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
整体流程
关键代码
处理训练集
当前数据分类
近邻值的选取
创新点与优化
三、实验结果与分析
关于准确度的提升
测评指标展示：
四、思考

一、实验题目

K近邻模型处理分类问题

二、实现内容

1. 使用KNN处理分类问题。在验证集上，通过调节K值、选择不同距离等方式得到一个准确率最优的模型参数，并将该过程记录在实验报告中。
2. 在测试集上应用步骤1中得到的模型参数（K，距离类型等），将输出结果保存为“学号*姓名拼音*KNN_classification.csv”，

算法原理

KNN模型在应对大量的数据集的时候，只选择与测试数据相近的信息，缩小计算范围由此减少计算量——这种方式减少了与测试样本无关的数据对样本可能产生的影响。

KNN问题处理数据分类。KNN会在已有的诸多集合中通过**距离计算**，选择K个有价值的数据，**缩小计算的范围**。这里有价值的数据就称为近邻数据，这些数据将会影响到当前数据的分类，指导当前数据实现分类操作。

Knn处理数据回归。Knn会根据定义的距离计算方式获得与当前处理信息最贴近的数据，并根据已有的数据使用**特定的算法**获得每种情感的比例。

Knn方法的不足。Knn方法当然不是完美的，这种方法在消除了无关数据对数学模型的影响之后，同时放弃了其他有关模型或者关联性较弱的数据对于当前模型的加权意义。这就使得Knn方法可以在大致方向上与其同类的消息匹配得很好，但是在数据的调整上很少有发挥的余地。

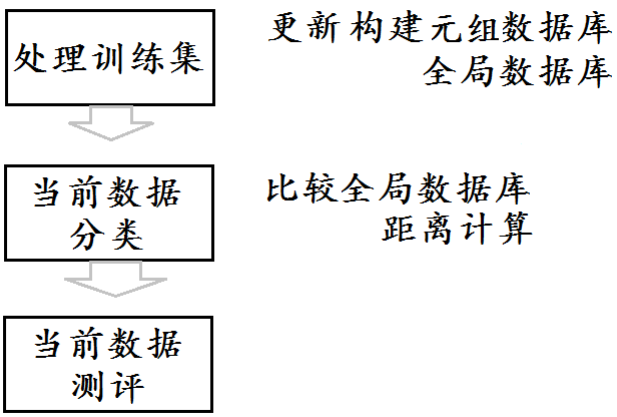
- 距离的计算：距离的计算可能有多种方式。包括曼哈顿距离、欧式距离以及余弦距离等。所谓距离计算方式是对当前的分类操作建立一个数学模型，对于实际问题而言，不同的数学模型可能会有不同的效果，这也取决于我们的模型的构建以及数学关系的嵌套是否可以与实际问题符合得更好。
- 缩小计算范围。当笔者在分析当前的一条数据与数据集中的所有数据时，我们不敢保证我们的算法可以发现每一条信息隐藏的关系——这种情形下，如果一定要将与当前处理的数据关系微弱的信息纳入当前的数学模型，十有八九会使得模型为了迁就该部分数据而导致数学模型的波动与失衡。所以，Knn就设计模型以避开这个问题，Knn只选择关系最紧密的几个近邻值纳入数学模型，这就十分有效地避免了无谓数据对模型的影响。
- 特定的算法：与距离的计算很类似，需要根据具体的情况，结合实际问题，创建数学模型以解释当前的情况。

伪代码

在实现数据处理之前，我们需要构建几个全局通用的数据库。

- 所有词的数据库，词之间互不重复。该数据库将用于构建one hot矩阵。
- 训练集每一行的信息。该数据库将用于在训练集巨大的数据量中筛选与当前数据最靠近的几个近邻值，用于作为遍历时候的索引。

整体流程



关键代码

为了处理数据的方便，笔者创建了一个新的类，专门用于储存train数据集中每一行的数据，并为之设立了必要的接口。

```
class Tuple{
public:
  // 消化分析一行的字符串，以更新成员变量;
    void Dismantle(string s, bool fix_base);//bool-是否添加入数据库中
    void Updaterow();
    void Register(string s){
        Dismantle(s, false);
        Updaterow();
    }
    double Distanceof(Tuple tuple);

    vector<double> row;//one hot的一行（在整体词集中，这一行的词是否出现过）
    vector<string> vca;//句子转化为词集
    string label;//标签
};
```

接下来的代码分析，笔者依据在“伪代码/整理流程”中的逻辑进行实际实现。

处理训练集

```
while(逐个字符读入){
    if(字符 == '\n'){
      Tuple tp;
      tp.Dismantle(s_tuple, true); // 更新一行内的数据
      Tuplebase.push_back(tp); // 更新全局数据库
      s_tuple = ""; // 收集字符的容器字符串清零
    }else{
      s_tuple += c; // 收集当前的字符
    }
}
```

在处理训练集数据的时候，我们主要的任务还是构建用于后续计算的数据库，以及更新数据库。全局数据库是一个方面，另一个方面，在查询k个近邻值的时候，每一行都是一个单位，这些行都将会作为一个单位出现在比较距离的程序之中。完成了这个步骤，训练集的数据才真正对我们的后续计算有意义。

当前数据分类

笔者在使用Knn对当前的一个句子进行情感分类的时候，按照要求，只依照一个条件——距离。所以笔者在后续的近邻值选择中，也就将问题简化为只有两个参量的问题：标签和距离。此二者的对应关系并非一一对应——这就意味着尽管我们需要其二者的映射关系，但是我们不能使用Map这种数据结构——在后续的比较中，又一定要遍历所有元组的标签和距离，从中选择距离最近的几个作为近邻值。为此，笔者为这两个参量的对应关系新建了一个结构体：

```
struct Candylabel{// candidate label-备选标签
    Candylabel(double dst, string lb){
        distance = dst;
        label = lb;
    }
    double distance;// 该标签值对应的距离
    string label; // 训练队列中的一个标签值
};
```

在遍历所有的Candylabel的时候，获得当前行的标签是非常容易的——标签正是笔者定义的一个成员变量，但是距离的运算就更为复杂了。这种情况下，笔者的选择是：

- 创建备选标签（标签映射距离）类的集合以备遍历
- 遍历一遍所有的训练集数据，以更新备选标签集合中元素的数据

```
vector<Candylabel> labelbase; // 创建备选标签的集合
// 遍历训练数据集，更新备选标签数据的内容
for(int i=0; i<Tuplebase.size(); i++){
  double crt_dst = tp.Distanceof(Tuplebase[i]);
  string crt_label = Tuplebase[i].label;
  Candylabel cl(crt_dst, crt_label);
  labelbase.push_back(cl);
}
```

近邻值的选取

随着备选标签集合数据的逐步更新，备选标签集合也就逐渐变得可用。笔者也就可用从备选标签集合中选择最靠前的几个近邻值。

```
sort(labelbase.begin(), labelbase.end(), labelCmp);
// 按照定义排序，排序之后，近邻值就显现出来了
```

获得近邻值集合之后，我们只需要：

- 对近邻值集合进行遍历；
- 找到其中出现次数最多的标签作为预测标签；

```
#ifdef Most
map<string, int> labelnum;// 对于某种标签的计数；
int max_num = 0;
string best_label;
for(int i=0; i<k; i++){// 遍历近邻值集合；
  labelnum[labelbase[i].label] += 1;
  int cnt = labelnum[labelbase[i].label];
  if(cnt >= max_num){// 更新近邻值集合众数；
    max_num = cnt;
    best_label = labelbase[i].label;
  }
}// CatchIT
//  cout << "label:" << best_label << endl;
```

之后的工作就是对于一个validation文件中的所有数据采取同样的操作方式，就可以得到预测集合的信息。对预测信息的数据集与validation原有的数据集进行比较，就可以获知准确率了。

```
double Ratio(vector<string> va, vector<string> vb){
  // 两个vector分别表示预测集合与validation集合的情感标签信息
    double size = va.size(), cnt = 0;
    for(int i=0; i<size; i++){
      // 计数相同元素
        if(va[i] == vb[i])  cnt++;
    }
    printf("cnt=%lf\n", cnt);
    printf("size=%lf\n", size);
    return ((double)cnt/size);// 比例输出
}
```

创新点与优化

使用对数数学模型，对近邻值的数据进行拟合，以求得到情感信息的判断优化。

笔者使用对数模型，对近邻值中的每一个数据都给予了一个为判断加权的机会。因为在近邻值的选取中，可能会出现以下情况：

- 近邻值大部分的情感是与测试数据不符的
- 少数情感文本中有一个与当前测试数据极为符合

按照众数决策，这时候一定会选择虽然不符合，但是大多数的答案，而笔者尝试对数模型去努力避开这一点。

```
//  calculating the posibilities;
// 根据近邻值使用对数数学模型，对不同的情感的权值进行计算;
    map<string, double> labelwgt;
    double sum_distance=0, sum_wgt=0;
    for(int i=0; i<n; i++)  sum_distance += labelbase[i].distance;
    for(int i=0; i<n; i++){
        labelwgt[labelbase[i].label] += abs(log( sum_distance/(labelbase[i].distance)) );
        sum_wgt += abs(log( sum_distance/(labelbase[i].distance)) );
    }
//  之后是对最高权值的情感进行输出
```

三、实验结果与分析

1. 实验结果示例展示

```
Analizing train set data...
[Tuple]:Cleaning head marks.
[Tuple]:Information extracting done.
[Tuple]:Initialized tuples done.
Analyze train set done.
[main]:reading validation set data...
[main]:reading finished.
[Tuple]:Classifying 311 sentences...
[Tuple]:Classifying tuples done.
[output]:writing into a file...
[output]:writing finished.
[assess]assessing...
[assess]assessing done.


Consequence:
cnt=96.000000
size=311.000000
30.8682%

--------------------------------
Process exited after 5.688 seconds with return value 0
请按任意键继续. . .
```

2. 评测指标展示及分析

当前的计算结果是基于：

- 欧拉距离
- 众数决策
- 近邻值为13的情况下得到的准确率。

这显然不是最大的准确度，笔者之前在调试的时候，最高的准确度达到40.836%，但是忘记了数据，由于时间紧迫，故没有将已经计算得出的数据展示出来，见谅。

关于准确度的提升

准确度提升可以从几个方面入手：

- 距离的计算方式

  除了使用欧式距离之外，还可以使用曼哈顿距离，对不同的元组之间进行计算。目前测试的情况，曼哈顿距离的均值要更高一些——而且准确度也更加稳定，虽然都不能超过40%，但是从5到30这个近邻区间，准确度也都可以到达35%到39%之间。

  总之，曼哈顿距离：稳定

  而对于欧式距离来说，尽管在笔者的测试中，欧式距离可以达到40.836%的准确度，但是在近邻值小于10的时候，准确度会急剧下降。

测评指标展示：

- 附上记录表仅供参考：

```
k=30    37.5&
k=25    37.1795%
k=19    36.5385%
k=17    39.1026%
k=16    38.4615%
k=15    39.1026%
k=14    38.4615%
k=13    40.836%++
k=12    39.1026%
k=11    39.1026%
k=9     33.9744%
k=7     34.2949%
```

- 近邻值的数据处理

  在获取近邻值之后，按照要求应该是对近邻值数据进行众数决策，选择近邻值数据中出现测试最多的标签作为我们的预计标签。

  这种方式：

  - 实现了对于相似数据的符合。与当前测试文本相似的数据都可以对我们的预测起到很好的指导作用。
  - 在陌生的数据出现时，预测效果差。我们获知的数据很多都是与训练集数据没有交集的——所谓没有交集，就是没有任何一个词是重复的。这种情况下，对近邻值的选择更多的都是文本出现的顺序，以及与当前测试数据的词数更为相近的数据成为了近邻值——很显然这并不是一个合适的筛选方案。

  所以笔者在根据已有的基础上，实现了另一种计算方式，对近邻值中的数据，都给予加权的机会，使得这些数据都会为我们的决策产生影响——目前测试效果来看，该分类方法比较稳定，但是准确度还是没有超过40%。

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不都为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探测到，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，但是文本的内部有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。