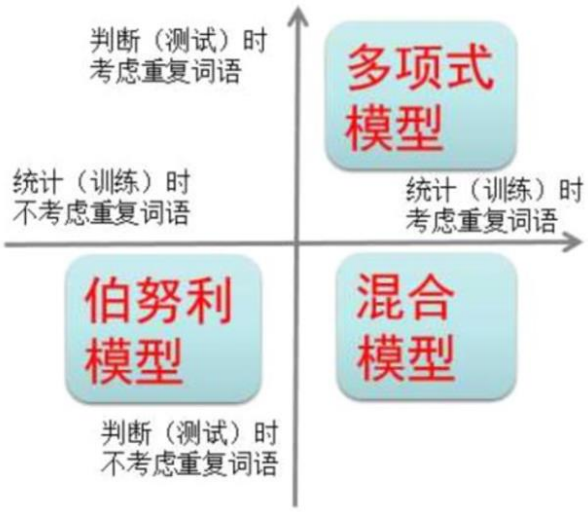
一、 划分训练样本和用来测试的样本：

按 2:1 进行划分，用来统计（训练）的样本占三分之二，测试样本三分之一，由于全部数据集太多，个人电脑跑起来比较费时，于是选取 5 到 6 类来进行实验，如下所示：



二、 选取朴素贝叶斯的模型：

一般有三种模型可供选取：***伯努利模型、多项式模型、混合模型***，伯努利将重复词只视为出现一次，会丢失词频信息，多项式统计和判断时都关注重复次数，混合模型结合前两种在测试时不考虑词频，但在统计时考虑重复次数；



结合效果发现多项式会有更好表现，于是选择了多项式模型。

三、 平滑技术的选择：

由于使用了多项式的模型，于是选择了更精确的平滑技术，计算的公式如下 eg:

平滑技术

$$\hat{P}(w_i \mid c) = \frac{count(w_i, c)}{\sum_{w \in V}\left(count(w, c)\right)}$$

$$= \frac{count(w_i, c)+1}{\left(\sum_{w \in V} count(w, c)\right) + |V|}$$

对应到本实验中即为：

$$P(\text{“term”} \mid class1) = \frac{(class1\ 中\ \text{“term”}\ 出现的总次数\ +\ 1)}{(class1\ 中总词数+词表的总词数)}$$

利用朴素贝叶斯的条件独立性假设：

$$P(x|c)=P(x1,x2\ldots xn|c)=P(x1|c)*P(x2|c)\ldots P(xn|c)$$

得到如下公式推导：

$$v_{MAP} = \arg\max_{v_j \in V} P(v_j|x_1, x_2, \ldots, x_n)$$
$$= \arg\max_{v_j \in V} \frac{P(x_1,x_2,\ldots,x_n|v_j)P(v_j)}{P(x_1,x_2,\ldots,x_n)}$$
$$= \arg\max_{v_j \in V} P(x_1, x_2, \ldots, x_n|v_j)P(v_j)$$

Using the Naïve Bayes assumption:

$$v_{NB} = \arg\max_{v_j \in V} P(v_j) \prod_i P(x_i|v_j)$$

由于本实验所选样本和测试数据集分布都比较均匀，P(Vi)可以近似相等，所以 P（Vi）可以约掉不用计算。

四、 Tricks 的应用：

使用取对数来代替很多个概率相乘，原本相乘的这些概率 P(term1|ci)P(term2|ci)P(term3|ci)，他们的值大多都非常小，所以程序会下溢出或者结果被四舍五入后得到 0，得到错误结果，使用对数可以完美解决。

而且取对数后因为 P(term|ci)的值都在 0-1 之间，而在这区间内对数函数变化很快，能够突出反映不同单词的概率对总体判断的影响，抵消部分当大量词在统计样本类别中未出现的情况，经过实验发现可以相当程度上增加准确率，如下图：

（1） 对对数进行加 1，使每个 P(term|ci)都大于 1，处于对数比较平缓的一段：

```
if term in postings1:
    p[0]+=math.log((postings1[term]+1)/(num_c1+nc1)+1)
else:
    p[0]+=math.log(1/(num_c1+nc1)+1)
```

```
8949 10810 19446 7252 7970
alt.atheism:class accuracy rate is:
0.9923076923076923
comp.graphics:class accuracy rate is:
0.018691588785046728
comp.os.ms-windows.misc:class accuracy rate is:
0.012307692307692308
comp.sys.ibm.pc.hardware:class accuracy rate is:
0.39197530864197533
comp.sys.mac.hardware:class accuracy rate is:
0.11356466876971609
```

（2） 对对数进行不加 1，使每个 P(term|ci)都在 0-1 之间，处于对数变化较快的一段：

```
if term in postings1:
    p[0]+=math.log((postings1[term]+1)/(num_c1+nc1))
else:
    p[0]+=math.log(1/(num_c1+nc1))
```

```
8949 10810 19446 7252 7970
alt.atheism:class accuracy rate is:
0.9884615384615385
comp.graphics:class accuracy rate is:
0.7819314641744548
comp.os.ms-windows.misc:class accuracy rate is:
0.012307692307692308
comp.sys.ibm.pc.hardware:class accuracy rate is:
0.9166666666666666
comp.sys.mac.hardware:class accuracy rate is:
0.8927444794952681
```

对比(1)和(2)可以发现处于对数变化较快时的准确率有较大提升(其中对于 comp.os.ms-windows.misc 类的数据发现无论怎么调整参数都无法提高准确率，怀疑该类数据的文档间没什么关联性)

五、 具体 python 代码实现细节：

使用 doc_filenames={}来分别存储在统计时记录某一个类别的所有文档的读取路径，使用 postings[term]=number_of_term 来记录建立每个类别的词典，并记录对应单词在该类别中的出现次数，使用 num_cx 来记录每个类别的总次数；total_aRate 表示总的在测试集上的准确率。

1、对于每一类统计用数据集遍历每个文档得到其 postings[]:

```
for id in doc1_filenames:
        f = open(doc1_filenames[id],'r',encoding='utf-8',errors='ignore')
        document = f.read()
        f.close()
        terms = tokenize(document)
        num_c1+=len(terms)#类 1 总词数
        unique_terms = set(terms)
        for term in unique_terms:
            if term not in postings1:
                postings1[term] = (terms.count(term))
            else:
                postings1[term]=(postings1[term]+(terms.count(term)))
```

2、Tokenize 使用和 SVM 同样的方法：

```
def tokenize(document):
    document=document.lower()
    document=re.sub(r"\W|\d|_|\s{2,}"," ",document)
    terms=TextBlob(document).words.singularize()

    result=[]
    for word in terms:
        expected_str = Word(word)
```

```
            expected_str = expected_str.lemmatize("v")
            result.append(expected_str)
        return result
```

3、对测试数据集上进行判断时，对文档进行同样的 tokenize，而后计算分别属于每个类别的概率（取对数后），选择最大的作为该文本的类别。

```
for id in doc1_test:
        f = open(doc1_test[id],'r',encoding='utf-8',errors='ignore')
        document = f.read()
        f.close()
        terms = tokenize(document)
        p=[0,0,0,0,0]
        for term in terms:
            if term in postings1:
                p[0]+=math.log((postings1[term]+1)/(num_c1+nc1))
            else:
                p[0]+=math.log(1/(num_c1+nc1))

            if term in postings2:
                p[1]+=math.log((postings2[term]+1)/(num_c2+nc2))
            else:
                p[1]+=math.log(1/(num_c2+nc2))

            if term in postings3:
                p[2]+=math.log((postings3[term]+1)/(num_c3+nc3))
            else:
                p[2]+=math.log(1/(num_c3+nc3))

            if term in postings4:
                p[3]+=math.log((postings4[term]+1)/(num_c4+nc4))
            else:
                p[3]+=math.log(1/(num_c4+nc4))
            if term in postings5:
                p[4]+=math.log((postings5[term]+1)/(num_c5+nc5))
            else:
                p[4]+=math.log(1/(num_c5+nc5))

        if p[ss]==max(p):
            count1+=1
    print(ss+1,"类名：",Newspath1[74:])
    print("判对文档数：",count1,"总的文档数：",len(doc1_test))
    total_aRate = (total_aRate+count1/len(doc1_test))
    print("准确率为：",count1/len(doc1_test))
```

六、 **最终结果展示：**

```
1 类名： alt.atheism
判对文档数： 256 总的文档数： 260
准确率为： 0.9846153846153847
2 类名： comp.graphics
判对文档数： 251 总的文档数： 321
准确率为： 0.7819314641744548
3 类名： rec.sport.hockey
判对文档数： 322 总的文档数： 330
准确率为： 0.9757575757575757
4 类名： comp.sys.ibm.pc.hardware
判对文档数： 297 总的文档数： 324
准确率为： 0.9166666666666666
5 类名： comp.sys.mac.hardware
判对文档数： 283 总的文档数： 317
准确率为： 0.8927444794952681
平均准确率为： 0.91034311414187
```

基本平均准确率能达到百分之 90 以上，达到预期实验效果。

结论分析与体会：

对于本次朴素贝叶斯的实验结果还是比较满意的，准确率达到预期，而且通过自己写的代码能完成一定的任务，有应用价值也感到比较有价值，希望再接再厉。