

## 朴素贝叶斯分类器对文本分类的应用

分类问题由两步组成：训练和预测，要建立一个分类模型，至少需要有一个训练数据集。贝叶斯模型可以很自然地应用到文本分类上：现在有一篇文档  $d$ ，判断它属于哪个类别  $ck$ ，只需要计算文档  $d$  属于哪一个类别的概率最大。贝叶斯模型应用到文本分类上，作了两个假设，一是各个特征词对分类的影响是独立的，另一个是词项在文档中的顺序是无关紧要的。

在分类问题中，并不是把所有的特征都用上，对一篇文档  $d$ ，我们只用其中的部分特征词项  $\langle t_1, t_2, \dots, t_{nd} \rangle$  ( $nd$  表示  $d$  中的总词条数目)，因为很多词项对分类是没有价值的，比如一些停用词 “is, are” 在每个类别中都会出现，这些词项还会模糊分类的决策面，因此除去这些停用词。

### 实现过程：

#### 1. 代码中几个简单函数：

Return\_File\_Name(Rootdir) # 获取文件名

file\_name(file\_dir) # 获取类(文件夹)名

Tokenization\_Stemmer(str1) # 对一个字符串进行分词，规范化

stop\_word() # 返回停用词的列表

Get\_str(filename) # 获得每一个文件的字符串

#### 2. 构建每个类的字典

把训练集中的每个类构建为一个字典。具体的操作为：遍历类中的每个文档的，每个文档为一个长字符串，然后把把这些字符串加起来，然后调用 Tokenization\_Stemmer(str1) 函数，对字符串进行分词和规范化。得到这个类的单词的列表。然后遍历列表，列表中出现的单词存为一个字典的 key，出现的次数存为 key 的 value 值，把所有的类的字典放到一个 List 列表中。具体代码：

*# 把每一个类中出现的单词存为一个字典的key，出现的次数存为value，把所有的类的字典放到一个列表中*

```
def Get_dic(name):
    List = []

    len_name = len(name) - 1

    s = ""
    for i in range(len(name)):
        if i != len_name:
            # 判断此文档是否和下一个文档是否在一个类中
            if name[i][7:-7] in name[i + 1][7:-7] or name[i + 1][7:-7] in name[i][7:-7]:
                s += Get_str(name[i])
            else:
                list1 = Tokenization_Stemmer(s)
                s = ""
                a = {}
                for j in range(len(list1)):
                    if list1[j] in a.keys():
                        a[list1[j]] += 1
                    else:
                        a[list1[j]] = 1
                Stopword = stop_word()
                # 去除停用词
                for k in range(len(Stopword)):
                    if Stopword[k] in a.keys():
                        del a[Stopword[k]]
                # 把每个类的字典添加到列表中
                List.append(a)
```

```

else:
    list1 = Tokenization_Stemmer(s)
    s = ""
    a = {}
    for j in range(len(list1)):
        if list1[j] in a.keys():
            a[list1[j]] += 1
        else:
            a[list1[j]] = 1
    Stopword = stop_word()
    for k in range(len(Stopword)):
        if Stopword[k] in a.keys():
            del a[Stopword[k]]
    List.append(a)

return List

```

3. 利用朴素贝叶斯公式对测试集每个文档和训练集中每个类进行计算，返回预测得到的类的序号。实际的计算过程中，多个概率值的连乘很容易下溢出为 0，因此转化为对数计算，连乘就变成了累加。在未转化为对数运算时，分类成功率仅有 49，在转化为对数计算时，分类的成功率则为 83 左右。代码实现：

```

#利用朴素贝叶斯公式对测试集每个文档和训练集中每个类进行计算，返回预测得到的类的序号
def Bayes(List,name):
    s = Get_str(name)
    max_p = -float('inf')
    max_name = -1
    list1 = Tokenization_Stemmer(s)
    for i in range(len(List)):
        len_list = len(List[i])
        p = 0
        for j in range(len(list1)):
            if list1[j] in List[i].keys():
                p += math.log((List[i][list1[j]] + 1) / (len_list + 10000)) #加10000进行平滑
            else:
                p += math.log(1 / (len_list + 10000))
        if p > max_p:
            max_p = p
            max_name = i
    return max_name

```

## 分类结果：

最后使用 `return_success_probability(name,file,List)` 函数返回分类成功的概率，最后测试集中共有 3161 个文档，其中 2635 个分类成功，分类成功率约为百分之 83.35。

在这个分类中只考虑了正确率也没有考虑其他评价指标，也没有进行优化。贝叶斯分类的效率很高，训练时，只需要扫描一遍训练集，记录每个类中每个词出现的次数，测试时也只需要扫描一次测试集，从运行效率这个角度而言，朴素贝叶斯的效率是很高的，相比 VSM 模型运行时间要少了几十倍，内存消耗也少，而准确率也能达到了一个理想的效果。