

中山大学移动信息工程学院本科生实验报告

(2017年秋季学期)

课程名称: *Artificial Intelligence*

| | | | |
|----|-------------|--------|--|
| 年级 | 1501 | 专业（方向） | 移动互联网 |
| 学号 | 15352015 | 姓名 | 曹广杰 |
| 电话 | 13727022190 | Email | 1553118845@qq.com |

一、实验题目

二、实现内容

算法原理

伪代码

关键代码

结构体框架

对当前数据的情感分析

count_words函数的实现

get_size函数的实现

创新点与优化

三、实验结果与分析

四、思考

一、实验题目

Classification for Naive Bayes

二、实现内容

1. 使用Bayes处理分类问题，只要求实现多项式方法。
2. 使用Bayes处理回归问题，使用相关系数衡量实验结果。

算法原理

Bayes方法是对于同样的事件，通过先验概率推测后验概率的方法。分为伯努利模型与多项式模型，本次实验中实现多项式模型。

在多项式模型中，遵循一个假设——即所有词的出现都是各自独立的，不会因为出现了“Same”前面就一定要加一个定冠词。这种情况下，对于每一个词出现的概率就可以估算了，本次实验中使用频率代替概率。

所以当一句话中出现了很多词，每一个词在出现的概率都各自独立，既然这些词恰好组成了当前的句子，就应该把这些词出现的概率相乘。而对于不同的情感来说，每一个词在不同的情感中出现的概率都不太一样，这样计算出的不同情感的权值也就不太一样。

通过这些概率，我们可以在后续处理无论是分类问题还是回归问题中都有所依仗。

伪代码

Bayes的伪代码与之前Knn的伪代码非常类似，都是对相关数据的导入与关联函数的调用。整体流程如下：

```
int main(int argc, char** argv) {
    // 读入训练数据并分析得到数据库
    // 根据数据库内容，计算更新训练数据内容

    valid("validation文件名");
    for(遍历validation的句子){
        // 储存为元组格式
        // 分析label
        // 拟合为输出容器
    }

    fout << "id" << ',' << "label" << endl;
    for(遍历输出文件){
        fout << i+1 << ',' << "获得的label" << endl;
    }
    // 对输出的文件内容与validation的文件内容进行分析
    // 计算准确度
    return 0;
}
```

关键代码

在笔者实现本次实验之前，需要设计全局变量，作为贯穿整个project的数据库，这些数据将会作为比较和计算的标准：

1. 所有词的有序集合（以出现顺序为序）
2. 元组的集合，即训练集本身
3. validation文件的句子的集合，将会作为处理validation中每一个文本时候的遍历索引

结构体框架

由于处理文本的缘故，本次实验中笔者依然使用了Tuple作为储存和管理一个文件的一行的信息的数据结构：

```

class Tuple{
public:
    void Collectvoc(string s, bool fix_base);
    void Updatebaserow();
    void Register(string s){
        Collective(string s, bool fix_base);
        Updaterow();
    }
    double Distanceof(Tuple tuple);
    void taste();
    vector<bool> row; // 在所有词出现的顺序集合中，该行中的词所占的位置
    vector<string> voca; // 该行中出现的词
    string label; // 标签
};

```

在拥有了这个数据结构之后，所有的文件通过特定的函数调用不断地满足以上的变量信息，就可以进行后续的操作。

笔者在这里处理训练集的时候，是依照原有的分类函数进行修改的，整体格局相近，但是由于读入的字符串更加繁杂琐碎，使得字符串处理的部分占据了很大的篇幅，这种修改的结果就是可读性不强，凌乱。笔者这里就不过多地贴码了，因为这一部分与主体思路偏离。读取文件之后，需要对当前需要识别的信息进行注册和分类。

在实现分类的时候，Bayes的处理方法是对当前数据的每一个情感都进行权值的计算，取最大值作为当前数据的情感。由于在实现的过程中，测试数据总看你出现训练数据中未曾出现的数据，这就直接导致输出信息为0，为此实验要求使用Laplace平滑对数据进行加工。

对当前数据的情感分析

按照我们之前所说的，当前的情感信息取决于每一个词出现的概率相乘。这时候，对于某一个词出现的概率的计算就很有发挥的余地了，毕竟我们是以频率估算概率，这种估算方法总是花样繁多。在本次实验中，经过Laplace平滑的修正，最后的计算公式如下：

词A出现的概率 = (A在感情a中出现的频率+1) / (感情a的词数+总词数) ¹

为了获得了每一个词在不同的情感中的概率，笔者按照文档上给出的信息进行编码，得到逻辑如下：

```

int numofv = count_words(crt_em, crt_v);
// 情感crt_em的词库量-含重复词
int rptcnt = get_size(crt_em, true);
double tmpp = (double)(numofv+1)/(rptcnt + vocbase.size()); //vocbase是全局变量

```

这种计算将会使得我们得到：

- 在当前测试句子内
- 每一个词的权值的乘积

在后续的实现中，只需要将不同情感下，这些词的乘积值进行比较，就可以获得权值最大的情感标签了。然而，为了获得以上的几个计数，笔者还是要进行一些函数的实现，包括count_words和get_size。

count_words函数的实现

- 顾名思义，该函数用于计算情感下该词的出现次数。
- 有两个参数：情感（字符串），被查询的词（字符串）；

```

if(emotion == "joy"){// emotion 为传入的参数，表示要在哪个数据库内查询
    if(joylabel_.size() == 0) return 0;// joylabel_表数据库，为vector，repeatable
    int cnt = count(joylabel_.begin(), joylabel_.end(), word);// STL计数count
    return cnt;
} //其他情感以此类推

```

get_size函数的实现

- 意义：查询当前词库的词量
- 参数：感情（字符串），是否含重复词（bool）

```

if(emotion == "joy"){// emotion 和 repeat 分别为传入参量
    if(repeat) return joylabel_.size();
    else return joylabel.size();
}

```

有了这些函数的帮助，我们就可以实现在公式中给出的那些信息了。于是也就知道了不同情感下，每一个词的权值，对每一种情感下的词的权值进行乘积，结果正是：

在这种情感下，出现这句话的概率

再选择一个出现概率最大的情况，即为解。

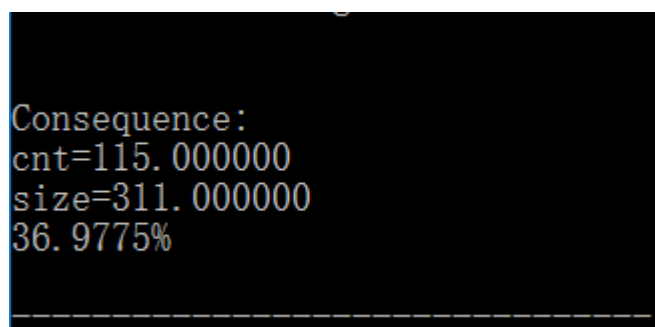
创新点与优化

总的来说，本次实验的实验要求中规中矩。按照要求实现的代码在最后都可以得到类似程度的效果。对于优化而言，由于分子使用的是在当前情感环境下对某一个词的计数信息，这个词出现的次数，会对我们计算的结果添加权值信息。而分母中的词重复词也在一定程度上抵消了这种加权的作用。这种操作使得计算的结果不再受训练集中某种情感的数量的影响。

可以优化的地方，是对于不同词的区别加权，对于介词和形容词的区别对待——然而笔者还没有实现。但是笔者放弃了在实验要求中，使用可重复集数量与非重复集数量的和的部分。改为使用全局的词库数量，使得计算值更为准确。

三、实验结果与分析

1. 实验结果示例展示



```

Consequence:
cnt=115.000000
size=311.000000
36.9775%
-----

```

2. 评测指标展示及分析

测评指标为validation的符合度，这里显示为36.9775%。

这个符合度是基于训练集的所有信息的计算结果，在达到了Knn计算的平均水平的基础上，并没有给我们带来什么惊喜。

- 分析原因：

尽管当前算法对于所有的词都进行了一定程度的拟合，产生结果也差强人意。但是很大程度上，并没有逃离相似度比较的模式。

- 对于每一个词都同等看待；
- 计算其出现的频率

这两点，本身就是在延续使用相似度判决的方法，所以出现Knn的平均水平也就不足为奇，甚至，Knn根据近邻值域的选择，还使得筛选更具有不确定性【笑】，反而误打误撞地提升了准确度。

四、思考

- 为什么Knn相似度加权的时候，要将距离的倒数作为权重

距离表示当前检查的元组与比较的元组之间的距离，距离越长则这两个元组的相似度越低，对照的元组参考性就越低，这种情况下，参考性与距离之间就呈负相关关系。因而，使用反比例函数是符合要求的。

- 如何归一化

如果当前的情感系数都不为0，那么可以对所有的系数做和，再将每一个系数除以系数之和，以达到归一化的目的。

如果当前的情感系数都未被检测到，可以都设置为1/6，但是这种方法的对该问题的符合度不高。

- 矩阵稀疏度不同的时候，曼哈顿距离与欧式距离有什么不同

曼哈顿距离与欧式距离都是距离范式中的一种形式，曼哈顿距离取参数值为1，欧式距离取参数为2。

当矩阵非常稀疏的时候，进行计算的数组中都具有很多的0，而非零的数据密度极低。在计算距离的时候，如果非零的值大于1，则曼哈顿距离计算得到的距离更大，而值小于1的时候，计算的距离更小——所谓波动都比欧式距离更小，因此，欧式距离的计算结构更加稳定。

结合矩阵的稀疏度，对于不同的矩阵：

- 使用曼哈顿距离计算得到的数据波动幅度更大，可以作为区分矩阵的特征量
- 使用欧式距离计算得到的数据幅度波动较小，可以限定对数据的过度拟合分析

- 伯努利模型与多项式模型各有什么优缺点

伯努利模型的重点集中于文本的数量，伯努利模型是以文本为计算单位的。而多项式模型是以词为单位的。

伯努利模型在应对区分度大的文本集的时候，优势大于多项式模型，伯努利模型的处理方式决定了其能够很清楚地分清，当前这个词，在文本中到底有多大的价值，如果在此类文本中，出现这个词的概率过于小，那就说明该词并不能被人们接受地表达该类别文本的意思。但是伯努利模型在处理少量数据的时候就显现出巨大的劣势——如果我们的文本数量少得可怜，但是文本的内部具有大量的信息可以挖掘，伯努利模型在这方面的表现很显然是不足的。

多项式模型对文本之间的区分并不是非常敏感，多项式模型在分析一个词的时候，参考的主体是整个词库，这个词库表达的内容繁多而且值得挖掘，如此，对于这个词的理解就很深刻。但是同时多项式模型淡化了文本的概念，过度放大了出现次数所占的比重，这种特点的结果就是对介词副词等无实际意义的连接词没有任何抵抗力，非常容易混淆视听。

- 如果测试集中出现了一个全词典都没有出现的词应该如何处理

常规来讲我们是没有任何办法的，在Bayes的regression实验中，有大量的测试数据都没有在训练集中出现过，这种情况下，就不得不放弃对于整个句子的句意的猜测。

如果该句子中还有其他的词汇，我们可以降低这个未知的词在分析过程中所占的比重，将重点转移到其他已知内涵的词语中。

但是正如Bayes的regression实验中，有的句子中一个词汇都没有出现过，这种情况下就真的是一筹莫展，就算是人，也鲜有办法。

1. 师兄的这个公式是错的啊，记得改，下不为例~[👉](#)