



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	1515	专业(方向)	软件工程
学号	15352334	姓名	吴佳卫

一、实验题目

k 近邻与朴素贝叶斯

二、实验内容

1. 算法原理

1) K-NN 分类

k-NN 分类算法会根据测试集从训练集中挑选出最接近(距离最小)的样本,根据 k 值的大小采取投票,前 k 个样本中数量最多的就是我们想要的结果。

2) K-NN 回归

K-NN 回归算法的基本原理与 K-NN 算法的相同,只不过我们在这里利用前 k 个样本计算出测试样本的属性值。计算过程可以是取平均值也可以根据距离加权。

3) 朴素贝叶斯分类

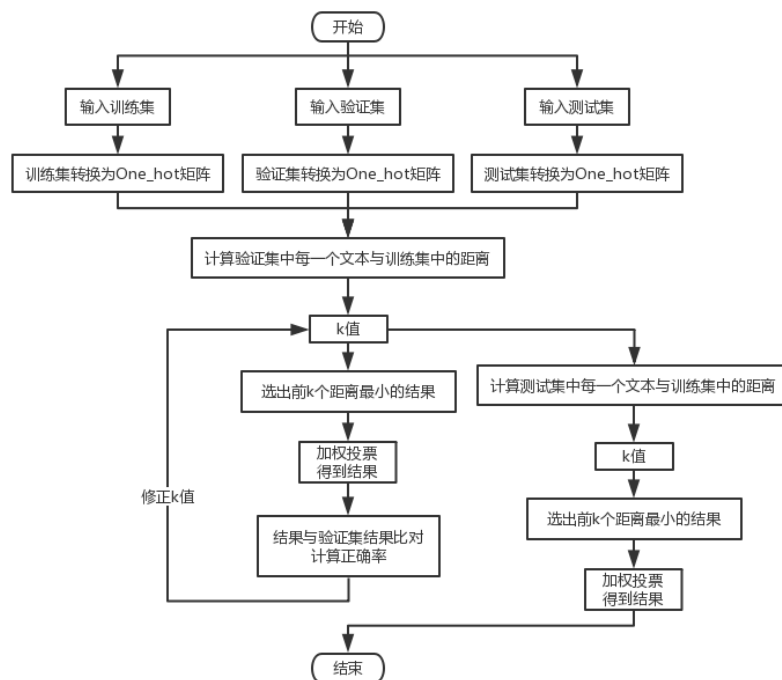
朴素贝叶斯算法的基础就是贝叶斯定理,根据贝叶斯定理我们可以得到两个事件交换之后的概率。朴素贝叶斯分类的原理是根据某一个项对于一个类别出现的概率来进行分类。比如说“joy”情绪中“happy”这个词出现的概率很大,那么我们也可以认为出现“happy”这个词意味着这个文本是“joy”类别的概率很大。

4) 朴素贝叶斯回归

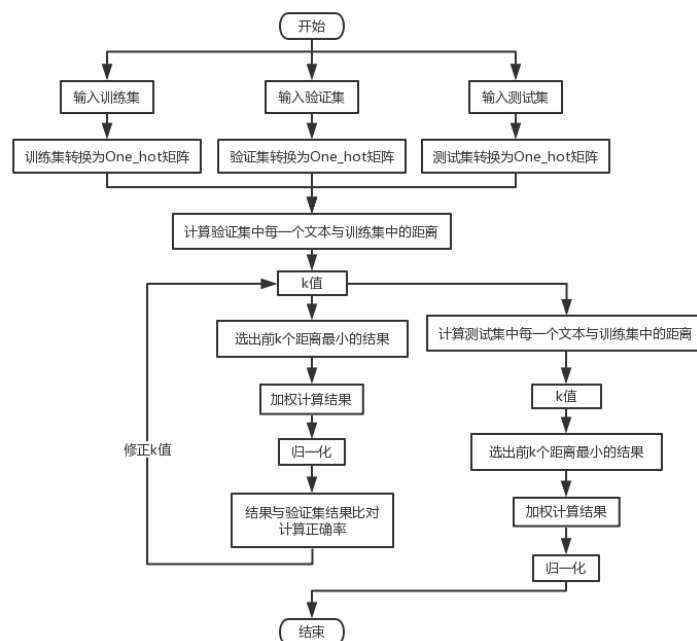
基本原理也是利用贝叶斯定理,与朴素贝叶斯分类不同的就是朴素贝叶斯回归计算的结果是这个样本的一些属性值,其余部分可以看做基本相同。

2. 伪代码

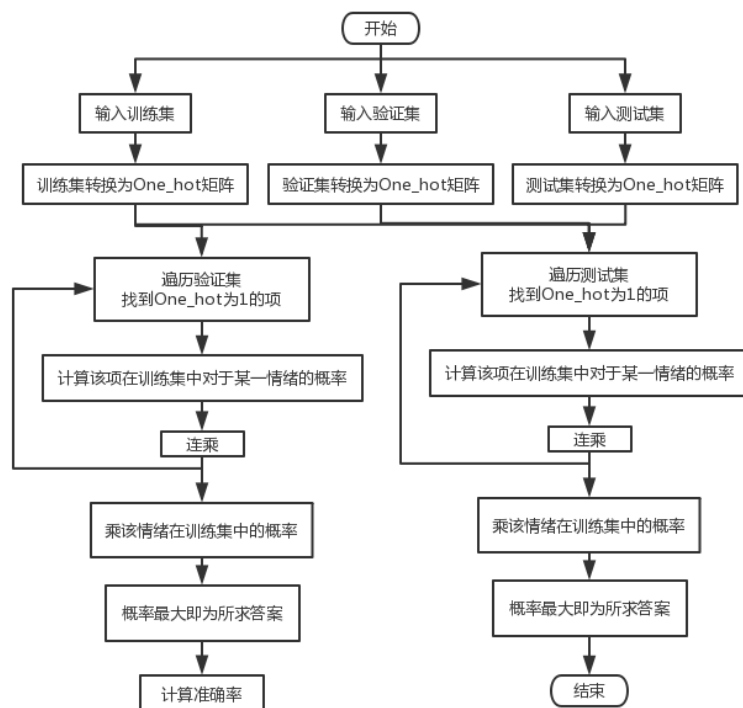
1) K-NN 分类



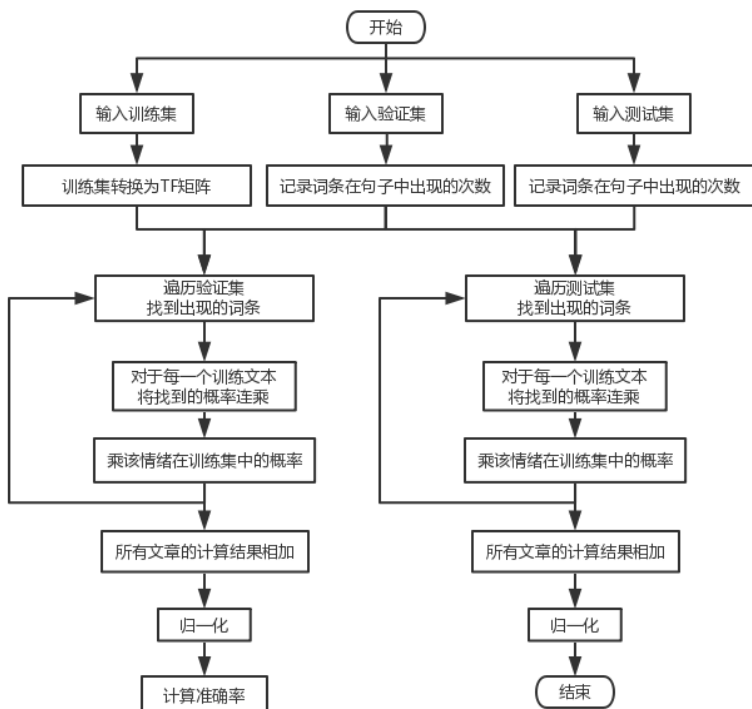
2) K-NN 回归



3) 朴素贝叶斯分类



4) 朴素贝叶斯回归



3. 关键代码截图

1) K-NN 分类

```
while(getline(fin,buffer)) //逐篇文章读入
{
    int p1=0;int p2=0;string tmp;
    while(p2<buffer.length())
    {
        if(buffer[p2]!=' '||buffer[p2]!='.') //根据空格或都逗号切词
        {
            tmp=buffer.substr(p1,p2-p1);
            vector<string>::iterator s=find(Word_list.begin(),Word_list.end(),tmp);
            if(s!=Word_list.end()) //该词汇是否已经出现过
            {
                One_hot[passage][s-Word_list.begin()]=1;
            }
            else
            {
                Word_list.push_back(tmp);
                One_hot[passage][Word_list.size()-1]=1;
            }
            p1=p2+1;
            p2=p1+1;
        }
        else
        {
            p2++;
        }
    }
    emotion_list.push_back(buffer.substr(p1,buffer.length()-p1)); //存入情感
    passage++;
}
```

读入训练集，存储入 One_hot 矩阵，基本算法与实验一相同；

```
double cal_dis(int train_case,int validation_case) //计算两个文本之间的距离
{
    double sum=0;
    for(int i=0;i<Word_list.size();i++)
    {
        double tmp_dis=One_hot[train_case][i]-validation_One_hot[validation_case][i];
        sum+=tmp_dis*tmp_dis;
    }
    sum=sqrt(sum);
}
```

基于 One_hot 矩阵计算两个文本距离的函数；

```
struct node
{
    string emotion; //训练集对应的情感
    double distance; //文本与训练集的距离
};

bool comp(const node &a,const node &b)
{
    return a.distance<b.distance;
}
```

为了存储方便新建的结构体，并且定义了比较函数，方便下文进行排序；



```
int emotion_id(string e)
{
    if(e=="anger") return 0;
    else if(e=="disgust") return 1;
    else if(e=="fear") return 2;
    else if(e=="joy") return 3;
    else if(e=="sad") return 4;
    else if(e=="surprise") return 5;
    else cout<<"EMOTION ERROR"<<endl;
}

string emotion_name(int i)
{
    if(i==0) return "anger";
    else if(i==1) return "disgust";
    else if(i==2) return "fear";
    else if(i==3) return "joy";
    else if(i==4) return "sad";
    else if(i==5) return "surprise";
}
```

为了方便存储代表情绪的字符串设计了 id 的转换规则；

```
vector<node> rank; //用于存储每一项
for(int i=0;i<passage;i++)
{
    node tmp;
    if(mod==0) //计算验证集
    {
        tmp.distance=cal_dis(i, validation_case);
    }
    else if(mod==1) //计算测试集
    {
        tmp.distance=cal_dis2(i, validation_case);
    }
    tmp.emotion=emotion_list[i];
    rank.push_back(tmp);
}
sort(rank.begin(), rank.end(), comp); //根据距离排序
int max_emotion;
if(rank[0].distance==0) //如果有距离为零直接选择
{
    max_emotion=emotion_id(rank[0].emotion);
}
else
{
    double vote[8];
    memset(vote, 0, sizeof(vote));
    for(int i=0;i<k;i++) //加权投票，用(1/距离)作为权重
    {
        vote[emotion_id(rank[i].emotion)] += 1/rank[i].distance;
    }
    double max=0;
    for(int i=0;i<6;i++) //找到最大项
    {
        if(vote[i]>max)
        {
            max=vote[i];
            max_emotion=i;
        }
    }
}
```

进行分类算法的核心；



```
int best_k=1;
double best_answer=0;
for(int k=10;k<sqrt(passage+1)+10;k++) //k的取值范围是10到根号下文章数加10
{
    emotion_list_result.clear();
    cout<<'running on k='<<k<<' ';
    double local_answer=0;
    for(int i=0;i<validation;i++)
    {
        emotion_list_result.push_back(emotion_name(classification(k,i,0)));
        if(emotion_list_result[i]==emotion_list_2[i]) local_answer++;
    }
    local_answer=local_answer/validation;
    cout<<local_answer<<endl;
    if(local_answer>best_answer) //取到最佳的k
    {
        best_answer=local_answer;
        best_k=k;
    }
}
cout<<'best_k='<<best_k<<endl;
```

在外部加一个循环，找到最优的 k，用于对测试集的计算；

```
void Print_result()
{
    ofstream fout('DATA\\classification_dataset\\15352334_wujiawei_KNN_classification.csv');
    fout<<'textid'<<','<<'label'<<endl;
    for(int i=0;i<test;i++)
    {
        fout<<i+1<<','<<test_emotion_list[i]<<endl;
    }
}
```

打印测试集的函数；

2) K-NN 回归

```
int mod=0;
if(buffer[p2]==' '||buffer[p2]==',')
{
    if(buffer[p2]==' ') mod=1;
    else if(buffer[p2]==',' && flag==true)
    {
        flag=false;
        mod=1; //输入文本
    }
    else if(buffer[p2]==',' && flag==false) mod=2; //输入情绪
}
else
{
    mod=0; //继续遍历
    p2++;
}
```

由于回归的文档和分类的是不一样的（回归的训练集后面有 6 个数字表示每一种情绪的概率），因此在输入的同时做判断，再决定下一个输入项的存储位置；



```

if ( mod == 1 )
{
    tmp=buffer.substr( p1, p2-p1);
    vector<string>::iterator s=find( Word_list.begin(), Word_list.end(), tmp);

    if ( s != Word_list.end() )
    {
        One_hot[ passage ][ s - Word_list.begin() ] = 1;
    }
    else
    {
        Word_list.push_back( tmp );
        One_hot[ passage ][ Word_list.size() - 1 ] = 1;
    }

    p1=p2+1;
    p2=p1+1;
}
else if ( mod == 2 )
{
    tmp=buffer.substr( p1, p2-p1);
    double tmp_num=stod( tmp);
    emotion_list[ passage ][ num_cnt ] = tmp_num;
    num_cnt++;
    p1=p2+1;
    p2=p1+1;
}
}
tmp=buffer.substr( p1, buffer.length()-p1);
emotion_list[ passage ][ num_cnt ] = stod( tmp);
passage++;

```

存储输入，与前面的原理是一样的，只不过文本存入文本向量，情感概率存入情感概率

向量；

```

double evaluate() //计算验证集计算结果与答案的相关系数
{
    double aver1=0;
    double aver2=0;
    double eva=0;
    double tmp1=0;
    double tmp2=0;
    double tmp3=0;
    for( int i=0; i<6; i++)
    {
        for( int j=0; j<validation; j++)
        {
            aver1+=emotion_list_result[j][i];
            aver2+=emotion_list_2[j][i];
        }
        aver1/=validation;
        aver2/=validation;
        tmp1=0;
        tmp2=0;
        tmp3=0;
        for( int j=0; j<validation; j++)
        {
            tmp1+=(emotion_list_result[j][i]-aver1)*(emotion_list_2[j][i]-aver2);
            tmp2+=(emotion_list_result[j][i]-aver1)*(emotion_list_result[j][i]-aver1);
            tmp3+=(emotion_list_2[j][i]-aver2)*(emotion_list_2[j][i]-aver2);
        }
        eva+=tmp1/(sqrt(tmp2)*sqrt(tmp3));
    }
    eva/=6;
    return eva;
}

```

添加了一个计算验证集计算结果与答案相关系数的函数，便于筛选出最佳的 k；



```
void regression(int k, int validation_case, int mod) // anger, disgust, fear, joy, sad, surprise
{
    vector<node> rank;
    for(int i=0; i<passage; i++)
    {
        node tmp;
        if(mod==0) tmp.distance=cal_dis(i, validation_case); //计算验证集
        else if(mod==1) tmp.distance=cal_dis2(i, validation_case); //计算测试集
        tmp.id=i;
        rank.push_back(tmp);
    }
    sort(rank.begin(), rank.end(), comp); //根据距离的远近进行排序
    double res=0;
    if(rank[0].distance==0) //如果有距离为零，直接赋值
    {
        for(int i=0; i<6; i++)
        {
            if(mod==0) emotion_list_result[validation_case][i]=emotion_list[rank[0].id][i];
            else if(mod==1) result[validation_case][i]=emotion_list[rank[0].id][i];
        }
    }
    else //前k项加权相加
    {
        for(int i=0; i<6; i++)
        {
            for(int j=0; j<k; j++)
            {
                res+=emotion_list[rank[j].id][i]/rank[j].distance;
            }
            if(mod==0) emotion_list_result[validation_case][i]=res;
            else if(mod==1) result[validation_case][i]=res;
        }
    }
}
```

回归函数，基本原理就是先排序后加权相加，基本框架与之前的相同；

```
int best_k=1;
double best_answer=0;
for(int k=1; k<=10; k++)
{
    memset(emotion_list_result, 0, sizeof(emotion_list_result));
    cout<<"running on k="<<k<<" ";
    double local_answer=0;
    for(int i=0; i<validation; i++)
    {
        regression(k, i, 0);
    }
    for(int i=0; i<validation; i++) //归一化
    {
        double sum_up=0;
        for(int j=0; j<6; j++)
        {
            sum_up+=emotion_list_result[i][j];
        }
        for(int j=0; j<6; j++)
        {
            emotion_list_result[i][j]/=sum_up;
        }
    }
    local_answer=evaluate(); //评估相关系数
    cout<<local_answer<<endl;
    if(local_answer>best_answer)
    {
        best_answer=local_answer;
        best_k=k;
    }
}
cout<<best_answer<<endl;
```

进行归一化与相关度计算，并通过一个循环找到对于验证集最佳的 k 值；



3) 朴素贝叶斯分类

```
double probability[8];
for(int i=0;i<8;i++) probability[i]=1;
for(int i=0;i<Word_list.size();i++)
{
    if(validation_One_hot[cases][i]==1)
    {
        for(int j=0;j<6;j++)
        {
            double tmp1,tmp2,tmp3;
            tmp1=0.01;
            for(int k=0;k<passage;k++)
            {
                if(One_hot[k][i]==1&&(emotion_id(emotion_list[k])==j))
                {
                    tmp1+=1;
                }
            }
            tmp2=emotion_word_num[j]+emotion_non_rep[j];
            tmp3=tmp1/tmp2;
            probability[j]*=tmp3;
        }
    }
}
```

主体部分（输入存储）与 k-NN 算法的大致相同。在回归的部分根据训练集进行连乘的计算，引入了拉普拉斯平滑；

```
for(int j=0;j<6;j++)
{
    double tmp=emotion_num[j]/passage;
    probability[j]*=tmp;
}
double best_probability=0;
double best_emotion=0;
for(int j=0;j<6;j++)
{
    if(probability[j]>best_probability)
    {
        best_probability=probability[j];
        best_emotion=j;
    }
}
return best_emotion;
```

根据贝叶斯公式计算最后的答案，最后对答案进行归一化；

```
double evaluate=0;
for(int i=0;i<validation;i++)
{
    validation_emotion_list2.push_back(emotion_name(classification(i,0)));
    if(validation_emotion_list[i]==validation_emotion_list2[i])
    {
        evaluate++;
    }
}
evaluate/=validation;
cout<<"evaluate "<<evaluate<<endl;
```

计算验证集答案与计算结果的相似度，进行算法评估。



4) 朴素贝叶斯回归

```
void regression(int cases, int mod) //anger, disgust, fear, joy, sad, surprise
{
    double final_emotion[10];
    for(int i=0; i<6; i++) final_emotion[i]=0;
    if(mod==0)
    {
        for(int i=0; i<passage; i++)
        {
            double tmp_emotion[10];
            for(int j=0; j<6; j++) tmp_emotion[j]=1;
            for(int j=0; j<Word_list.size(); j++)
            {
                for(int k=0; k<validation_TF[cases][j]; k++)
                {
                    for(int l=0; l<6; l++)
                    {
                        tmp_emotion[l]*=(1+TF[i][j])/(Word_list.size()+word_num[i]);
                    }
                }
            }
            for(int j=0; j<6; j++)
            {
                final_emotion[j]+=tmp_emotion[j]*emotion_list[cases][j];
            }
        }
        double tmp_sum=0;
        for(int i=0; i<6; i++)
        {
            tmp_sum+=final_emotion[i];
        }
        for(int i=0; i<6; i++)
        {
            validation_emotion_list2[cases][i]=final_emotion[i]/tmp_sum;
        }
    }
}
```

回归部分完成概率的相乘再相加，引入拉普拉斯平滑。其余部分与朴素贝叶斯分类相同；

4. 创新点&优化

1) K-NN 分类

主要运用了两项优化，第一项是在对前 k 项进行投票统计众数时引入了权重。即排名越往前的文本应该有更大的相似度。因此将（1/距离）作为权重进行加权的投票，排名越前，距离越小将拥有更大的权重。

第二项优化是让模型拥有自我评估的能力，引入了根据验证集计算准确度的函数，从而确定对于模型最优的 k 值，直接运用到测试集中。

2) K-NN 回归

首先写了一个计算相关系数的函数，用于评估模型的准确度，然后主要也使用了自我评估的优化，选取最优的 k 并计算测试集。

3) 朴素贝叶斯分类

引入拉普拉斯平滑，并且在模型中根据验证集计算准确率。

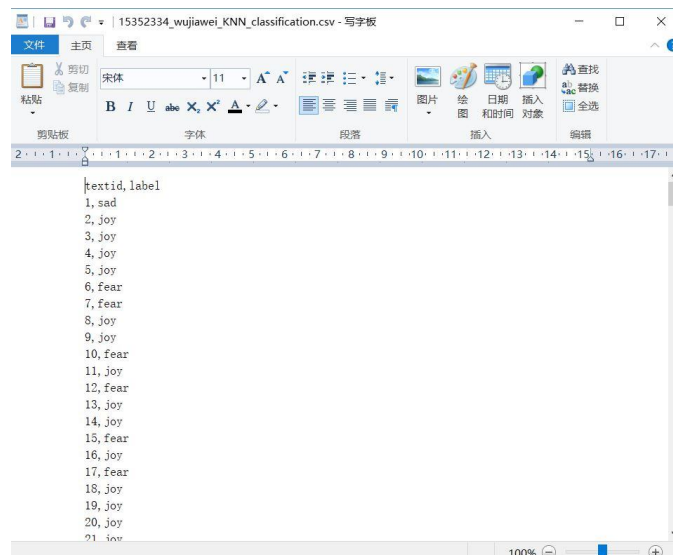
4) 朴素贝叶斯回归

引入拉普拉斯平滑,并且定义了计算相关系数的函数,在模型中根据验证集计算准确率。

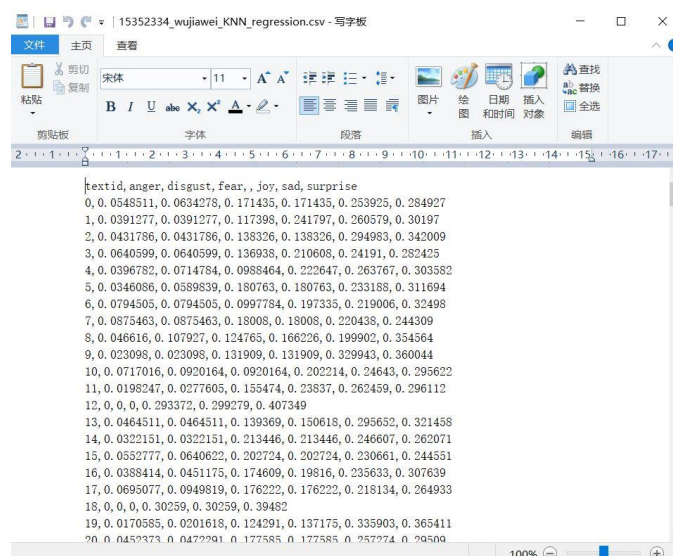
三、 实验结果及分析

1. 实验结果展示示例

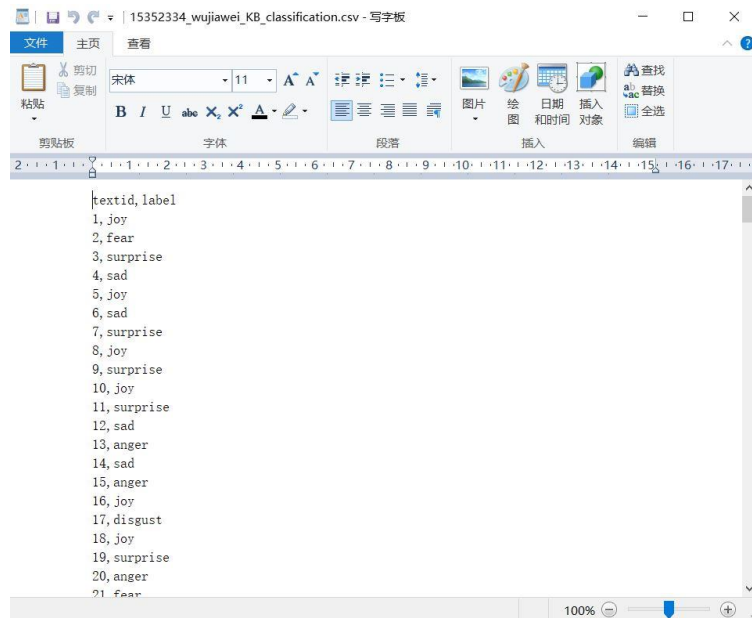
1) K-NN 分类



2) K-NN 回归

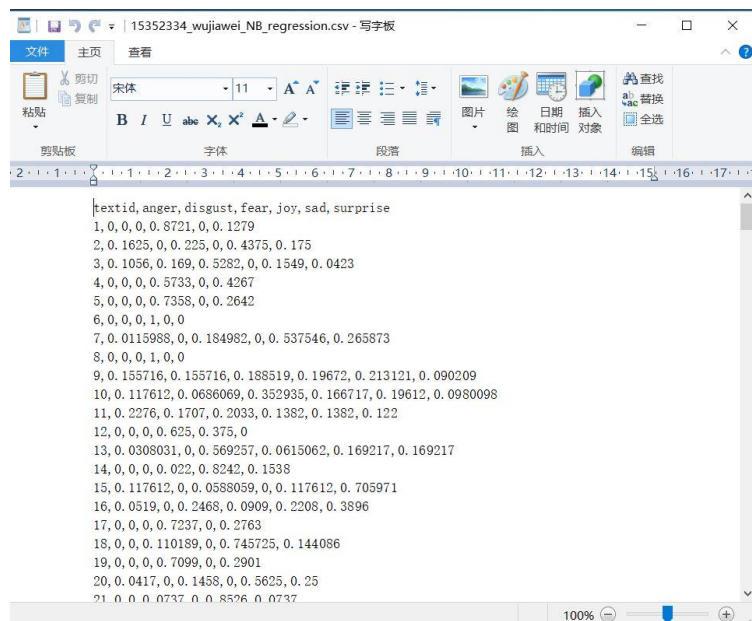


3) 朴素贝叶斯分类



textid, label
1, joy
2, fear
3, surprise
4, sad
5, joy
6, sad
7, surprise
8, joy
9, surprise
10, joy
11, surprise
12, sad
13, anger
14, sad
15, anger
16, joy
17, disgust
18, joy
19, surprise
20, anger
21, fear

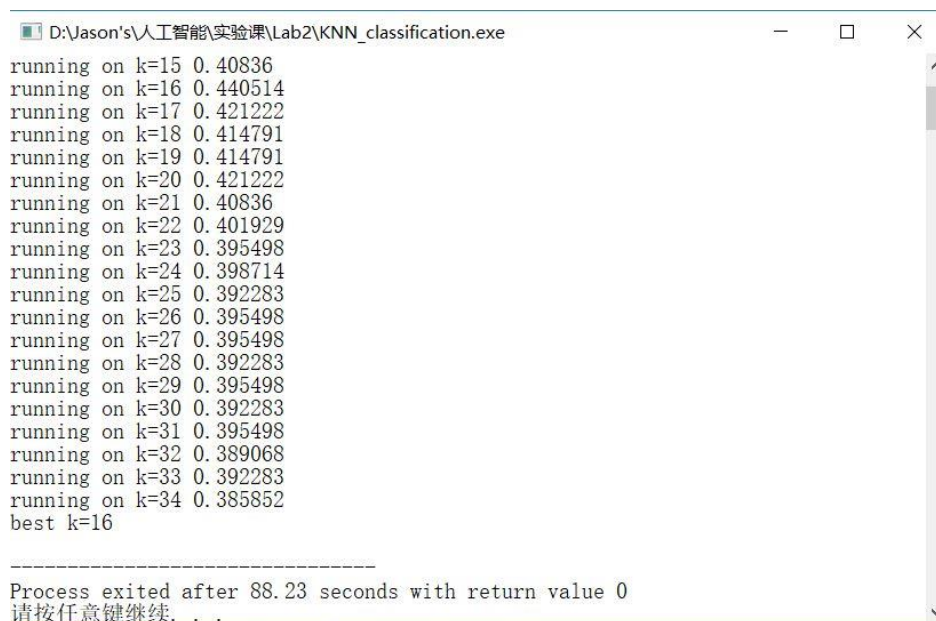
4) 朴素贝叶斯回归



textid, anger, disgust, fear, joy, sad, surprise
1, 0, 0, 0, 0.8721, 0, 0.1279
2, 0.1625, 0, 0.225, 0, 0.4375, 0.175
3, 0.1056, 0.169, 0.5282, 0, 0.1549, 0.0423
4, 0, 0, 0, 0.5733, 0, 0.4267
5, 0, 0, 0, 0.7358, 0, 0.2642
6, 0, 0, 0, 1, 0, 0
7, 0.0115988, 0, 0.184982, 0, 0.537546, 0.265873
8, 0, 0, 0, 1, 0, 0
9, 0.155716, 0.155716, 0.188519, 0.19672, 0.213121, 0.090209
10, 0.117612, 0.0686069, 0.352935, 0.166717, 0.19612, 0.0980098
11, 0.2276, 0.1707, 0.2033, 0.1382, 0.1382, 0.122
12, 0, 0, 0, 0.625, 0.375, 0
13, 0.0308031, 0, 0.569257, 0.0615062, 0.169217, 0.169217
14, 0, 0, 0, 0.022, 0.8242, 0.1538
15, 0.117612, 0, 0.0588059, 0, 0.117612, 0.705971
16, 0.0519, 0, 0.2468, 0.0909, 0.2208, 0.3896
17, 0, 0, 0, 0.7237, 0, 0.2763
18, 0, 0, 0.110189, 0, 0.745725, 0.144086
19, 0, 0, 0, 0.7099, 0, 0.2901
20, 0.0417, 0, 0.1458, 0, 0.5625, 0.25
21, 0, 0, 0.0737, 0, 0.8526, 0.0737

2. 评测指标展示即分析

1) K-NN 分类

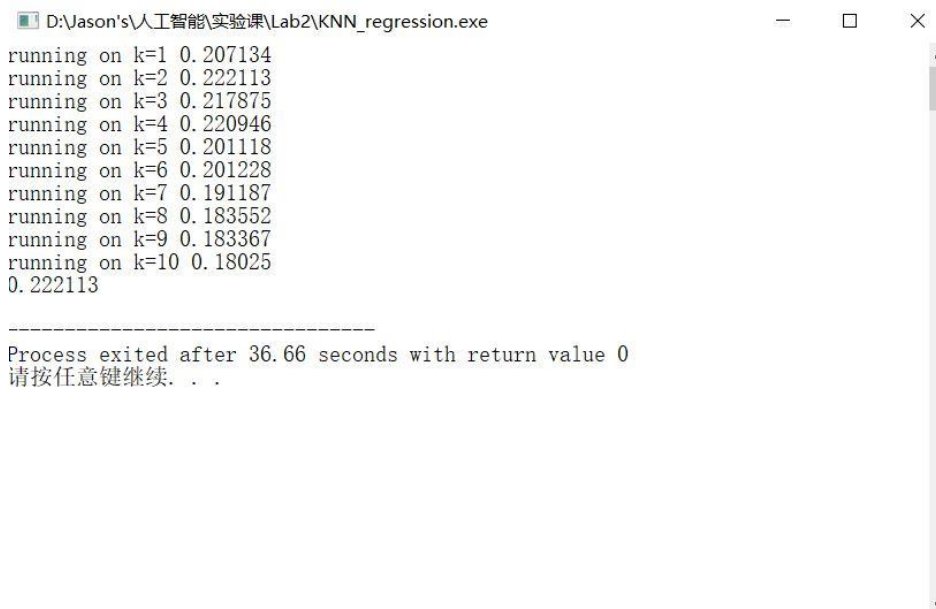


```
D:\Jason's\人工智能\实验课\Lab2\KNN_classification.exe
running on k=15 0.40836
running on k=16 0.440514
running on k=17 0.421222
running on k=18 0.414791
running on k=19 0.414791
running on k=20 0.421222
running on k=21 0.40836
running on k=22 0.401929
running on k=23 0.395498
running on k=24 0.398714
running on k=25 0.392283
running on k=26 0.395498
running on k=27 0.395498
running on k=28 0.392283
running on k=29 0.395498
running on k=30 0.392283
running on k=31 0.395498
running on k=32 0.389068
running on k=33 0.392283
running on k=34 0.385852
best k=16

-----
Process exited after 88.23 seconds with return value 0
请按任意键继续. . .
```

如图，当 k 取 16 时，最高的准确度为 44.05%

2) K-NN 回归



```
D:\Jason's\人工智能\实验课\Lab2\KNN_regression.exe
running on k=1 0.207134
running on k=2 0.222113
running on k=3 0.217875
running on k=4 0.220946
running on k=5 0.201118
running on k=6 0.201228
running on k=7 0.191187
running on k=8 0.183552
running on k=9 0.183367
running on k=10 0.18025
0.222113

-----
Process exited after 36.66 seconds with return value 0
请按任意键继续. . .
```

如图，当 k 取 2 时，最高的准确度为 0.222113 (相关系数)



3) 朴素贝叶斯分类

```
D:\Jason's\人工智能\实验课\Lab2\NB_classification.exe
evaluate 0.382637

-----
Process exited after 0.4304 seconds with return value 0
请按任意键继续. . .
```

最优的准确度为 38.26%

4) 朴素贝叶斯回归

```
D:\Jason's\人工智能\实验课\Lab2\NB_regression2.exe
0.0149002

-----
Process exited after 5.298 seconds with return value 0
请按任意键继续. . .
```

可能是引入拉普拉斯平滑时出现的问题，导致只有弱相关的结果。

四、 思考题

1.在 k-NN 回归根据相似度加权为什么用倒数作为权重？

因为基本的思想是距离越小的样本应该给更大的权重，因此取倒数可以很好的将两者关联起来，但是要预先处理距离为 0 的情况。

2. k-NN 同一测试样本的各个情感概率总和应该为 1，如何处理？

使用一个简单的归一化即可，也就是概率除以概率总和。

3.对于曼哈顿距离与欧式距离，在矩阵稀疏程度不同时，两者表现有什么区别，为什么？

曼哈顿距离可以产生稀疏权值矩阵，因为计算各个绝对值之和很容易产生 0 元素。

欧式距离可以防止过拟合，因为权值向量考虑到各个方向上的平方和求根。

4.伯努利模型与多项式的优缺点？

伯努利模型实现简单，但是结果可靠度不高，没办法根据词频进行计算。多项式可以给词频高的项更大的权重，但是实现更加复杂。

5.使用朴素贝叶斯算法时，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？

引入拉普拉斯平滑函数，通常为分子加一，分母加样本总数，但是分子为 1 是过于平滑的，实际取值需要远小于 1。