中山大学数据科学与计算机学院
移动信息工程专业-人工智能
本科生实验报告
（2016 学年秋季学期）

课程名称：Artificial Intelligence

教学班级	14M2	专业（方向）	移动互联网
学号	14353205	姓名	刘万里

一、 实验题目

利用朴素贝叶斯的方法对数据集进行处理，分别得到分类和回归的结果。

二、 实验内容

1. 算法原理

分类：

需要一个数组记录每种感情对应的词汇总数；

还需要一个数组记录每种感情对应的不重复的词汇总数，用来做拉普拉斯平滑

也需要每种感情下每个词汇对应的出现次数，因此可以用 map<string,int>来存储，然后再用 vector 来 push6 个这样的 map，就可以用对应的感情下标去寻找那个感情所拥有的词汇了。

回归：

先按照实验 1 得到 TF 矩阵，然后在此基础上，读取并处理测试集，每得到一行测试集就按照公式去进行每种感情的概率计算，其中进行拉普拉斯平滑即可。

2. 伪代码

对应的读取文件的操作都不再赘述：

分类：

```
double pro[7] ; //6 种情绪的概率
Pro[7] = {1};
while(ss>>dict)  //读入每行的数据集内容的词汇
{
    or(int i = 1 ; i <= 6; i++)
    {
    //计算每种情绪下这个单词的概率乘积因子，并且在这里进行拉普拉斯平滑
        it1 = 对应的情绪的 map 对应单词的迭代器;
        if(it1 == 空)
    pro[i] *= 拉普拉斯平滑因子;
        else
            pro[i] *= 出现次数 / 该情绪的词汇数量;
        }
        double max = 0;
        int ans;
        for(int i = 1 ; i <= 6; i++)
        {
            pro[i] *= p(i 代表的情绪的概率)
```

```
        if(pro[i] > max)
        {
            max = 这个已知最大的概率;
            ans = 这种情绪;
        }
    }
    if(ans == emo) 正确次数++;
```

回归：

```
//读取了测试文档的一行
double pro[7];
double sum = 0;
pro[i] = {0};
for(int j = 1; j <= 6; j++)  //感情
{
    for(int i = 0 ; i < TRAINLINES; i++) //训练行数
    {
        double k = 1.0;
        double sum_xk = 0;
        得到这行的词汇在第 i 行的 TF 值之和
        for(int x = 0 ; x < thisLine.size() ;x++)//这行的词汇个数
        {
            //这里用了拉普拉斯平滑
            if(x 这个词汇在第 i 行的 TF == 0)
                K = 拉普拉斯平滑值
            else
                k *= TF 值;
        }
        k *= 第 i 行为第 J 种感情的概率;
        pro[j] = 每一行的概率之和;
    }
    sum += pro[j];//计算概率之和用来归一化
}
归一化每一项并输出到文档中
```

3. **关键代码截图（带注释）**

分类：

维护每个情绪对应的存储词汇的数据结构 map<string(词汇)，int（出现次数）>， 并记录好每种情绪出现的不重复的单词个数

```
void Cal_Dicts_Emos(int emo,string dict)
{
    switch (emo)
    {
            case 1:
            it1 = EMO1.find(dict);
            if(it1==FirstIndex.end())//迭代器到了end()，说明这个词在这种感情还没出现过
            {
                FirstIndex.insert(pair<string,int>(dict,1));
                dictsOfEmotion_NoRepeat[1]++; //护好dictsOfEmotion_NoRepeat[7]，来做拉普拉斯平滑-
            }
            else
                it1->second++;
            break;
```

对测试集进行概率计算和分类：

```
double pro[7] ; //6种情绪的概率
for(int i = 1; i <= 6; i++) pro[i] = 1;
string dict;
while(ss>>dict)  //读入每行的数据集内容的词汇
{
    for(int i = 1 ; i <= 6; i++)
    {
        //计算每种情绪下这个单词的概率乘积因子，并且在这里进行拉普拉斯平滑
        it1 = DictsEmos[i - 1].find(dict);
        if(it1 == DictsEmos[i - 1].end())
            pro[i] *= (1.0/(dictsOfEmotion_Repeat[i] + dictsOfEmotion_NoRepeat[i]));
        else
            pro[i] *= ( 1.0 * it2->second / dictsOfEmotion_Repeat[i]) ;
    }
}
```

```
double max = 0;
int ans;
for(int i = 1 ; i <= 6; i++)
{
    pro[i] *= (1.0 * emo_times[i] / TRAINLINES);
    //cout<<pro[i]<<endl;
    if(pro[i] > max)
    {
        max = pro[i];
        ans = i;
    }
}
cout<<ans<<endl;
if(ans == emo) rightTimes++;
```

最后用得到的 rightTims/测试文本样例行数就可以得到正确率！

回归：

```
double pro[7];
double sum = 0;
for(int i = 1 ; i <= 6; i++) pro[i] = 0;
for(int j = 1; j <= 6; j++)  //感情
{
    for(int i = 0 ; i < TRAINLINES; i++) //训练行数
    {
        double k = 1.0;
        double sum_xk = 0;
        for(int x = 0 ; x < thisLine.size() ;x++)
        {
            sum_xk += tf[i][ FirstIndex.find(thisLine[x]) -> second];
        }
        for(int x = 0 ; x < thisLine.size() ;x++)//这行的词汇个数
        {
            //这里用了拉普拉斯平滑
            if(tf[i][ FirstIndex.find(thisLine[x]) -> second] == 0)
            k *= (( tf[i][ FirstIndex.find(thisLine[x]) -> second] + 1 ) /(sum_xk + thisLine.size()) );
            else
            k *= tf[i][ FirstIndex.find(thisLine[x]) -> second];
        }
        k *= P[i][j];
        pro[j] += k;
    }
    sum += pro[j];
}
```

```
//归一化并输出到文档中
for(int j = 1; j <= 6; j++)
{
    pro[j] = pro[j] / sum;
    output<< pro[j] <<" ";
}
output<<endl;
```

4. 创新点&优化

无

三、 实验结果及分析

1. 实验结果展示示例（可图可表可文字，尽量可视化）

分类：

得到的分类结果几乎全部都为 2。

```
D:\grade_three\one\Artifical_Intelligence\lab3_贝利斯算法实现\Classifica...
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
2
分类正确率为0.026000
```

回归：

validation 结果：

	anger	disgust	fear	joy	sad	surprise
r	0.028639	-0.039	0.009769	0.066102	0.006215	-0.04914
average	0.003764					
evaluatic	极弱相关	加油哦				

Test 结果：

I	J	K	L	M	N	O
	anger	disgust	fear	joy	sad	surprise
r	0.081604753	-0.00458	0.001673	0.036954	-0.04118	-0.06038
average	0.002348012					
evaluatic	极弱相关 加油哦					

Naive Bayes 算法：

设每个数据样本用一个 n 维特征向量来描述 n 个属性的值，即：X={x1，x2，…，xn}，假定有 m 个类，分别用 C1, C2,…，Cm 表示。给定一个未知的数据样本 X（即没有类标号），若朴素贝叶斯分类法将未知的样本 X 分配给类 Ci，则一定是

$P(C_i|X)>P(C_j|X)\quad 1\leq j\leq m,\ j\neq i$

根据贝叶斯定理

由于 P(X)对于所有类为常数，最大化后验概率 P(Ci|X)可转化为最大化先验概率 P(X|Ci)P(Ci)。如果训练数据集有许多属性和元组，计算 P(X|Ci)的开销可能非常大，为此，通常假设各属性的取值互相独立，这样

先验概率 P(x1|Ci)，P(x2|Ci)，…，P(xn|Ci)可以从训练数据集求得。

根据此方法，对一个未知类别的样本 X，可以先分别计算出 X 属于每一个类别 Ci 的概率 P(X|Ci)P(Ci)，然后选择其中概率最大的类别作为其类别。

朴素贝叶斯算法成立的前提是各属性之间互相独立。当数据集满足这种独立性假设时，分类的准确度较高，否则可能较低。另外，该算法没有分类规则输出。

优点：

一、朴素贝叶斯模型发源于古典数学理论，有着坚实的数学基础，以及稳定的分类效率。

二、朴素贝叶斯模型所需估计的参数很少，对缺失数据不太敏感，算法也比较简单。

缺点：

一、理论上，朴素贝叶斯模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为朴素贝叶斯模型假设属性之间相互独立，这个假设在实际应用中往往是不成立的（可以考虑用聚类算法先将相关性较大的属性聚类），这给朴素贝叶斯模型的正确分类带来了一定影响。在属性个数比较多或者属性之间相关性较大时，朴素贝叶斯模型的分类效率比不上决策树模型。而在属性相关性较小时，朴素贝叶斯模型的性能最为良好。

二、需要知道先验概率。

三、分类决策存在错误率