

人工智能实验

期中项目：分类+回归

16 级计科教务 2 班

16337327

郑映雪

（负责实现五分类）

实验题目

期中项目：分类+回归

实验分工

二分类：朱志儒

五分类：郑映雪

回归：钟哲灏、郑镇航

其中我负责五分类的数据处理和代码的编写。二分类的使用算法与我的一样，是朴素贝叶斯，关于回归，两位同学则选择了最小二乘法。

实验内容

算法原理

1、数据预处理

本次实验的数据与第一个分类实验不同，不仅是大段大段的语句，而且乱码很多，需要好好清洗。

①删除无用字符。首先，在 python 的 string 库里，有一个 maketrans 函数，利用这个函数以及设置好参数可以清除字符串里的指定内容，比如参数为 punctuation 就鉴别出所有常见标点符号，digits 就可以鉴别出所有数字等。利用鉴别+替换，可以清除与判别类型不相关的数字和标点符号。另外还发现了一些类似于-rrb-、-lrb-之类的莫名符号，可以利用列表判断把它删掉。

②使用停用词表。在分词的时候，利用停用词表对不重要的词进行过滤。我使用的停用词表是 <https://www.ranks.nl> 网站的停用词表。将一些类似于 I、you、himself 类的人称代词，以及 the 这样的冠词等没有实际意义的词语删掉，这样不仅可以减少运算时间，而且可以一定程度上规避一些可能影响准确率的词语。

③生成情感词向量。使用 word2vec 库对数据进行处理，通过计算相似的词和计算距离较近的词找出情感词汇，以便在之后的计算中加大对这些情感词汇的权重。

2、使用算法——KNN 和朴素贝叶斯

在开始对数据处理完毕之后，我想使用之前做过的 KNN 对数据进行测试，但效果不尽人意。

我对 KNN 的理解在之前的实验一有写在报告中。KNN 最近邻是监督式学习的一个机器模型。通俗来说就是，如果一个样本与训练集中的 K 个样本最为相似，那么这个样本的值就等于这 K 个样本的值中的众数。作为对“相似度”的刻画，向量之间的距离就是一个很好的标准。通过矩阵构造得出每一个文本的向量，计算距离后找出最近的 K 个向量，值即为这 K 个向量中的众数。距离计算的选择有街区距离、欧氏距离、极大距离和余弦距离等。但在 K 取不同值时以及取不同的计算距离的方法时，分类结果可能会有显著不同，所以要拨出验证集进行验证，找到使准确率最高的 K 和最佳计算距离的方法。以上是进行使用 KNN 近邻算法对文本进行分类的方法。

在本次的实验中，KNN 的状况并不理想——这次数据会太大了，大矩阵运算会让电脑卡死，而且跑结果的速度特别慢。我对每一次跑结果进行了测试，发现电脑总是卡死在 numpy 的大矩阵处理上——这下大矩阵是没辙了。电脑放空一晚上，终于跑出了结果——20%准确率，还不如全写 0 呢。所以我放弃了 KNN，选择了朴素贝叶斯。

朴素贝叶斯基于贝叶斯定理。它基于特征条件独立学习输入和输出的联合概率分布，然后根据贝叶斯定理求出拥有最大概率的值。具体算法如下：

假设训练数据集可以表示为：

$$T = \{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$$

设先验概率分布，即各个类的概率为

$$P(Y = c_k), k = 1, 2, \dots, K$$

设条件概率分布，即各个类下各属性取值的概率为：

$$P(X = x^j | Y = c_k), k = 1, 2, \dots, K$$

在朴素贝叶斯法中，各个条件分布是独立的，所以上式可化为：

$$P(X = x^j | Y = c_k) = \prod_{j=1}^n P(X^j = x^j | Y = c_k)$$

根据贝叶斯定理：

$$\begin{aligned} P(Y = c_k | X = x) &= \frac{P(X = x | Y = c_k)P(Y = c_k)}{\sum_k P(X = x | Y = c_k)P(Y = c_k)} \\ &= \frac{\prod_k P(X^j = x^j | Y = c_k)P(Y = c_k)}{\sum_k \prod_k P(X^j = x^j | Y = c_k)P(Y = c_k)} \end{aligned}$$

由于分母对于同一类的数据都是一样的，所以只要比较分子的大小就可以了。所以我们得到 y 的一个判断公式，这个判断公式来源于风险最小化原则。

$$y = \arg \max P(Y = c_k) \prod_j P(X^j = x^j | Y = c_k)$$

3、朴素贝叶斯应用于五分类

在五分类中，朴素贝叶斯的应用步骤如下：

①计算先验概率、条件概率。

队友在二分类上使用朴素贝叶斯，数据结构是较为简单的。但是五分类如果设置五个列表，那代码会非常紊乱。所以我选择字典嵌套字典的结构（外层键名为类别，内层键名为不重复词汇表的每个单词）来计算先验概率和条件概率。先验概率计算统计训练集各类的个数再除以总数即可；条件概率在每个分类下设置各个词语的字典，如果该词语在文本中，则键名为该词语的键值+1。这样一来可以直截了当地算出在各类下每一个词语的比重，即算出条件概率。

②对测试集的每一条数据计算。

由上一条原理我们知道，对于

$$\frac{\prod_k P(X^j = x^j | Y = c_k)P(Y = c_k)}{\sum_k \prod_k P(X^j = x^j | Y = c_k)P(Y = c_k)}$$

由于每一个分类下的分母是一样的，所以我们对测试集的每一条数据，只需要计算该式的分子即可。由此我们可以算出五个分类下的不同数据。

③找出最大概率，确定类别。

根据 $\arg \max P(Y = c_k) \prod_j P(X^j = x^j | Y = c_k)$ 确定类别，即找出概率最大的那一类即为 x 的预测类别。

对于 $P(X^j = x^j | Y = c_k)$ ，有可能会存在分子为 0 的情况，这个时候我们可以采用拉普拉斯平滑，其中 S_j 为该属性下取值的个数：

$$P(X^j = x^j | Y = c_k) = \frac{\sum_{i=1}^N I(x_i^j = a_{ji}, y_i = c_k) + \lambda}{\sum_{i=1}^N I(y_i = c_k) + S_j \lambda}$$

$$P(Y = c_k) = \frac{\sum_{i=1}^N I(y_i = c_k) + \lambda}{N + K \lambda}$$

利用拉普拉斯平滑，可以在一定程度上提升正确率，同时对 λ 值的调参也可以提升正确率。

至此，我们可以看到，朴素贝叶斯没有复杂的大矩阵运算，仅采用 python 自带的基本数据结构就可以完成计算，实践也证明，跑朴素贝叶斯的代码时，速度很快，而且电脑也不会内存占用过大而卡死。朴素贝叶斯的学习和预测的效率都较高。

以李航《统计学习方法》里的数据为例：

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
x1	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
x2	S	M	M	S	S	S	M	M	L	L	L	M	M	L	L
y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

采用拉普拉斯平滑，取 $\lambda=1$ ：

$$P(y = 1) = \frac{10}{17}, P(y = -1) = \frac{7}{17}$$

$$P(x1 = 1|y = 1) = \frac{3}{12}, P(x1 = 2|Y = 1) = \frac{4}{12}, P(x1 = 3|y = 1) = \frac{5}{12}$$

$$P(x2 = S|y = 1) = \frac{2}{12}, P(x2 = M|Y = 1) = \frac{5}{12}, P(x2 = L|y = 1) = \frac{5}{12}$$

$$P(x1 = 1|y = -1) = \frac{4}{9}, P(x1 = 2|Y = -1) = \frac{3}{9}, P(x1 = 3|y = -1) = \frac{2}{9}$$

$$P(x2 = S|y = -1) = \frac{4}{9}, P(x2 = M|Y = -1) = \frac{3}{9}, P(x2 = L|y = -1) = \frac{2}{9}$$

给出测试集数据，设 $x = (2, S)$ ，计算：

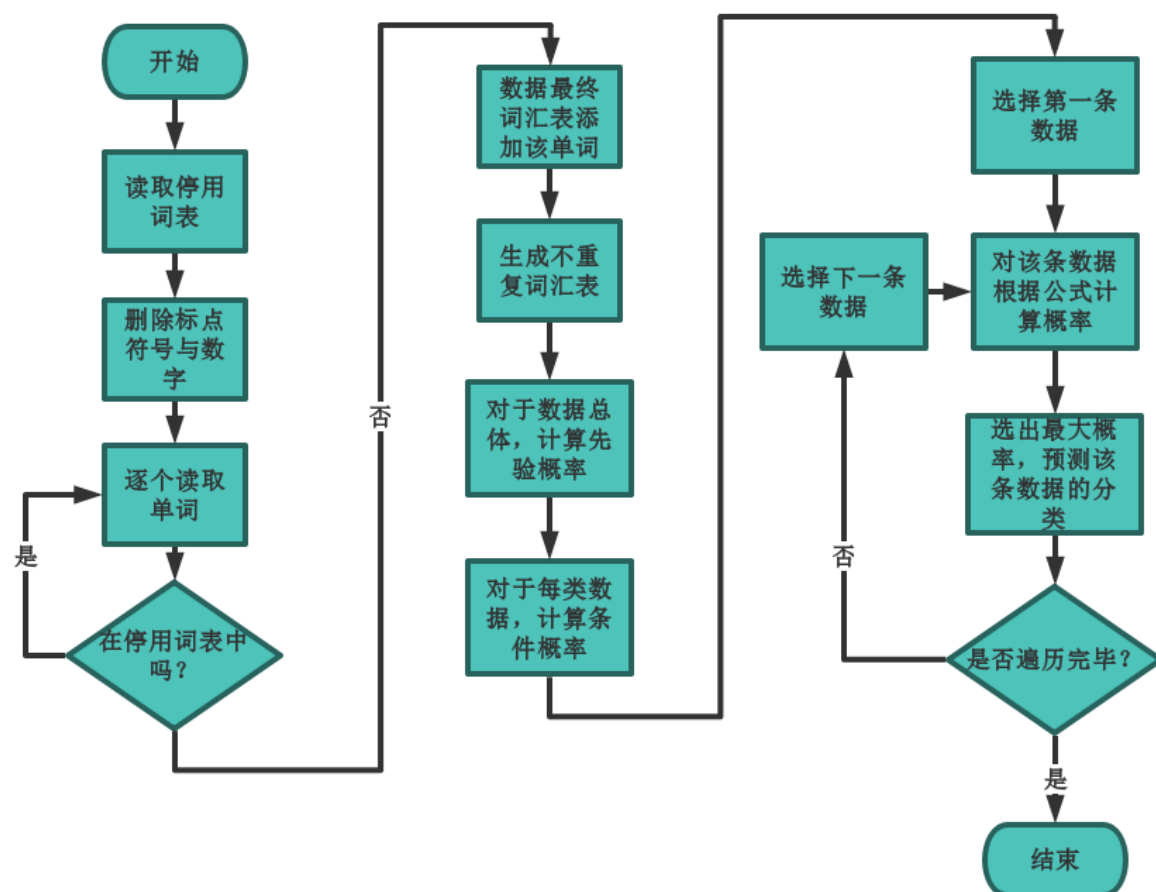
$$P(y = 1)P(x_1 = 2|y = 1)P(x_2 = S|y = 1) = \frac{5}{153}$$

$$P(y = -1)P(x_1 = 2|y = -1)P(x_2 = S|y = -1) = \frac{28}{459}$$

后者较大，所以将 x 分类到 $y=-1$ 处。

在本实验中，每个单词就是属性，每个训练集的属性取值有 0、1 两种情况，即“该词在此句中”或“该词不在词句中”。

流程图



关键代码

KNN 的代码与实验一大致相同，加上本次实验我主要是在 NB 算法上进行参数调试和数据处理，所以不贴上来了。下面贴上我主要花时间和精力的关键代码：

1、数据处理

对于这次实验的数据，处理数据是非常关键的一步。首先我从网上找到了常用的停用词表，利用停用词表筛去无用词汇。同时，使用 string 库的 maketrans 函数，删去所有的标点符号和数字，以及莫名其妙出现的 rrb、lrb 乱码，达到数据的纯净性。

```
import string
dictionary = []
with open('vocabulary.txt', 'r', encoding='UTF-8') as file: # 读取训练集的句子
    for line in file:
        line = line.strip('\n')
        dictionary.append(line)
dictionary.append('rrb')
dictionary.append('lrb')
data = []
table = str.maketrans(string.punctuation + string.digits,
                      ' ' * (len(string.punctuation) + len(string.digits))) #
删除标点符号和数字
```

2、生成不重复词汇表

对于分好词汇的训练集，我们生成不重复词汇表，以词汇表里的每个词作为一个属性，以便进行下面的操作。

```
def tran(data):
    voca = []
    for sentence in data:
        tmp = sentence.split(' ')
        for tmp1 in tmp:
            voca.append(tmp1)
    voca2 = set(voca) # 利用 set 函数去除列表中重复元素
    return voca2
```

3、计算训练集的先验概率和条件概率。

此部分代码如下图所示：其中有 4 个参数，data 为处理过后的训练集数据，vari 为不重复词汇表，class1 是一个字典，结构如下：

{‘0’：{词汇名：在该分类下的数据出现该词的次数}，‘1’：{词汇名：在该分类下的数据出现该词的次数}………}

先验概率较为好算。这里还使用了这样一个字典嵌套字典的结构，方便计算五个分类下的条件概率。

```
def calprobability(data, vari, class1, emotion):#计算先验概率和条件概率
    index = 0
    for i in range(5):
        for varis in vari:
            class1[str(i)][varis] = 0
    for datas in data:
        tmp = datas.strip(' ').split(' ') # 将每个词语分割成词组
        for varis in vari:
            if varis in tmp:
                class1[emotion[index]][varis] += 1#计算各属性取值 0 或 1 (在或不在该条数据中)
        print(index, " 成功")
        index += 1
```

4、对每一个测试集数据估计其分类

对于每一条测试集数据，首先也是需要使用 1 中的代码进行数据清洗的。部分参数解释：vadata 为测试集数据，testemotion 为估计的情感（传入函数时空列表），class1 用于计算条件概率，class2 也是一个字典，用于计算先验概率。

首先设置循环求每一个分类的先验概率，即 $P(y = i)$ ，然后对于每一条测试集的数据，代入下面的公式：

$$\prod_k P(X^j = x^j | Y = c_k) P(Y = c_k)$$

注意这个公式需要应用在每一个分类上，最后利用 python 里的 index 函数返回最大值的下标，即返回使公式值最大的类别，该类别即为这个数据的预测类别。


```

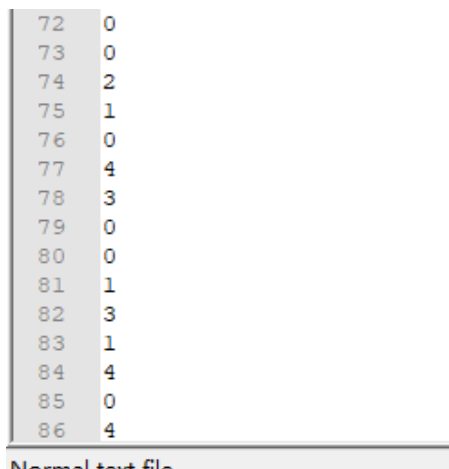
def classification(vadata, testemotion, class1, class2, sumdic, length,
vari):#根据朴素贝叶斯公式计算并预测分类
    pclass2 = {}
    for i in range(5):
        pclass2[str(i)] = math.log((class2[str(i)] + 18) / (length + 5)) # 拉
普拉斯平滑求  $P(Y=i)$ 
    sum1 = 0
    for vadatas in vadata:
        tmppro = []
        tmp2 = vadatas.strip(' ').split(' ')
        for i in range(5):
            probab = pclass2[str(i)]
            for voca in tmp2:
                if voca in vari:
                    probab += math.log((class1[str(i)][voca] + 18) /
(sumdic[str(i)] + 2)) #乘法转化为取对数后的加法
            tmppro.append(probab)
        testemotion.append(tmppro.index(max(tmppro)))
    print(sum1, ' 预测成功')
    sum1 += 1

```

实验结果及分析

实验结果展示

最终使用朴素贝叶斯跑出的结果部分截图如下：



72	0
73	0
74	2
75	1
76	0
77	4
78	3
79	0
80	0
81	1
82	3
83	1
84	4
85	0
86	4

单纯看结果的话，可以看出 4 和 0 预测的值比较多，其中不同的 λ 值测出的结果侧重点不同。比如 λ 值过小的话，那么整体数据就更偏向 0 一些。

1	0
2	2
3	0
4	1
5	0
6	0
7	0
8	3
9	0
0	0
1	0

KNN 算法我没有进行结果输出，大矩阵运算会让我的电脑持续卡爆，所以我干脆跑了一个指标出来就专注于写朴素贝叶斯算法了。

评测指标展示

利用朴素贝叶斯方法，取训练集前 18000 个为训练集，后 6000 个为验证集，准确率如下所示：

```
5997 预测成功
5998 预测成功
5999 预测成功
0.3401666666666667
```

通过调节 λ 的值和对数据的各种不同的方式处理，准确率本身是有变化的，但在 0.34 左右浮动。并且到了真正实战上 rank 的时候，不知为什么总是上不了 34。我思考了一下觉得这是因为我的验证集可能跟测试集的分布不同，所以我在最后关头垂死挣扎了一波，提交了很多个不同参数、不同情况下的结果上去。