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

（2017-2018 学年秋季学期）

课程名称：**Artificial Intelligence**

教学班级	1501 班	专业（方向）	互联网
学号	15352010	姓名	蔡烨

一、 实验题目

K 近邻与朴素贝叶斯——分类和回归

二、 实验内容

1. 算法原理

KNN：

在 K 近邻算法中，利用训练集，找出测试样本与每个训练集样本的距离（距离度量有多种方式：曼哈顿距离、欧式距离、余弦值等），找出距离最近的 K 个训练集样本，将这 K 个样本中出现次数最多的情感标签赋给测试样本，即是 K 近邻算法。实际上就是用训练集把特征空间划分为一个个子空间，当测试样本落在某个训练样本的区域内，便把测试样本标记为这一类。

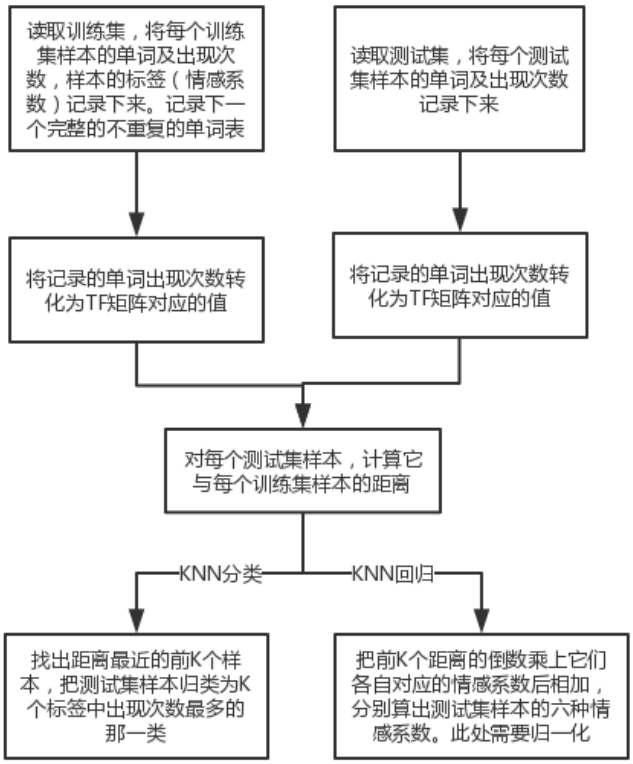
NB：

朴素贝叶斯的前提是假设各个属性相互独立，即忽略了各个属性的联系。然后利用贝叶斯算法，求解出给定的测试样本各个类别出现的概率，分类的情况下则把测试样本标记为概率最大的那个类别。贝叶斯的公式如下，而 NB 则是要找出使得 p(x|y)p(y)最大的 y 值。

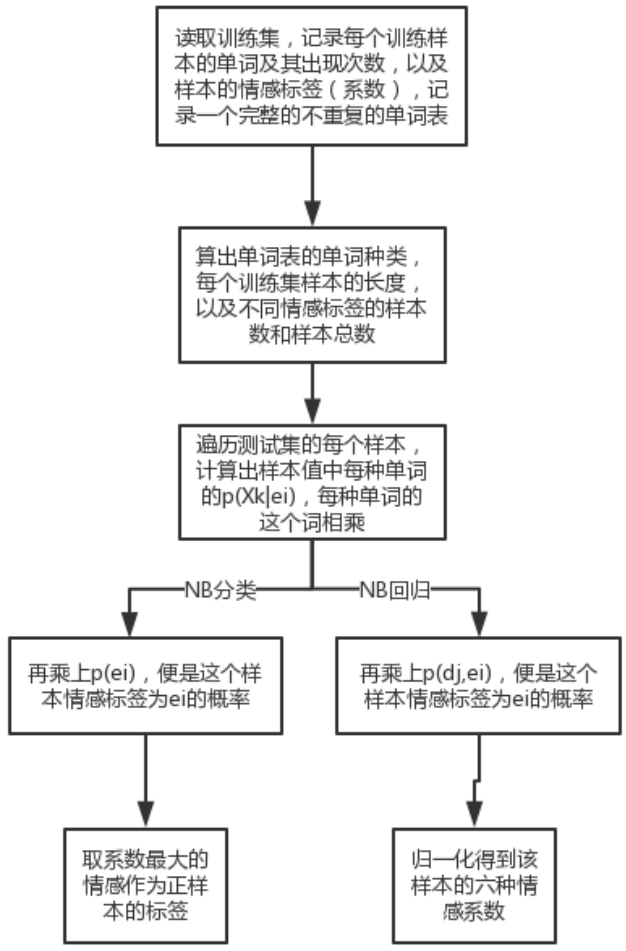
$$p(y|x) = \frac{p(x,y)}{p(x)} = \frac{p(x|y)p(y)}{p(x)}$$

2. 伪代码

KNN：


读取训练集，将每个训练集样本的单词及出现次数，样本的标签（情感系数）记录下来。记录下一个完整的不重复的单词表
读取测试集，将每个测试集样本的单词及出现次数记录下来
将记录的单词出现次数转化为TF矩阵对应的值
将记录的单词出现次数转化为TF矩阵对应的值
对每个测试集样本，计算它与每个训练集样本的距离
KNN分类
KNN回归
找出距离最近的前K个样本，把测试集样本归类为K个标签中出现次数最多的那一类
把前K个距离的倒数乘上它们各自对应的情感系数后相加，分别算出测试集样本的六种情感系数。此处需要归一化

NB：


读取训练集，记录每个训练样本的单词及其出现次数，以及样本的情感标签（系数），记录一个完整的不重复的单词表
算出单词表的单词种类，每个训练集样本的长度，以及不同情感标签的样本数和样本总数
遍历测试集的每个样本，计算出样本值中每种单词的p(Xk|ei)，每种单词的这个词相乘
NB分类
NB回归
再乘上p(ei)，便是这个样本情感标签为ei的概率
再乘上p(dj,ei)，便是这个样本情感标签为ei的概率
取系数最大的情感作为正样本的标签
归一化得到该样本的六种情感系数

3. 关键代码截图（带注释）

KNN 分类：

```
map<string,int> words;//词汇表及单词的出现顺序
vector< map<string,double> > lines;//训练集各个样例的单词及其TF矩阵对应的值
vector<string> label;//训练集每个样例的情感标签
vector< map<string,double> > test;//测试集的单词及其对应的TF矩阵的值
vector<string> validation;//测试集的标签
vector<string> predict;//预测结果
fstream f1;
```

第一次读取训练集样本时，先记录下每个单词在样本中出现的次数，之后再统一处理把次数换成 TF 矩阵对应的值。

读取单词时的判断：

```
if(!words.count(s)) {
    words[s] = words.size();
    wordsOfLine[s] = 1;
}
else
    wordsOfLine[s] += 1;
```

计算训练集的 TF 矩阵：

```
//计算训练集的TF矩阵
for(int i=0; i<lines.size(); i++){
    double num = 0;//单词总个数
    for(map<string,double>::iterator it=lines[i].begin(); it!=lines[i].end(); it++)
        num += it->second;
    for(map<string,double>::iterator it=lines[i].begin(); it!=lines[i].end(); it++)
        it->second = it->second / num;
}
```

计算测试集样本的 TF 矩阵：

```
int sizeInWords = 0; //第row行测试样例的单词数
for(map<string,double>::iterator it=test[row].begin(); it!=test[row].end(); it++)
    if(words.count(it->first)) //如果出现不在单词表的单词，不参与计数
        sizeInWords += it->second;
for(map<string,double>::iterator it=test[row].begin(); it!=test[row].end(); it++){
    if(words.count(it->first))
        it->second = it->second / sizeInWords; //计算其对应的TF矩阵的值
}
```

基于 TF 矩阵计算两个样本向量的距离

```
for(map<string,double>::iterator it=lines[i].begin(); it!=lines[i].end(); it++){
    if(!test[row].count(it->first))//不在测试样例中
        dis += pow(it->second,2); //欧式距离
        dis += it->second; //曼哈顿距离
    else
        dis += pow((it->second - test[row][it->first]),2);
        dis += abs(it->second - test[row][it->first]);
}
for(map<string,double>::iterator it=test[row].begin(); it!=test[row].end(); it++)
    if(!lines[i].count(it->first))
        if(words.count(it->first))
            dis += pow(it->second,2);
            dis += it->second;

dis = sqrt(dis); //欧氏距离
distance.insert(pair<double,int>(dis,i));
```

先遍历测试样本的单词
如果出现不在单词表的单词，不作处理

再遍历训练样本中的单词，找到那些没出现在测试样本中的单词，因为它们会影响两者之间的距离

找到前 K 个距离对应的样本，标签每出现一次，其对应的 motion 的键值对的值便多加一。这样键值对中值最大的那个便是最近的 K 个样本里出现次数最多的标签。

```
map<string,int> motion;
motion["joy"] = 0;
motion["fear"] = 0;
motion["sad"] = 0;
motion["anger"] = 0;
motion["surprise"] = 0;
motion["disgust"] = 0;

//找到前k个数的众数，得到其label
multimap<double,int>::iterator iter=distance.begin();
for(int num=0; num<9; num++){ //k的取值
    motion[label[(iter++)->second]] += 1;
}
```

如果出现次数最多的不止一个标签，那么找出前 K 个最近样本中那些属于这些标签的距离最近的一个样本，把测试样本归类为跟它一样的标签。

```
bool flag = 0;
for(map<string,int>::iterator it=motion.begin(); it!=motion.end(); it++){
    if(it->second > max){
        max = it->second;
        pred = it->first;
        flag = 0;
    }
    else if(it->second == max)
        flag = 1;
}
if(flag) {//不存在唯一的最大值时，选用距离最近的label
    //取属于众数的情况中距离最近的那个label
    for(multimap<double,int>::iterator it=distance.begin(); it!=distance.end(); it++){
        if(motion[pred] == motion[label[it->second]]){
            predict.push_back(label[it->second]);
            break;
        }
    }
    predict.push_back(label[distance.begin()->second]);//取最近距离的label
}
```

如果这个最大值是第一次出现，flag为0

如果这个最大值前面有跟它一样大的，flag为1

KNN 回归：

KNN 回归与 KNN 分类，一直到计算 TF 矩阵的值，代码都是一样的，不同的是计算情感系数的方法。

```
map<string,int> words;//词汇表及单词的出现顺序
vector< map<string,double> > lines;//训练集各个样例的单词及其对应的tf矩阵的值
vector< vector<double> > label;//训练集每个样例的情感标签
vector< map<string,double> > test;//测试集的单词及其对应的tf矩阵的值
vector< vector<double> > predict;//预测结果
fstream f1;
```

首先判断最近的距离是否 0，如果是则意味着训练集里有某个样本和这个测试样本出现了单词种类和次数是一样的。因为在这个实验中，单词出现的顺序并不是影响因素，因此可以认为这是两个一模一样的样本，所以直接将训练集样本的情感系数赋给该测试集样本。

```
vector<double> motion;
for(int i=0; i<6; i++){ //六种情绪
    multimap<double,int>::iterator iter = distance.begin();
    double res = 0.0;
    if(iter->first == 0) { //最近的距离为0，即存在一模一样的样例
        for(int k=0; k<6; k++){
            motion.push_back(label[iter->second][k]);
        }
        break;
    }
    else{
        for(int j=0; j<5; j++){ //K的取值
            res += label[iter->second][i] / (1.0*iter->first);
            iter++;
        }
    }
    motion.push_back(res);
    res = 0.0;
}
```

而如果最近距离不是 0，那就找出最近的 K 个距离，将它们取倒数乘上对应的样本的情感系数，然后相加。将得到的六种情感系数归一化（不是严格意义上的归一化，只是使得六种情感系数的和为 1）后则是最后的结果。

```
//归一化情感系数
double sum = 0.0;
for(int i=0; i<6; i++)
    sum += motion[i];
for(int i=0; i<6; i++)
    motion[i] = motion[i] / sum;
predict.push_back(motion);
```

NB 分类：

读取文件和计算单词数都同 KNN 算法。

```
map<int,string> motion;
map<string,int> words;//单词表
vector< map<string,int> > label[6];//不同标签的每一行的词及其出现次数
int total_words_label[7] = {0}; //不同标签中总共出现多少次单词（不去重）
int total_words_norepeat = 0;//训练集的单词表总数
int total_doc_label[7] = {0};//文章总数，不同标签的文章数
vector<string> result;//存放预测结果
vector<string> validation;//验证集的标签
fstream f1,f2;
```

而预测情感标签时，是计算出每种情感的系数，取最大的那个作为标签。
使用的公式如下；

$$p(e_i) = \frac{N_{e_i}}{N} \qquad p(x_k|e_i) = \frac{nw_{e_i}(x_k) + 1}{nw_{e_i} + V_{e_i}}$$ （拉普拉斯平滑）

```
double predict[6];//六种情绪的预测值
for(int i=0; i<6; i++)
    predict[i] = 1;
for(int i=0; i<6; i++){//六种情绪
    vector<int> timeOfWord;//nWi(xk) 单词在情感标签为i的训练样本中出现的次数
    for(map<string,int>::iterator it=test.begin(); it!=test.end(); it++){
        int cnt = 0;
        for(int j=0; j<label[i].size(); j++){
            if(label[i][j].count(it->first)){
                cnt += label[i][j][it->first];
            }
        }
        timeOfWord.push_back(cnt);
    }

    for(int j=0; j<timeOfWord.size(); j++) //各个p(xk/ei) = (1+nWi(xk)) / (nWi+V)相乘
        predict[i] *= (1.0*timeOfWord[j]+1)/(1.0*total_words_label[i]+1.0*words.size());
    // P(ei) = Ni/N
    predict[i] *= (1.0*total_doc_label[i])/(1.0*total_doc_label[6]);
```

NB 回归：

```
vector< map<string,int> > lines;
vector<int> length;//每一行的长度
map<string,int> wordlist;
vector< vector<double> > label;//行：某个文章的六种情绪
vector< vector<double> > predict;//预测值
fstream f1,f2;
```

读取步骤依然同上，差别在于计算情感标签，使用的公式如下：

$$arg\max_{e_i} p(e_i|X) = arg\max_{e_i} \sum_{j=1}^{M}\prod_{k=1}^{K'} p(x_k|e_i,d_j)p(d_j,e_i) \qquad \mathbf{p}(\boldsymbol{X_k}|\boldsymbol{d_j}) = \frac{(\boldsymbol{X_k}+\mathbf{0.01})}{\sum_{k=1}^{K}\boldsymbol{X_k}+\mathbf{0.01}\boldsymbol{k}}$$

其中 k 为整个单词表的单词种类

```
vector<double> motion;
for(int i=0; i<6; i++){//六种情绪
    double motion_label = 0;
    for(int j=0; j<lines.size(); j++){//每一个训练集的样例
        double motion_line = 1;
        for(map<string,int>::iterator it=wordsOfLine.begin(); it!=wordsOfLine.end(); it++){
            double temp = 0;//Xk，单词在样本中出现的次数
            if(lines[j].count(it->first))
                temp += lines[j][it->first];
            temp += 0.01;
            //p(Xk|dj,ei) = (Xk+0.01) / (sum(Xk)+0.01*K)
            motion_line *= temp / (1.0*length[j]+0.01*wordlist.size());
        }
        motion_line *= label[j][i];
        motion_label += motion_line;
    }
    motion.push_back(motion_label);
}
```

4. 创新点&优化（如果有）

用 map 这种数据结构的键值对方法，读取单词的同时记录下它出现的次数。后来将次数转换为 TF 矩阵对应的值，可以看成是一种稀疏矩阵的存储方式。因为这样就不用存储整个 TF 矩阵，只需要将有值的地方记录即可。而显然，每个样本中出现的单词，都在 TF 矩阵中有对应的值，没有出现的单词，则值为 0。

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）

采用跑小数据集的方式验证：

分类：

train_set:

```
Words,label
hello introduce this is test,joy
hello this is a test,joy
no girlfriend,sad
this my girlfriend,joy
hello you have no grilfriend,sad
```

test_set:

```
Words,label
hello this is my girlfriend,?
```

KNN 分类：

```
textid,label
1,joy
```

KNN 分类：

```
textid,joy,sad,anger,fear,disgust,surprise,totallabel
1,0.9319,0.0681,0.0000,0.0000,0.0000,0.0000,joy
```

回归：

train_small:

```
Words (split by space),anger,disgust,fear,joy,sad,surprise
hello introduce this is test,0.0,0.0,0.0,0.8721,0.0,0.1279
hello this is a test,0.1625,0.0,0.225,0.0,0.4375,0.175
no girlfriend,0.1056,0.169,0.5282,0.0,0.1549,0.0423
this my girlfriend,0,0,0,0.5733,0,0.4267
hello you have no girlfriend,0.0,0.0,0.0,0.7358,0.0,0.2642
```

test_small:

```
Words,label
hello this is my girlfriend,?
```

KNN 回归：

```
id,anger,disgust,fear,joy,sad,surprise
1,0.041539,0.000000,0.057516,0.537047,0.111837,0.252060
```

NB 回归：

```
textid,anger,disgust,fear,joy,sad,surprise
1,0.011674,0.000099,0.016387,0.553465,0.031354,0.387022
```

2. 评测指标展示即分析（如果实验题目有特殊要求，否则使用准确率）

KNN 分类：

其中 k=9 时，验证集准确率最高为 0.4469. 其他方式结果如下：

矩阵类型	距离度量方式	准确率最高时的 K 取值	最高准确率
TF	曼哈顿距离	9	0.4469
TF	欧式距离	25	0.3569
One-hot	曼哈顿/欧式	13	0.4244

KNN 回归：

矩阵类型	距离度量方式	相关性最高时的 K 取值	最高相关性
TF	曼哈顿距离	5	0.3438
TF	欧式距离	4	0.2802
One-hot	曼哈顿/欧式	4	0.2985

NB 分类：

采用拉普拉斯平衡，验证集准确率为 0.4405

NB 回归：

采用拉普拉斯平滑，验证集相关性为 0.3589

3.思考题

（1）KNN 回归中根据相似度加权时，把距离的倒数作为权重。为什么要用倒数？

答：因为距离越近，其影响的程度应该越高，相应的权重应该更大，所以应该用倒数。这样才会使得距离小的权重大。

（2）同一测试样本的各个情感概率总和应该为 1，如何处理？

答：同一测试样本的情感概率算出来之后，分别去除以六个概率的总和，得到的才是最后各个情感概率，这样他们的总和为 1。

（3）在稀疏程度不同的时候，曼哈顿距离和欧式距离表现有什么区别，为什么？
答：当矩阵不稀疏时，曼哈顿距离和欧式距离所判定的结果可能不同。当矩阵稀疏的时候，曼哈顿距离实际上能代表欧式距离，不容易导致判断不同。
曼哈顿距离增长大于欧式距离，所以当矩阵不稀疏的时候，曼哈顿距离和欧式距离的差别会比较明显，采用两种度量方式判定的结果可能会比较矛盾。

（4）伯努利模型和多项式模型有什么优缺点？
答：伯努利模型以句子为基准，方便计算，但是只有存在和不存在即 1 和 0 两个特征值，直接放大或消除了单词对整个句子的影响，容易使得计算某个词代表的某个情感概率时出现极端化结果。
多项式模型是以单词为基准，避免了二值分化，更能合理地体现单词所代表的情感概率，使得某种情感下某个单词所带来的概率影响比较合理，但同时也增加了计算复杂度。

（5）如果测试集中出现了一个之前全词典中没有出现过的词，该如何解决？
答：KNN 中，因为采用的是 TF 矩阵，当出现不在单词表里的单词时，不计算它对样本长度的影响，也不算它对 TF 的影响。也可以看做去掉这个单词后再进行其他操作。NB 中因为加入了拉普拉斯平滑，所以即使不存在单词表中，该单词也能获得一定的情感系数。

:

|---------- 如有优化，重复 1，2 步的展示，分析优化后结果 ----------|

PS：可以自己设计报告模板，但是内容必须包括上述的几个部分，不需要写实验感想