

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	m2	专业(方向)	软件工程(移动工程信息)
学号	15352446	姓名	钟展辉

一、实验题目

实验二: K 近邻与朴素贝叶斯——分类和回归

二、实验内容

1. 算法原理

①训练集、验证集和测试集的用法: 训练集和验证集都是有附带标签或者说正确答案的, 而测试集没有。训练集用来构建模型, 验证集用于调试模型以提高其准确度, 测试集是最后将模型付诸实践所用。

②分类问题是预测离散值的问题, 最终根据文本猜测、得到某一个情绪; 回归问题是预测连续值的问题, 最终得到的是根据文本猜测出每一种情绪的概率。

③K-NN (k-Nearest Neighbor) 的思想简单来说就是, 要评价一个未知的东西 X, 只需找 k 个与 X 相似的已知的东西, 并通过 k 个已知的, 对 X 进行评价。一般是选取与 X 特征最相似, 也就是说距离最邻近的 k 个点, 而这个距离的衡量方式有很多种, 在这个实验中可以有曼哈顿距离、欧氏距离、余弦距离等。

④NB (Naïve Bayesian), 贝叶斯定理: 已知某条件概率, 如何得到两个事件交换后的概率,

也就是在已知 $P(A|B)$ 的情况下如何求得 $P(B|A)$? 其公式为 $P(B|A) = \frac{P(A|B)P(B)}{P(A)}$ 。而朴素贝叶斯的思想基础是这样的: 对于给出的待分类项, 求解在此项出现的条件下各个类别出现的概率, 哪个最大, 就认为此待分类项属于哪个类别。

2. 伪代码

关于建立 onehot、TF、TFIDF 矩阵、读取 txt/csv 文件、写入 txt/csv 文件等代码基本继承 lab1 代码, 在此基础上对代码进行增删。

以下伪代码思路是用验证集调试训练集模型, 没有运行测试集。



①KNN_classification:

```
double Accuracy(int start,int end)
```

```
{
```

```
    对比 Label 数组和 my_label 数组，计算准确率。
```

```
}
```

```
double Distance(int a,int b)
```

```
{
```

```
    利用 TFIDF 矩阵计算 a、b 的余弦距离；
```

```
}
```

```
void Classify(int cur)//cur 为当前验证集文本在 matrix 矩阵/Label 数组中所在行的序号
```

```
{
```

```
    调用 Distance 函数求该验证集文本与每条训练集文本的距离，结果存储在 Dis 数组中；
```

```
    取 Dis 数组前 k 个最大的数（余弦距离越大，两文本越接近）；
```

```
    设置 mood[6]数组代表六种情绪的可能性大小；
```

```
    以余弦距离作加权，根据前 k 个最大的文本计算得出六种情绪的可能性大小，即 mood 数组，最后取出其中可能性最大的情绪，记录在数组 my_label 中（即我猜测的情绪）。
```

```
}
```

```
void KNN_Classification()
```

```
{
```

```
    用文件流先后打开训练集文件和验证集文件，读取文本单词，存储在 string 类型的二维数组 word_matrix[][]中，读取每行文本的标签，存储在 string 数组 Label 中；
```

```
    其中 train_num 为训练集文本数，row 为总文本数，则训练集存储在二维数组 word_matrix[0] ~ word_matrix[train_num] 间，验证集存储在 word_matrix[train_num] ~ word_matrix[row]间，Label 数组同理；
```

```
    将训练集和验证集文本一起做成 TFIDF 矩阵，其中当验证集文本出现训练集文本中没有出现过的新单词时不增加新列，而是直接忽视这个新单词；
```

```
    for(int i=train_num;i<row;i++) Classify(i);
```

```
    调用函数 Accuracy 计算准确率。
```

```
}
```

```
int main()
```

```
{
```

```
    KNN_Classification();
```

```
}
```



②KNN_regression:

```
double Distance(int a,int b)
```

```
{
```

利用 TFIDF 矩阵计算 a、b 的余弦距离；

```
}
```

```
void Regress(int cur)//cur 为当前验证集文本在 matrix 矩阵/Label 数组中所在行的序号
```

```
{
```

调用 Distance 函数求该验证集文本与每条训练集文本的距离，结果存储在 Dis 数组中；

取 Dis 数组前 k 个最大的数（余弦距离越大，两文本越接近）；

predict[6][1000]数组代表六种情绪的概率；

以余弦距离作加权，累加前 k 个文本的各情绪的概率，存储在 predict[i][cur]中，最后作概率标准化（各情绪概率相加为 1），得到 predict[i][cur]；

```
}
```

```
void KNN_Regreesion()
```

```
{
```

用文件流先后打开训练集文件和验证集文件，读取文本单词，存储在 string 类型的二维数组 word_matrix[][]中，读取每行文本的标签，存储在 string 数组 actual[6 种情绪][600 多行训练集文本]中；

其中 train_num 为训练集文本数，row 为总文本数，则训练集存储在二维数组 word_matrix[0] ~ word_matrix[train_num] 间，验证集存储在 word_matrix[train_num] ~ word_matrix[row]间，actual 数组同理；

将训练集和验证集文本一起做成 TFIDF 矩阵，其中当验证集文本出现训练集文本中没有出现过的新单词时不增加新列，而是直接忽视这个新单词；

```
for(int i=train_num;i<row;i++) Regress(i);
```

用文件流按实验要求格式输出验证集各文本的 6 个情绪的预测概率（predict 数组）（复制到 validation.xlsx 中进行相关系数评估）

```
}
```

```
int main()
```

```
{
```

```
KNN_Regreesion();
```

```
}
```



③NB_classification:

```
double Accuracy(int start,int end){
```

对比 Label 数组和 my_label 数组，计算准确率。

```
}
```

```
double Distance(int a,int b){
```

利用 TFIDF 矩阵计算 a、b 的余弦距离；

```
}
```

```
void Classify(int cur)//cur 为当前验证集文本在 matrix 矩阵/Label 数组中所在行的序号
```

```
{
```

设置 mood[6]数组代表六种情绪的可能性大小；

遍历该验证集文本（onehot 矩阵中的这一行），当检测到非零数值时，遍历所有训练集范围的 onehot 矩阵，统计出下列数据：

（假设验证文本标签为 sad）

sum，所有标签为 sad 的训练集文本的总单词数目；

num，当前所遍历到的单词在所有标签为 sad 的训练集文本中出现的次数；

chance，标签为 sad 的训练集文本的数目；

*num_of_word，训练集所有不重复的单词数目；

每次检测到非零数值时，都按公式 $mood[i] = (temnum + coe) / (sum + (coe * num_of_word))$ ；累积，（运用了拉普拉斯平滑），最后再乘以 sad 的占比即 $mood[i] = chance / train_num$ ；得到的 mood 数组，选出其中最大值，其情绪存储在 my_label 数组中。

```
}
```

```
void NB_Classification()
```

```
{
```

用文件流先后打开训练集文件和验证集文件，读取文本单词，存储在 string 类型的二维数组 word_matrix[][] 中，读取每行文本的标签，存储在 string 数组 Label 中；

其中 train_num 为训练集文本数，row 为总文本数，则训练集存储在二维数组 word_matrix[0] ~ word_matrix[train_num] 间，验证集存储在 word_matrix[train_num] ~ word_matrix[row] 间，Label 数组同理；

将训练集和验证集文本一起做成 onehot 矩阵，其中当验证集文本出现训练集文本中没有出现过的新单词时不增加新列，而是直接忽视这个新单词；

```
for(int i=train_num;i<row;i++) Classify(i);
```

调用函数 Accuracy 计算准确率。

```
}
```

```
int main()
```

```
{
```

```
NB_Classification();
```

```
}
```

④NB_regress:

```
void Regress(int cur)//cur 为当前验证集文本在 matrix 矩阵/Label 数组中所在行的序号
{
    double mood[6];
    double temlabel[6][1000];
    temlabel 数组复制 actual 数组中的内容;
    for(int i=0;i<TF 矩阵列数;i++)//matrix 为 TF 矩阵
    {
        if(matrix[cur][i]==0) continue;
        for(int j=0;j<train_num;j++)
            for(int z=0;z<6;z++)
                temlabel[z][j]*=matrix[j][i];//累乘当前训练集文本中与当前验证集文本相
                同的单词对应的矩阵数值
    }
    循环遍历 temlabel 数组, 对每个情绪都累加其在各个训练集文本的 temlabel 值, 结果
    存储在 mood 数组中。最后作概率标准化(各情绪概率相加为 1), 得到 predict[i][cur];
}
```

```
void NB_Regression()
```

```
{
```

用文件流先后打开训练集文件和验证集文件, 读取文本单词, 存储在 string 类型的二维数组 word_matrix[][] 中, 读取每行文本的标签, 存储在 string 数组 actual[6 种情绪][600 多行训练集文本]中;

其中 train_num 为训练集文本数, row 为总文本数, 则训练集存储在二维数组 word_matrix[0] ~ word_matrix[train_num] 间, 验证集存储在 word_matrix[train_num] ~ word_matrix[row] 间, actual 数组同理;

将训练集和验证集文本一起做成 TF 矩阵, 其中当验证集文本出现训练集文本中没有出现过的新单词时不增加新列, 而是直接忽视这个新单词;

在构造 TF 矩阵时, 对训练集范围的数值, 进行拉普拉斯平滑操作。(验证集范围内不作改变)。

```
for(int i=train_num;i<row;i++) Regress(i);
```

用文件流按实验要求格式输出验证集各文本的 6 个情绪的预测概率(predict 数组)(复制到 validation.xlsx 中进行相关系数评估)

```
}
```

```
int main()
```

```
{
```

```
    NB_Regression();
```

```
}
```



3. 关键代码截图（带注释）

读取训练集文本（读取验证集相似）

```
ifstream fin;
fin.open("train_set.csv");//流变量连接文件，在同一个目录则不用指定路径只需要文件名。
string s;
int row=0;
getline(fin,s);//舍弃第一行
while(getline(fin,s))
{
    int comma=s.find(',');//寻找逗号
    string ss=s.substr(0,comma);//substr(a,b) 截取从第a位置开始长度为b的字符串
    string label=s.substr(comma+1);
    getword(ss,row);//取得文本单词
    getlabel(label,row);//取得文本标签
    row++;
    //if(row>5) break;
}
train_num=row;
fin.close();
cout<<train_num<<endl;
```

getword 函数:

```
void getword(string s,int row)
{
    int cnt=0;
    stringstream ss;
    ss.clear();
    ss.str(s);
    while(ss)
    {
        ss>>word_matrix[row][cnt++];
    }
    num[row]=cnt-1;
}
```

分类的 getlabel 函数:

```
void getlabel(string l,int row)
{
    Label[row]=l;
}
```

回归的 getlabel 函数:



```
void getlabel(string l,int row)
{
    int start=0,end;
    double temnum;
    int cnt=0;
    for(int i=0;i<l.size();i++)
    {
        if(l[i]==','||i==l.size()-1)
        {
            if(l[i]==',') end=i-1;
            else end=i;
            string tems=l.substr(start,end-start+1);
            start=i+1;

            stringstream ss;
            ss.clear();
            ss.str(tems);
            ss>>temnum;
            setlabel(temnum,row,cnt++);
        }
    }
}
```

求余弦距离:

```
double Distance(int a,int b)
{
    double A=0,B=0,AB=0;
    for(int i=0;i<num_of_word;i++)
    {
        A+=matrix[a][i];
        B+=matrix[b][i];
        AB+=matrix[a][i]*matrix[b][i];
    }
    A=sqrt(A);B=sqrt(B);
    double result=AB/(A*B);
    return result;
}
```

寻找前 k 个最大值:

```
double maxx;
int Index[40]; // 记录k个最大的余弦距离的索引
double index_dis[40]; // 记录k个最大的余弦距离的值
for(int i=0;i<k;i++) // 冒泡法求前k个最大值
{
    maxx=0;
    for(int j=0;j<train_num;j++)
    {
        if(Dis[j]>maxx)
        {
            maxx=Dis[j];
            Index[i]=j;
            index_dis[i]=Dis[j];
        }
    }
    Dis[Index[i]]=0; // 把当前找到的最大值设为0使其不影响第二个最大值的寻找
}
```




KNN_classification 中, 以余弦距离作加权, 取得前 k 个情绪中可能性最大的情绪:

```
double mood[6];
for(int i=0;i<6;i++) mood[i]=0;
string mood_name[6]={"anger","disgust","fear","joy","sad","surprise"};
for(int i=0;i<k;i++)
{
    for(int j=0;j<6;j++)
        if(Label[Index[i]]==mood_name[j])
        {
            mood[j]+=index_dis[i]; break;
        }
}
double maxnum=0;
int maxindex;
for(int i=0;i<6;i++)
{
    if(mood[i]>maxnum)
    {
        maxnum=mood[i];
        maxindex=i;
    }
}
my_label[cur]=mood_name[maxindex];
```

KNN_regression 中, 以余弦距离作加权, 累加前 k 个文本的各情绪的概率, 存储在 predict[i][cur] 中

```
for(int i=0;i<k;i++)
{
    for(int j=0;j<6;j++)
    {
        if(index_dis[i]!=0) predict[j][cur]+=actual[j][Index[i]]*index_dis[i];
    }
}
```

NB_classification, 朴素贝叶斯分类模型:

```
double mood[6];
memset(mood,0,sizeof(mood));
for(int i=0;i<6;i++)
{
    double sum,temnum,chance;//sum, 相同情感, 所有单词数目;
    //num, 当前单词在相同情感中出现次数, chance当前情感所占个数
    mood[i]=1;
    for(int j=0;j<num_of_word;j++)
    {
        if(matrix[cur][j]==0) continue;
        sum=0; temnum=0; chance=0;
        for(int k=0;k<train_num;k++)
        {
            if(Label[k]==mood_name[i])
            {
                chance++;
                sum+=num[k]; //num[k]是当前训练文本的单词数
                temnum+=matrix[k][j];
            }
        }
        mood[i]*=(temnum+coe)/(sum+(coe*num_of_word)); //运用了拉普拉斯平滑
    }
    mood[i]*=chance/train_num; //乘以当前情绪在所有情绪中的占比
}
```




NB_regression

```
double mood[6];
for(int i=0;i<6;i++) mood[i]=0;
double temlabel[6][1000];
for(int i=0;i<6;i++)//temlabel数组复制actual数组中的内容:
    for(int j=0;j<train_num;j++) temlabel[i][j]=actual[i][j];
for(int i=0;i<num_of_word;i++)//遍历当前验证集所在TF矩阵所在行的每一列
{
    if(matrix[cur][i]==0) continue;
    for(int j=0;j<train_num;j++)//遍历所有训练集文本
    {
        for(int z=0;z<6;z++)
        {
            temlabel[z][j]*=matrix[j][i];
            //累乘当前训练集文本中与当前验证集文本相同的单词对应的矩阵数值
        }
    }
}
//循环遍历temlabel数组, 对每个情绪都累加其在各个训练集文本的temlabel值, 结果存储在mood数组中
for(int i=0;i<6;i++)
{
    for(int j=0;j<train_num;j++)
    {
        mood[i]+=temlabel[i][j];
    }
}
for(int i=0;i<6;i++) predict[i][cur]=mood[i];
```

4. 创新点&优化（如果有）

优化:

我总结了各个代码优化过程中的效果，其中的概率都是调试 k 从 4 取到 20 之间时获得的最大概率：

KNN_classification:

```
c0: 0.40  欧式距离 onehot
c1: 0.43  余弦距离 onehot
c2: 0.46  余弦距离 onehot 余弦距离作权值 (double数值初始化)
c3: 0.4744(k=13) 余弦距离 tfidf 余弦距离作权值 (double数值初始化)
```

KNN_regression:

```
re: 0.22  onehot 欧式距离 最后概率标准化
re2:0.21  tf-idf 欧式距离 最后概率标准化
re3:0.27  one-hot 余弦距离 最后概率标准化
re4:0.402 (k=11)  tfidf 余弦距离 最后概率标准化
```

可见使用余弦距离和 TFIDF 矩阵是优化效果最好的。

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的 TF-IDF。因此，TF-IDF 倾向于过滤掉常见的词语，保留重要的词语。

NB_classification 和 NB_regression 都使用拉普拉斯平滑优化，或者说，不使用拉普拉斯平滑的话，根据朴素贝叶斯公式计算出来的概率基本都是 0，因为矩阵十分稀疏，验证集文本与训练集文本如果不完全相同的话，得到的概率都是 0，这样得到的结果完全不可靠。

比如 NB_regression，不使用拉普拉斯平滑得到的结果：



textid	anger	disgust	fear	joy	sad	surprise
1	0	0	0	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	0	0
6	0	0	0	0	0	0
7	0	0	0	0	0	0
8	0	0	0	0	0	0
9	0	0	0	0	0	0
10	0	0	0	0	0	0
11	0	0	0	0	0	0
12	0.026505	0.044209	0.24785	0.026505	0.557612	0.09732
13	0	0	0	0	0	0
14	0	0	0	0	0.7907	0.2093
15	0	0	0	0	0	0
16	0	0	0	0.7386	0	0.2614
17	0	0	0	0	0	0
18	0	0	0	0	0	0
19	0	0	0	0	0	0
20	0	0.3131	0	0.1616	0	0.5253
21	0.133313	0.124412	0.222222	0.217822	0.137814	0.164416
22	0	0	0	0	0	0
23	0.050799	0.040241	0.048814	0.467016	0.128113	0.265017
24	0	0	0	0	0	0
25	0	0	0	0	0	0
26	0	0	0	0	0	0
27	0	0	0	0	0	0
28	0	0	0	0	0	0

使用拉普拉斯平滑：

textid	anger	disgust	fear	joy	sad	surprise
1	0.079485	0.093972	0.232626	0.238086	0.17312	0.182712
2	0.079405	0.035182	0.266445	0.204783	0.205202	0.208983
3	0.060954	0.034803	0.247969	0.191193	0.152974	0.312107
4	0.076832	0.059079	0.267179	0.099616	0.349726	0.147566
5	0.047909	0.02758	0.126646	0.379354	0.185634	0.232877
6	0.259789	0.082027	0.194525	0.036437	0.259369	0.167852
7	0.169335	0.073206	0.23734	0.122784	0.219516	0.177819
8	0.060468	0.054362	0.107395	0.498436	0.074077	0.205263
9	0.04953	0.081583	0.19275	0.229136	0.183766	0.263235
10	0.047784	0.052614	0.161823	0.367935	0.130627	0.239216
11	0.04859	0.009438	0.152145	0.477466	0.050607	0.261755
12	0.032317	0.026875	0.257203	0.118679	0.343208	0.221718
13	0.215399	0.126976	0.108337	0.043411	0.271167	0.234709
14	0.029382	0.022447	0.078725	0.052285	0.53867	0.278491
15	0.114174	0.068778	0.182089	0.336845	0.166136	0.131978
16	0.078869	0.038469	0.070194	0.437094	0.121087	0.254286
17	0.174838	0.143535	0.079269	0.318297	0.085266	0.198795
18	0.021014	0.01967	0.150808	0.479221	0.13164	0.197648
19	0.022499	0.02106	0.245938	0.260957	0.126721	0.322825
20	0.073419	0.129436	0.189446	0.234669	0.180915	0.192115
21	0.179158	0.162023	0.225287	0.116682	0.231257	0.085594
22	0.044325	0.067145	0.191693	0.245023	0.207794	0.24402
23	0.025242	0.02043	0.131718	0.412756	0.117465	0.292389
24	0.110729	0.051991	0.187777	0.098128	0.327401	0.223973
25	0.081079	0.018171	0.236563	0.239831	0.212531	0.211825
26	0.110759	0.064308	0.065041	0.395446	0.062087	0.302358
27	0.092554	0.064782	0.175248	0.244814	0.27091	0.151693
28	0.050574	0.066739	0.05765	0.366519	0.084975	0.373544

为什么要做平滑处理？

零概率问题，就是在计算实例的概率时，如果某个量 x ，在训练集中没有出现，会导致整个实例的概率结果是 0。当一个词语没有在训练样本中出现，该词语的概率为

0，使用连乘计算文本出现概率时也为 0。这是不合理的，不能因为一个事件没有观察到就武断的认为该事件的概率是 0。

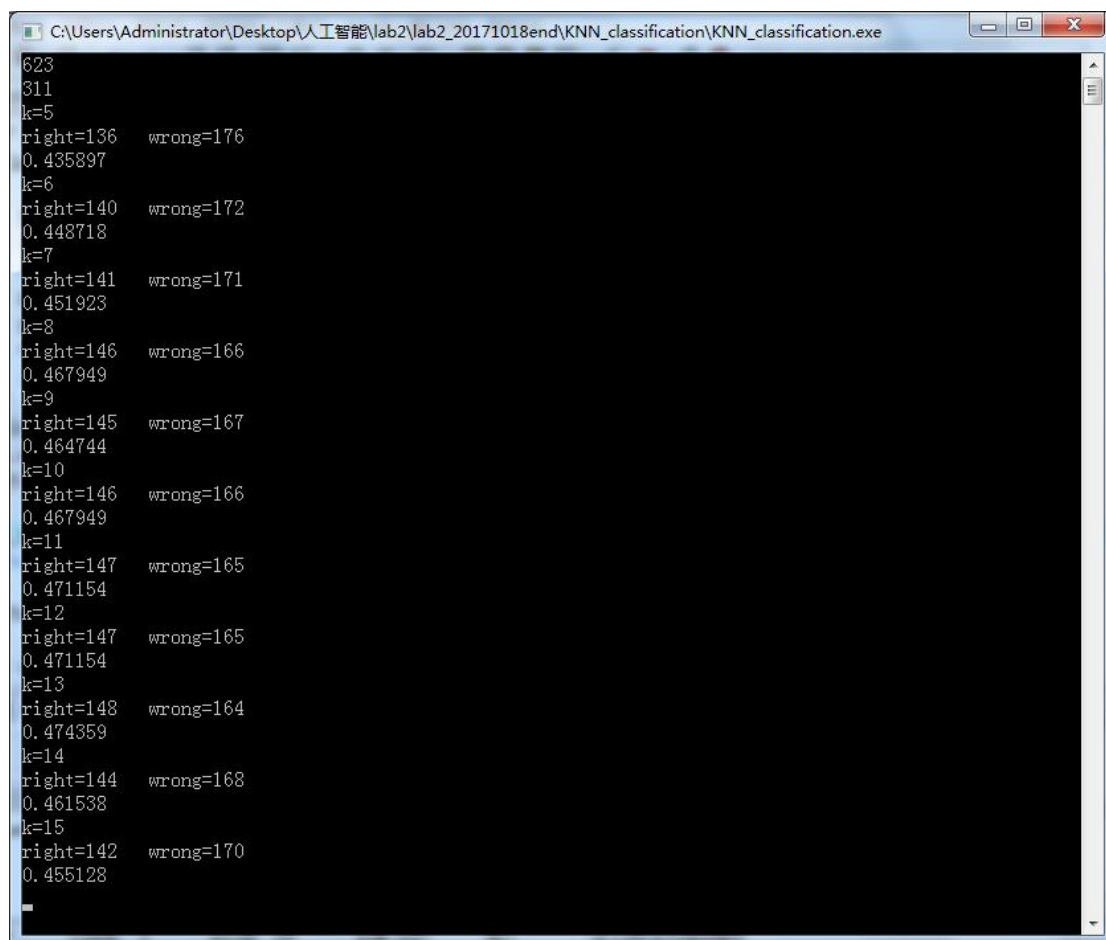
当训练样本很大时，每个分量 x 的计数加 1 造成的估计概率变化可以忽略不计，但可以方便有效的避免零概率问题。

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）

展示模型应用于验证集时得到的准确度和相关度。

①KNN_classification: 当 k 取 5~15 之间时，验证集预测的准确率：



```
623
311
k=5
right=136   wrong=176
0.435897
k=6
right=140   wrong=172
0.448718
k=7
right=141   wrong=171
0.451923
k=8
right=146   wrong=166
0.467949
k=9
right=145   wrong=167
0.464744
k=10
right=146   wrong=166
0.467949
k=11
right=147   wrong=165
0.471154
k=12
right=147   wrong=165
0.471154
k=13
right=148   wrong=164
0.474359
k=14
right=144   wrong=168
0.461538
k=15
right=142   wrong=170
0.455128
```

当 $k=13$ 时准确率最高。

②KNN_regression:

取 $k=12$ 时相关度最高



	A	B	C	D	E	F	G	H	I	J
1	textid	anger	disgust	fear	joy	sad	surprise			anger
2	1	0.079485	0.093972	0.232626	0.238086	0.17312	0.182712		r	0.371007165
3	2	0.079405	0.035182	0.266445	0.204783	0.205202	0.208983		average	0.400392109
4	3	0.060954	0.034803	0.247969	0.191193	0.152974	0.312107		evaluatic	低度相关 666
5	4	0.076832	0.059079	0.267179	0.099616	0.349726	0.147566			
6	5	0.047909	0.02758	0.126646	0.379354	0.185634	0.232877			
7	6	0.259789	0.082027	0.194525	0.036437	0.259369	0.167852			
8	7	0.169335	0.073206	0.23734	0.122784	0.219516	0.177819			
9	8	0.060468	0.054362	0.107395	0.498436	0.074077	0.205263			
10	9	0.04953	0.081583	0.19275	0.229136	0.183766	0.263235			
11	10	0.047784	0.052614	0.161823	0.367935	0.130627	0.239216			
12	11	0.04859	0.009438	0.152145	0.477466	0.050607	0.261755			
13	12	0.032317	0.026875	0.257203	0.118679	0.343208	0.221718			
14	13	0.215399	0.126976	0.108337	0.043411	0.271167	0.234709			
15	14	0.029382	0.022447	0.078725	0.052285	0.53867	0.278491			
16	15	0.114174	0.068778	0.182089	0.336845	0.166136	0.131978			
17	16	0.078869	0.038469	0.070194	0.437094	0.121087	0.254286			
18	17	0.174838	0.143535	0.079269	0.318297	0.085266	0.198795			
19	18	0.021014	0.01967	0.150808	0.479221	0.13164	0.197648			
20	19	0.022499	0.02106	0.245938	0.260957	0.126721	0.322825			
21	20	0.073419	0.129436	0.189446	0.234669	0.180915	0.192115			
22	21	0.179158	0.162023	0.225287	0.116682	0.231257	0.085594			
23	22	0.044325	0.067145	0.191693	0.245023	0.207794	0.24402			
24	23	0.025242	0.02043	0.131718	0.412756	0.117465	0.292389			
25	24	0.110729	0.051991	0.187777	0.098128	0.327401	0.223973			
26	25	0.081079	0.018171	0.236563	0.239831	0.212531	0.211825			
27	26	0.110759	0.064308	0.065041	0.395446	0.062087	0.302358			
28	27	0.092554	0.064782	0.175248	0.244814	0.27091	0.151693			
29	28	0.050574	0.066739	0.05765	0.366519	0.084975	0.373544			

③NB_classification: 拉普拉斯平滑 coe 系数取 0.44 时, 准确率最高

```
C:\Users\Administrator\Desktop\人工智能\lab2\lab2_20171018end\NB_classification\NB_classification.exe
623
311
coe=0.24
right=137      wrong=175
0.439103
coe=0.28
right=140      wrong=172
0.448718
coe=0.32
right=142      wrong=170
0.455128
coe=0.36
right=146      wrong=166
0.467949
coe=0.4
right=148      wrong=164
0.474359
coe=0.44
right=148      wrong=164
0.474359
coe=0.48
right=147      wrong=165
0.471154
coe=0.52
right=148      wrong=164
0.474359
coe=0.56
right=148      wrong=164
0.474359
coe=0.6
right=147      wrong=165
0.471154
coe=0.64
right=147      wrong=165
0.471154
coe=0.68
right=146      wrong=166
```




④NB_regression: 取 coe 系数为 0.009

	A	B	C	D	E	F	G	H	I	J
1	textid	anger	disgust	fear	joy	sad	surprise			anger
2	1	0.100655	0.087649	0.213104	0.225187	0.187965	0.18544		r	0.314988863
3	2	0.0436	0.04255	0.190332	0.233043	0.136363	0.354113		average	0.354033376
4	3	0.070098	0.044242	0.238456	0.20695	0.184085	0.256169		evaluation	低度相关 666
5	4	0.081059	0.051263	0.18816	0.217633	0.271444	0.19044			
6	5	0.053025	0.029773	0.174249	0.318811	0.200813	0.223328			
7	6	0.109919	0.057436	0.126591	0.276714	0.192318	0.23702			
8	7	0.160379	0.078417	0.326803	0.049704	0.271561	0.113136			
9	8	0.08699	0.08699	0.043499	0.521645	0.108689	0.152187			
10	9	0.009805	0.046987	0.244442	0.172837	0.027532	0.498398			
11	10	0.076873	0.078285	0.238187	0.113227	0.285076	0.208353			
12	11	0.037375	0.001895	0.106321	0.319555	0.190857	0.343997			
13	12	0.026505	0.044209	0.24785	0.026505	0.557612	0.09732			
14	13	0.105576	0.105071	0.11109	0.094305	0.295259	0.288698			
15	14	0.000289	0.000705	0.001277	0.000164	0.788066	0.209499			
16	15	0.084524	0.069067	0.191609	0.259729	0.200782	0.194289			
17	16	5.12E-06	2.91E-06	1.07E-05	0.738568	1.80E-05	0.261395			
18	17	0.114871	0.05119	0.156273	0.298457	0.154578	0.224631			
19	18	0.015091	0.008895	0.091584	0.536302	0.177497	0.17063			
20	19	0.055865	0.033954	0.210642	0.214124	0.179081	0.306334			
21	20	0.033627	0.215613	0.079447	0.194982	0.08465	0.391682			
22	21	0.133331	0.124418	0.222254	0.217741	0.137881	0.164374			
23	22	6.14E-05	5.65E-05	0.093451	0.546463	0.08001	0.279958			
24	23	0.051859	0.049834	0.068402	0.387906	0.12523	0.316768			
25	24	0.081173	0.134359	0.141091	0.13406	0.318536	0.190781			
26	25	0.039057	0.016673	0.076189	0.46792	0.09276	0.307402			
27	26	0.096134	0.062852	0.176364	0.217671	0.233741	0.213238			
28	27	0.139236	0.092606	0.230245	0.06229	0.320586	0.155038			
29	28	0.052377	0.047201	0.062857	0.250564	0.090227	0.506265			

2. 评测指标展示及分析（如果实验题目有特殊要求，否则使用准确率）

上面的实验结果展示即是评测指标（准确率、相关度）展示。

分析：

KNN_classification: 使用了余弦距离、tfidf 矩阵以及余弦距离加权

KNN_regression: 使用了余弦距离、tfidf 矩阵

NB_classification: 使用 onehot 矩阵，拉普拉斯平滑

NB_regression: 使用 TF 矩阵，拉普拉斯平滑

四、 思考题

1、

计算test1与每个train的距离，选取TopK个训练数据
把该距离的倒数作为权重，计算test1属于该标签的概率：

$$P(\text{test1 is happy}) = \frac{\text{train1 probability}}{d(\text{train1, test1})} + \frac{\text{train2 probability}}{d(\text{train2, test1})} + \frac{\text{train3 probability}}{d(\text{train3, test1})}$$

思考：为什么是倒数呢？

思考：同一测试样本的各个情感概率总和应该为1 如何处理？

在 TopK 个数据中选出与当前验证/测试文本最相近的、最有可能的情绪，所以应该是出现概率越高、距离越近的情绪可能性越高。而公式中的距离是欧式距离，欧氏距离越小，距离越近，因此用倒数形式；如果用余弦距离，则不需要用倒数形式，因为余弦距离越大，距离越近。



为了使同一测试样本的各个情感概率总和为 1，使其标准化。

```
//cur 为当前测试样本序号
```

```
double sum=0;
```

```
for(int i=0;i<6;i++) sum+=predict[i][cur];
```

```
for(int i=0;i<6;i++) if(sum!=0) predict[i][cur]/=sum;
```

2、

• 距离公式：

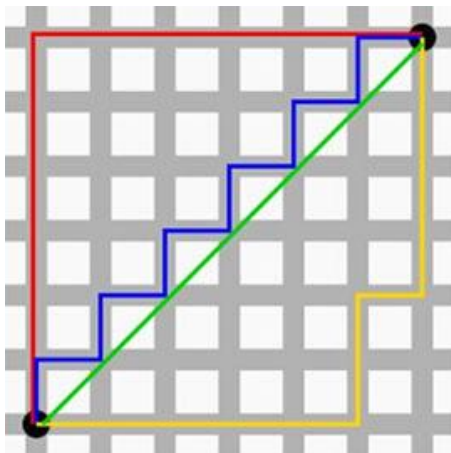
L_p 距离(所有距离的总公式)：

$$L_p(x_i, x_j) = \left\{ \sum_{i=1}^n |x_i^{(l)} - x_j^{(l)}|^p \right\}^{\frac{1}{p}}$$

- $p = 1$ ：曼哈顿距离；
- $p = 2$ ：欧式距离，最常见。

思考：在矩阵稀疏程度不同的时候，这两者表现有什么区别，为什么？

矩阵较为稀疏时，曼哈顿距离与欧氏距离差别不大，矩阵较为密集时，曼哈顿距离比欧式距离大很多，如下图所示，矩阵越密集，点数越多，两个文本在标准坐标系上的绝对轴距总和越大。此时，就要考虑曼哈顿距离跟欧氏距离各自的特点。当矩阵密集时，曼哈顿距离比欧式距离更稳定，因为很有可能会出现较多文本欧式距离相同，这时欧式距离难以用来筛选，而曼哈顿距离能掩盖特征之间的邻近关系。





3、

ID	text	class label
1	good,thanks	joy
2	No impressive, thanks	sad
3	Impressive good	joy
4	No, thanks	?

Bernoulli Model (伯努利模型):

$$P_{(\text{thanks}|\text{joy})} = 1/2$$



Multinomial Model (多项式模型):

ID	goods	thanks	no	impressive	class label
1	1	1	0	0	joy
2	0	1	1	1	sad
3	1	0	0	1	joy
4	0	1	1	0	?

$$P_{(\text{thanks}|\text{joy})} = 1/4$$

思考题：这两个模型分别有什么优缺点

二者的计算粒度不一样，多项式模型以单词为粒度（考虑了单词出现的频率），伯努利模型以文件为粒度（只考虑单词出现与否）

伯努利模型优点：实际中，训练文档通常充足，不使用单词在文档中的频率信息，也可以很好的分类，过多考虑频率信息非不会对分类有帮助，反而起相反作用。

伯努利模型缺点：只考虑单词出现和不出现的情况，忽略了频率信息（有可能混淆了重要单词和不重要单词区别）。

多项式模型优点：在训练阶段可以不考虑频率信息，但是在分类阶段，我们针对文档，这时单词在文档中的频数信息尤为重要。如果仅仅考虑出现与否，不同类别出现共同频数高的词被忽略，可能导致分类误差大。

多项式模型缺点：假设过于严格，即假设同一单词在同一文档中的多次出现是独立的（事实并非如此）。

4、

- (1) 分类（使用**准确率**衡量结果）
分类只要求实现**多项式模型**
- (2) 回归（使用**相关系数**衡量结果）
 - 归一化最后的情感概率，使得六中情感概率相加为 1
 - 本次实验同样提供了 validation 数据集
- (3) 推荐实现拉普拉斯平滑

思考题：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

没出现过的单词也就无从利用，而且单词不影响计算该测试文本与训练集文本的距离，为了降低算法复杂度，在将测试文本转化成 TF 矩阵格式时直接将这个词摒弃。

|----- 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