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	[redacted]
学号	[redacted]

一、 实验题目

K 邻近和朴素贝叶斯——分类和回归

二、 实验内容

1. 算法原理

（1）KNN 算法：

《1》 KNN 算法是一种基于实例的算法，即学习过程知识简单的存储已知的训练数据，遇到新的查询实例时，从训练集中取出相似的实例，因此 KNN 算法是一种懒惰学习方法。

《2》 KNN 分类工作原理是：存在一个样本数据集合，并且样本集中每个数据都存在标签，即我们知道样本每一个数据与所属分类的对应关系，输入没有标签的新数据后，将新数据的每个特征与样本集中的数据对应的特征进行对比，然后提取样本集中特征最相似的 K 个数据，则将 K 个数据中出现次数最多的类别作为该样本的类别。

KNN 回归的工作原理类似，是：在特征空间中找出 k 个最相似(即特征空间中最邻近)的样本，将这些样本的属性的平均值（或者是将不同距离的的 K 个样本对该样本产生的影响给予不同的权重，通过权重算出的属性值）赋给该样本，从而得到该样本的属性。

《3》 KNN 模型的优缺点：

优点：① 精度高，对异常值不敏感，无数据输入假定；

② 思想简单，理论成熟，既可以用来分类也可以用来做回归，即可用来做线性分类，也可以用来做非线性分类；

缺点：① 计算量大；

② 样本不平衡问题。[一个类的样本容量很大，而其他类的样本容量很小，有可能导致当输入一个新样本时，该样本的 K 个邻居中大容量类的样本占多数。]

（2）NB 算法：

《1》 NB 分类的工作原理：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。即 利用贝叶斯公式根据某对象的

先验概率计算出其后验概率，然后选择具有最大后验概率的类作为该对象所属的类。

NB 回归的工作原理：对于给出的对象，求解在此项出现的条件下对于训练集中的每个对象的各个特征出现的概率，将每个特征对于训练集的概率相加，再乘以其属性的概率，即可得到该对象该特征的概率。

《2》NB 模型的优缺点：

优点：① 算法简单、所需估计参数很少、对缺失数据不太敏感。

② 朴素贝叶斯的计算过程类条件概率等计算彼此是独立的，因此特别适于分布式计算

缺点：① 因为朴素贝叶斯分类假设样本各个特征之间相互独立，这个假设在实际应用中往往是不成立的，从而影响分类正确性；

② 不能学习特征间的相互作用;对输入数据的表达形式很敏感。

（3）平滑：

$\hat{P}(x_k|\omega_i)=\frac{|D_{i,x_k}|}{|D_i|}$ 如果后验概率的连乘式中有一项是 0，就会导致后验概率为 0。解决方法是使用平滑：$\hat{P}(x_k|\omega_i)=\frac{|D_{i,x_k}|+\alpha}{|D_i|+\alpha c_k}$，$C_k$ 表示第 K 为特征的可能取值的个数，系数 a 为非负数。如果 a=1 就是 Laplace 平滑，相当于认为特征的分布式均匀分布，先验地认为每个特征取值在每个类中出现一次；如果 0≤a <1，就是 Lidstone 平滑。

2. 伪代码

（1）KNN 分类：

```
/*作用：将一个字符串 str 切割成一个个单词，在 str 中，单词以空格分割*/
vector<string> get_word(string temp){
    设置切割字符串的标记为空格；
    for(遍历字符串 temp){
        每切割出一个单词，将该单词放入向量中，然后就在 Diff 单词集中寻找该单词；
        if( Diff 中没有该单词) 将该单词放入 Diff 单词集中；
    }
    返回存放着单词的向量;
}
```

```
/*作用：读取 trainning 文件，将每行文本的单词放入 train 中，情绪放在 train_label 中*/
void GetTrain(const char *filename){
    调用 open 函数打开文件；
    if（打开失败） return；
    else{
        while（一行一行地读取文件的内容，将每行文本的内容放入字符串 temp 中）{
            设置切割字符串的标记为逗号；
            将逗号后面的字符串切割出来，放入向量 train_label 中；
```

```
            在 temp 中删除逗号以及逗号之后的字符；
            调用函数 get_word 将 temp 中的 words 部分切割成单词；
        }
    }
    调用 close（）函数关闭文件；
}
```

```
/*读取 validation 文件，将每行文本的单词放入 validation 中，情绪放在 label 中*/
void GetVal(const char *filename)  和 GetTrain 类似。
/*作用：读取 test 文件，将每行文本的单词放入 test 中，情绪放到 test_label 中*/
void GetTest(const char *filename)  和 GetTrain 类似，只是要先切割掉每个文本前的序号。
```

```
/*作用：获得 one-hot 和 bridge 矩阵；one-hot:记录单词是否出现；bridge：记录单词出现的次数  */
void get_one_bri(){
    for(遍历 train 和 validation){
        for（遍历 Diff 单词集）{
            if（row< train 的行数）则在 train 中寻找 Diff；else  在  validation 中寻找；
            if（找不到）  one-hot 和 bridge 都记录为 0；
            else{
                 one-hot 记录为 1；
                while(继续在 train 或 validation 中寻找 Diff){
                    if（第一次找到）{
                        bridge 记录为 1；
                        并且在 train 或 validation 中删除该 Diff；
                     }else{
                        将 bridge 的记录++；
                     }
                }
            }
        }
    }
}
```

```
/*作用：获得 tf 矩阵 ；tf = (单词出现次数)/（文本单词总数）*/
void get_tf(){
    for（遍历 tf 的每一行）{
      for（遍历 bridge 的该行的每一列）将该行的数值相加得到文本单词总数；
      for（遍历 tf 该行的每一列）用 bridge 对应位置的数值/文本单词总数得到 tf 的值。
   }
}
```

```
/*作用：获取 tf_idf 矩阵 ； idf = log2(文本总数)/(出现该单词的文本数+1)；
  tfidf[i][j] = tf[i][j]*idf[i]   */
void get_tfidf(){
    for（遍历 one-hot 的每一列）{
      for（遍历 one-hot 的每一行）
      将 one-hot 每列的总数算出即可知一个单词出现在多少篇文本中；
      通过 idf = log2(文本总数)/(出现该单词的文本数+1)计算 idf 向量[D：train 的文本数];
    }
    通过 tfidf[i][j] = tf[i][j]*idf[i]计算得出 tf_idf 矩阵的值；
}
```

```
/*计算距离，选出 K 个 label，得到预测的 label*/
void get_result(int k,string dis_way,vector<vector<double> > two){
    for（遍历训练集 train 的 two 矩阵的每一行）{
        for（遍历 validation 的 two 矩阵的每一行）{
            通过不同的度量距离的方式来计算 validation 的某一行文本和 train 中所有文
            本的距离放入 distance 中；
        }
        if（度量距离的方式选择的是余弦相似度）将 distance 按从大到小排序；
        else  将 distance 按从小到大排序；
        选取 distance 的前 K 个距离对应的 train_label，将他们放入在 k_label 中；
        for（遍历 K_label，在里面找 6 个情感标签 diff_label）{
            ① 初始方法：每找到一种标签，就将该标签的出现次数++；
                        记录该标签在 diff_label 中的位置；
            ② 优化一：每找到一种标签，就将该标签的出现次数++；
     将之前记录的 label 的距离和该标签的距离对比，将距离小的 label 的位置记录起来；
            ③ 优化二：if（度量方式是余弦相似度）{
                            将该 label 在 diff_label 中的位置记录起来；
                            权重+= 距离的平方；
                        }else{
                            将该 label 在 diff_label 中的位置记录起来；
                            权重+= 1/距离的平方；
                        }
        }
        ①②将 k_label 按 label 的出现次数从大到小排序;选择出现次数最多的 label 作
        为预测 label；
        ③ 将 k_label 按权重从大到小排序，选择权重最大的 label 作为预测 label；
    }
}
```

（2）KNN 回归：

KNN 回归的代码前部分基本和 KNN 分类类似，所以这里只写出不同的伪代码。

不同点①：

因为 KNN 分类读取的文件的内容的格式不同，所以切割字符串不太一样。KNN 分类文件的内容格式为：words，label；而 KNN 回归是：words，情感系数 1…情感系数 6；

所以 KNN 回归切割字符串时要先以逗号为标记将后面 6 个情感系数切割出来放入 emotion 中，再将 words 切割成一个个单词。

不同点②：

```
/*计算距离，选出 K 个 label，得到预测的 label*/
void get_result(int k,string dis_way,vector<vector<double> > two){
    for（遍历训练集 train 的 two 矩阵的每一行）{
      for（遍历 validation 的 two 矩阵的每一行）{
          通过不同的度量距离的方式来计算 validation 的某一行文本和 train 中所有文本的距离放入 distance 中；
      }
      优化：按照公式 X' = (X - X_min) / (X_max - X_min) 对 distance 中存放的距离进行规范化；
      if（度量距离的方式选择的是余弦相似度）将 distance 按从大到小排序；
      else  将 distance 按从小到大排序；
      按照公式 P(test1 is happy) = train1 probability / d(train1,test1) + train2 probability / d(train2,test1) + train3 probability / d(train3,test1) 计算每个情感系数；
      【注意：度量方式是余弦相似度时是乘以 distance，而不是乘以倒数】
      将每行文本的每个情感系数除以情感系数总和使得他们的总和为 1；
   }
}
```

（3）NB 分类：

NB 分类的代码前部分基本和 KNN 分类类似，这里不再重复，只写出不同点。

① /*作用：获得 NumOfWords 矩阵*/

void get_numofwords()中的 NumOfWords 和 KNN 分类中的 bridge 矩阵类似，记录着每一行 train 中 Diff 单词集中每个单词出现的次数以及每一行 validation/test 中 Diff 单词集的每个单词是否出现（出现为 1，不出现为 0）。

```
② /*作用：计算在 train 中每个 label 出现的概率，以及出现的位置*/
void get_labelpossibility(){
    for(遍历 train 的 label){
        计算出 6 种情感中每种情感出现的次数；
        记录 6 中情感中每种情感在 train 中出现的位置；
    }
```

计算每种情感出现的概率 P（e_i）

《1》没优化：计算方式：$p(e_i)=\frac{N_{e_i}}{N}$

《2》优化：计算方式：P（e_i）= $\frac{N_{e_i}+1}{N+6}$

}

③ /*作用：运用多项式模型，计算出 在每种情感的前提下单词出现的概率 */

```
void Multinomial(){
  for(遍历每种情感){
    for(遍历出现该情感 label 的 train 的文本){
      计算出现该 label 的 train 文本的所有单词数；
      《1》优化：计算出该 label 的 train 文本的所有不重复单词数；
    }
    for(遍历 NumOfWords 中出现该 label 的行数){
      计算在出现该情感 label 的 train 文本中 Diff 中每个单词出现的次数；
    }
```

《1》没优化：$p(x_k|e_i)=\frac{nw_{e_i}(x_k)}{nw_{e_i}}$

《2》 优化：$\hat{P}(x_k|\omega_i)=\frac{|D_{i,x_k}|+\alpha}{|D_i|+\alpha c_k}$ （经过试验，a=0.00001 效果最好）

```
  }
}
```

```
/*作用：计算每种情感在 validation/test 每行文本的前提下出现的概率，然后选择概率最大的 label 作为预测的情感 label*/
void get_result(){
    for(遍历 validation/test 每一行文本){
        将该 label 下文本中每个单词出现的概率相乘，再乘以该情感出现的概率得到该文本前提下为该 label 的概率[ P（di | labelj）];
        在六个 P（di | labelj）中找出最大的概率的 label 作为该行文本的预测 label；
    }
}
```

（4）NB 回归：

NB 回归的代码前部分基本和 KNN 回归类似，这里不再重复，只写出不同点。

① NB 回归中没算 tf_idf 矩阵；

NB 回归中 tf 矩阵没优化和 KNN 回归的 tf 矩阵；优化：$\mathrm{tf}_{i,j}=\frac{n_{i,j}}{\sum_k n_{k,j}}$ 改为

$$tf_{i,j} = \frac{n_{i,j} + a}{\sum_k n_{k,j} + ac}$$

C 为全部不重复单词数，即 Diff 中的单词个数。

② /*计算 validation/test 每行文本中每种情感系数*/

```
void get_result(){
    for(遍历 validation/test 的每行文本){
        找出 validation/tes 该行文本出现的单词在 Diff 的位置 Diff_pos；
        for( 计算在该文本前提下 6 个 label 出现的概率){
            for（遍历 train 的每行文本）{
                将该行文本对应的 tf 矩阵中 Diff_pos 位置的数值相乘，再乘以该 label 的概率。
            }
            将上面 for 循环算出来的概率全部加起来获得在该文本前提下该 label 出现的概率。
        }
    }
}
```

3. 关键代码截图（带注释）

（1）KNN 分类

总体思路：

读取 csv 文件 train_set 和 validation_set，将里面的文本内容切割成单词分别放入向量 train 和 validation 中，label 分别放入 train_label 和 validation_label 中。在切割单词时，每切割出一个单词就判断一下该单词是否在 Diff_label 单词集中，如果不在，就把该单词放入 Diff_label 中。然后遍历 train 和 validation 每行文本的单词，在它们中寻找 Diff_label 中的单词从而获得 one-hot 和 bridge 矩阵。one-hot 记录该单词是否出现在该文本，出现记为 1，否则为 0；bridge 记录该单词在该文本中出现几次。然后通过 bridge 矩阵来获得 tf 矩阵，tf = (单词出现次数)/（文本单词总数）。bridge 每一行的总和记为文章单词总数，将 bridge 每一行的数据分别除以该行的总和即可得到 tf 矩阵。接下来计算 train 的 idf 向量，idf = log2(文本总数)/(出现该单词的文本数+1)。然后根据公式 tfidf[i][j] = tf[i][j]*idf[i]获得 tf_idf 矩阵。最后计算 validation 每一行文本和 train 的每一行文本的距离，可利用 one-hot 矩阵、tf 矩阵或 tf_idf 矩阵根据不同的距离度量方式（曼哈顿距离，欧氏距离、余弦相似度等）来计算。选出距离最近的 K 个 train 文本对应的 labels，然后在这些 labels 中找出出现次数最多（或权重最大）的 label 作为 validation 某行文本的预测 label。

使用的向量说明：

```
vector<string> train_label; //存放训练集的标签（情绪）
vector<string> Diff; //存放不同的单词
vector<vector<string> > train;   //存放训练集train的单词
vector<vector<double> > one_hot; //one-hot矩阵
vector<vector<string> > validation; //存放验证集validation
vector<string> validation_label; //验证集的label
vector<string> result_label;//存放通过模型得出的验证集的结果标签
vector<vector<double> >bridge; //记录训练集和验证集每个单词的出现次数
vector<vector<double> >tf; //训练集和验证集的tf矩阵
vector<vector<double> > tf_idf; //训练集和验证集的tf矩阵
```

《1》读取 csv 文件中的文本内容，并且将每一行文本的 words 切割成单词。

① 将文件打开：

ios::in 供读，文件不存在则创建(ifstream 默认的打开方式)。

```
ifstream ReadFile;
string temp; //将读取的每行文本放到temp中
ReadFile.open(filename,ios::in);
```

② 读取文本的每一行文本，并且将文本放到字符串 temp 中：

csv 文件读取出来每行内容的形式：“str1,str2,str3”,即每格的内容以，隔开。

```
while(getline(ReadFile,temp)){  //每一行中每一格的内容以，分开
```

③ 每一行文本“，”前面是 words，后面是 label，所以先找到“，”的位置，将逗号后面的 label 放入 train_emotion 中，将逗号前面的 words 切割出来。

```
//切割出label
string s1=",";
int pos,size=temp.size();
pos=temp.find(s1,0);//找到，的位置，逗号之后的文本即为label
string emotion=temp.substr(pos+1,size);
train_label.push_back(emotion);
temp.erase(pos,size);//切割掉label，剩下为words
```

④ 将剩下的 words 切割成一个个单词放入 train 中。遍历 words，以空格为标记切割成一个个单词。

```
//将一行文本切割成单词
vector<string> v;
string pattern=" ";//单词以空格分开
temp+=pattern;
int size=temp.size(),pos;
for(int i=0;i<size;i++){
    pos=temp.find(pattern,i);
    if(pos<size){
        string s=temp.substr(i,pos-i);//切割出来的单词
        v.push_back(s);
```

然后在 Diff(单词集：里面存放不同的单词)中寻找切割出来的单词，如果 Diff 中没有该单词，就把单词放入 Diff 中。

```
vector<string>::iterator it;
//在Diff(单词集）中寻找这个单词
it=find(Diff.begin(),Diff.end(),s);
if(it==Diff.end()){
    //如果Diff中没有该单词，则将该单词放入Diff中
    Diff.push_back(s);
}
```

⑤ 以上部分是获取 train 训练集的步骤，获取 validation 验证集的步骤相同。

《2》获得 one-hot 和 bridge 矩阵：

bridge 矩阵和 one-hot 矩阵类似，只是 one-hot 记录文本中是否出现该单词，而 bridge 是记录该单词的次数。按照题意应该获取是 validation 的每一行文本分别和 train 中的文本形成的 one-hot 矩阵和 bridge 矩阵，但是为了方便，还是存放在同一个向量里（one-hot 和 bridge），之后的 tf 和 tf_idf 矩阵也是这样。

① one-hot 矩阵和 bridge 矩阵的行数=train 的行数+validation 的行数。

```
/*作用：获得one-hot和bridge矩阵
one-hot:记录单词是否出现
bridge：记录单词出现的次数 */
void get_one_bri()
{
    int column=Diff.size(),row1=train.size(),row2=validation.size(),row=row1+row2,k;//one_hot矩阵的行数和列数
```

② 遍历 train 和 validation。在它们中寻找 Diff（单词集），如果没找到记录为 0，如果找到了，对于 one-hot 记录为 1，bridge 则将原来的数值++；

```
if(i<row1){ //0~（row1-1）行:train训练集的部分
    vector<string>::iterator it;
    //在每行文本的单词中找Diff中的单词
    it=find(train[i].begin(),train[i].end(),Diff[j]);
    if(it==train[i].end()){ //找不到记录为0
        v.push_back(0);
        v1.push_back(0);
    }else{      //找到记录为1
        v.push_back(1);
        bool first=true; //记录是否是第一次找到
        //在该行文本中能够找到该单词，则把该单词删除，然后继续寻找该单词，直到找不到为止
        while(it!=train[i].end()){
            it=train[i].erase(it);
            if(first==true){
                v1.push_back(1); //第一次找到，记录为1
            }else{
                v1[j]++;//不是第一次找到，把原来记录的数字++
            }
            it=find(train[i].begin(),train[i].end(),Diff[j]);
            first=false;
        }
    }
```

《3》 获得 tf 矩阵：

tf 矩阵可以直接通过 bridge 矩阵来获得。计算 bridge 每一行数据的总和就可以得到每一行文本的单词总数。然后再用 bridge 每一行的数据分别除以该行的单词总数即可得到 tf 矩阵。

TF（Term Frequency）：向量的 **每一个值** 标志对应的词语 **出现的次数** **归一化后的频率**。

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

```
for(int i=0;i<size_of_row;i++){
    int Denominator=0;
    for(int k=0;k<size_of_column;k++){
        Denominator+=bridge[i][k];//计算tf公式的分母(文本的单词总数)
    }
    for(int j=0;j<size_of_column;j++){
        if(bridge[i][j]>0){
            num=1.0*bridge[i][j]/Denominator;
            v.push_back(num);
        }
        else if(bridge[i][j]==0) v.push_back(0);
    }
    tf.push_back(v);
```

《4》获取 tf_idf 矩阵：

① 在获取 tf_idf 之前，我们首先得获得 idf 向量。在获取 idf 向量时，我们得注意一点：我们获取的是 train 文本的 idf 向量，所以 D=train 的文本数。t 也是 train 训练集中出现该单词的文章总数。

IDF：逆向文件频率；假设总共有 |D| 篇文章，$|\{j:t_i \in d_j\}|$ 表示出现了该单词的文章总数，IDF值的计算公式如下：

$\mathrm{idf_i} = \log \frac{|D|}{|\{j:t_i \in d_j\}|}$

$idf_i = log \frac{|D|}{1+|\{j:t_i \in d_j\}|}$

```
int D=train.size();
vector<double> idf;
int size_of_row=D;
int size_of_column=Diff.size();
for(int j=0;j<size_of_column;j++){
    int t=0;
    for(int i=0;i<size_of_row;i++){
        t+=one_hot[i][j];
        //通过计算将one_hot训练集部分 每列的总数
        //算出可知一个单词出现在多少篇文本中
    }
    t+=1;
    double res = log2(1.0*D/t);//D-- 文本总数 ， res是idf的数据
    idf.push_back(res);
}
```

② 按照公式 $\mathrm{tfidf_{i,j}} = \mathrm{tf_{i,j}} \times \mathrm{idf_i}$ 计算 tf_idf 矩阵的值。

```
for(int i=0;i<row;i++){
    for(int j=0;j<size_of_column;j++){
        double tfidf =1.0*idf[j]*tf[i][j];//tf_idf = idf*tf
        v.push_back(tfidf);
    }
    tf_idf.push_back(v);
    v.clear();
}
```

《5》用不同的度量距离的方法计算距离，然后选出距离目标文本最近的 K 个文本的 label，然后在这些 label 中选出出现次数最多的一个 label。

① 定义结构体 dis_pos，用于记录 train 的第 pos 行文本与目标文本的距离 dis 。

```
typedef struct{
    double dis;
    int pos;
}dis_pos;
```

② 遍历 validation 的每一行，分别计算 validation 的每一行文本与 train 所有文本的距离。

```
dis_pos res;
int row1=train.size(),row2=validation.size(),row=row1+row2;
for(int i=row1;i<row;i++){ //遍历验证集two矩阵每一行
    vector<dis_pos> distance;  //记录验证集中的一行和训练集的每一行到的距离
    vector<string> k_label; //记录选取的训练集中的K个最接近的文本的label
    for(int j=0;j<row1;j++){ //遍历训练集one-hot每一行，算出验证集的一行和训练集每一行的距离
```

③ 根据度量方法的不同，按照不同的计算方法分别计算距离。

欧式距离[n=2]：

```
double dis=0;
if(dis_way=="Euclidean"){
    for(int g=0;g<Diff.size();g++){
        dis+=(two[i][g]-two[j][g])*(two[i][g]-two[j][g]);
    }
    dis=sqrt(dis);// 算欧氏距离
```

曼哈顿距离（城市距离）[n=1]：

```
}else if(dis_way=="CityBlock"){
    for(int g=0;g<Diff.size();g++){
        if(two[i][g]>two[j][g])
        dis+=two[i][g]-two[j][g];
        else dis+=two[j][g]-two[i][g];
    }
```

[n=3]：

```
}else if(dis_way=="3-th"){
    for(int g=0;g<Diff.size();g++){
        double d;
        if(two[i][g]>two[j][g]) d=two[i][g]-two[j][g];
        else d=two[j][g]-two[i][g];
        dis+=pow(d,3);
    }
    dis=pow(dis,1.0/3);
```

余弦相似度：

```
}else if(dis_way=="cos"){
    double ab,a,b;
    for(int g=0;g<Diff.size();g++){
        ab+=1.0*two[i][g]*two[j][g];
        a+=1.0*two[i][g]*two[i][g];
        b+=1.0*two[j][g]*two[j][g];
    }
    dis = 1.0*ab/(1.0*sqrt(a)*sqrt(b));
}
res.dis=dis;
res.pos=j;
```

④ 如果是 distance 存放的是余弦相似度，那么就把 distance 按从大到小。因为余弦值作为衡量两个个体间差异的大小的度量，余弦值为正且值越大，表示两个文本差距越小。如果存放的是其他距离，那么就把 distance 按从小到大排序。

```
if(dis_way=="cos") sort(distance.begin(),distance.end(),comparationbig);
else sort(distance.begin(),distance.end(),comparationsmall);

bool comparationsmall(dis_pos d1,dis_pos d2)
{
    return d1.dis<d2.dis;
}

bool comparationbig(dis_pos d1,dis_pos d2)
{
    return d1.dis>d2.dis;
}
```

⑤ 直接提取 distance 的前 K 位，distance 记录了 pos 和 dis[距离 train 第 pos 个文本的距离为 dis]。通过 distance 的 pos 获得 K 个 label。

```
for(int m=0;m<k;m++){
    k_label.push_back(train_label[distance[m].pos]);
}
```

⑥ 设置情感标签向量 diff_label；

```
vector<string> diff_label;
diff_label.push_back("anger");
diff_label.push_back("disgust");
diff_label.push_back("fear");
diff_label.push_back("joy");
diff_label.push_back("sad");
diff_label.push_back("surprise");
int diff_size=6;
```

⑦ **初始方法**;数组 num 记录 diff_label 对应的 pos 和 count[记录该情感标签出现的次数]。

```
num[m]=m;
num[m].count++;
```

将出现次数从大到小排列，然后选择出现次数最多的 label 作为预测的 label。

优化一： 初始方法；设置情感标签向量 diff_label；数组 num 记录 K_label 对应的距离较小的 pos 和 count[记录该情感标签出现的次数]。

```
/*相同label时记录dis最小的位置*/
if(num[m].pos!=-1){
    if(distance[num[m].pos].dis>distance[q].dis) num[m].pos=q;
}else{
    num[m].pos = q;
}
num[m].count++;
```

将出现次数从大到小排列，然后选择出现次数最多的 label 作为预测的 label。

优化二：设置情感标签向量 diff_label；数组 num 记录 diff_label 对应的 pos 和 count[记录该情感标签的权重]。

```
num[m].count=0;
for(int q=0;q<k;q++){
    if(diff_label[m]==k_label[q]){
        if(dis_way=="cos"){
            num[m].pos=m;
            //权重
            num[m].count += 1.0*((distance[q].dis)*(distance[q].dis));
        }else{
            num[m].pos=m;
            //权重
            num[m].count += 1.0/((distance[q].dis)*(distance[q].dis));
        }
    }
}
```

将权重从大到小排列，然后选择权重最大的 label 作为预测的 label。

```
sort(num,num+diff_size,comparation2); //按权重从大到小排列
result_label.push_back(diff_label[num[0].pos]); //选择权重最大的label
```

《6》将 validation 的预测 label 和 validation_label 中真实的 label 进行对比，求得准确率。

```
double correct()
{
    int count=0,size=validation.size();
    for(int i=0;i<size;i++){
        if(result_label[i]==validation_label[i]) count++;
    }
    double accuracy = (1.0*count)/size;
    return accuracy;
}
```

《7》将 result_label 写入 csv 文件中：

```
ofstream outFile;
outFile.open("F:\\学习资料\\大三上\\人工智能\\实验\\zhouwuxiawuKNN\\result.csv",ios::out);
int row_of_res=result_label.size();
for(int r=0;r<row_of_res;r++){
    outFile << result_label[r] << "\n";
}
outFile.close();
return 0;
```

（2）KNN 回归

总体思路：

KNN 回归从切割单词到计算距离和 KNN 分类基本是一样的，只是 KNN 分类是每行文本只有一个 label，而 KNN 回归是每行文本都有六个情感系数。在计算好 validation 每一行文本与 train 的所有文本的距离后，将距离进行归一化。

Feature scaling	$X' = \frac{X - X_{min}}{X_{max} - X_{min}}$	X_{min} is the min value and X_{max} is the max value

然后将归一化的距离进行排序（余弦相似度按照从大到小排序，其余距离按照从小到大排序），选取 K 个距离近的文本，根据

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

来计算文本的每个情绪系数。(注意：余弦相似度是用相乘，其他距离时用距离的倒数。)最后将每个文本的 6 个情感系数除以 6 个情感系数的总和从而让六个情感系数相加等于 1。

《1》和 KNN 分类相似的切割字符串、获取 one-hot、tf、tf_idf 矩阵、计算距离的代码就不展示了，参考 KNN 分类即可。我们直接从计算好了距离之后开始。

① 将距离进行归一化：

找出距离中的最大值和最小值，然后 距离 =（距离-最小值）/（最大值-最小值）。

```
/*线性函数归一化*/
int row_of_dis=distance.size();
double min=100000,max=0;
for(int m=0;m< row_of_dis;m++){
    if(distance[m].dis>max) max=distance[m].dis;
    if(distance[m].dis<min) min=distance[m].dis;
}
for(int m=0;m<row_of_dis;m++){
    distance[m].dis = 1.0*(distance[m].dis-min)/(max-min) ;
}
```

② 对距离进行排序。（如果是余弦相似度则按从大到小排序，其他就按照从小到大排序）。

```
/*余弦相似度：余弦值为正且值越大，表示两个文本差距越小；为负代表差距大
 所有余弦相似度按从大到小排序 ，其他距离按从小到大排序 */
if(dis_way=="cos") sort(distance.begin(),distance.end(),comparationbig);
else sort(distance.begin(),distance.end(),comparationsmall);
```

③ 排好序后前面 K 个距离就是我们需要的，然后按照下面的公式计算每个情感系数。如果是余弦相似度则是乘以距离，其他是乘以距离的倒数。

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

```
int column_of_emotion=6;
vector<double> em;
/*余弦相似度：余弦值为正且值越大，表示两个文本差距越小；为负代表差距大
 所有余弦相似度越大 则 权重就应该越大，所以用* ；
 其他距离 距离越小，则权重应该越大，所以用倒数 */
if(dis_way=="cos"){
    for(int c=0;c<column_of_emotion;c++){
        double p=0;
        for(int r=0;r<k;r++){
            p+=1.0*train_emotion[distance[r].pos][c]*(distance[r].dis);
        }
        em.push_back(p);
    }
}else{
    for(int c=0;c<column_of_emotion;c++){
        double p=0;
        for(int r=0;r<k;r++){
            p+=1.0*train_emotion[distance[r].pos][c]/(distance[r].dis+1);
            //分母为距离+1,防止出现分母为0的情况
        }
        em.push_back(p);
    }
```

④ 然后让每个情感系数除以该测试样本情感系数的总和，从而让一个文本的情感系数总和为 1。

```
for(int r=0;r<row_of_res;r++){
    double all=0;
    for(int c=0;c<column_of_res;c++){
        all+=result_emotion[r][c];  //计算每个测试样本所有情绪的概率的总和
    }
    for(int h=0;h<column_of_res;h++){
        /*让每个情绪的概率除以该测试样本情绪的概率总和
          这样同一测试样本的各个情绪概率总和为1 */
        if(all!=0) result_emotion[r][h]/=all;
    }
}
```

（3）NB 分类

总体思路：NB 分类从切割单词到获得 NumOfWords(和 KNN 分类的 bridge 矩阵类似)和 KNN 分类基本是一样的，就不重复了。首先计算 train 中每个 label 出现的概率以及出现在 train 中的行数。每个 label 的概率=（train 中出现该 label 的文本数）/（trian 的总文本数）；优化后，每个 label 的概率=（train 中出现该 label 的文本数+1）/（trian 的总文本数+6）。接下来运用多项式模型，计算出在每种情感的前提下 Diff 中每个单词出现的概率

$p(x_k|e_i)=\frac{nw_{e_i}(x_k)}{nw_{e_i}}$，优化后，$p(x_k|e_i)=\frac{nw_{e_i}(x_k)+a}{nw_{e_i}+ca}p(e_i$ C:为出现该 label 的文本的所有不重复单词。最后计算每种情感在 validation/tese 每行文本的前提下出现的概率 [该概率=在该种情感的前提下该行文本中的每个单词出现的概率相乘，再乘以该情感 label 出现的概率。]，选择出现的概率最高的 label 作为该行文本的预测 label。

《1》 计算在 trian 中每个 label 出现的概率以及记录每个 label 在 train 中出现的位置：

① 计算在 train 中每个 label 出现的次数，以及记录每个 label 在 train 中的位置

```
//每种情感出现的次数
double accur_angry=0,accur_disgust=0,accur_fear=0,accur_joy=0,accur_sad=0,accur_surprise=0;
//记录每种情感label在train中出现的行数
vector<int> v0,v1,v2,v3,v4,v5;
int size_of_trainlabel=train_label.size();
for(int i=0;i<size_of_trainlabel;i++){
    if(train_label[i]==emotion[0]){
        accur_angry+=1;
        v0.push_back(i);
    }else if(train_label[i]==emotion[1]){
        accur_disgust+=1;
        v1.push_back(i);
    }else if(train_label[i]==emotion[2]){
        accur_fear+=1;
        v2.push_back(i);
    }else if(train_label[i]==emotion[3]){
        accur_joy+=1;
        v3.push_back(i);
    }else if(train_label[i]==emotion[4]){
        accur_sad+=1;
        v4.push_back(i);
    }else if(train_label[i]==emotion[5]){
        accur_surprise+=1;
        v5.push_back(i);
    }
}
```

② 用每个 label 出现的次数除以文本的总数即可得到每个 label 出现的概率

```
accur_angry=1.0*accur_angry/size_of_trainlabel;
p_label.push_back(accur_angry);
accur_disgust=1.0*accur_disgust/size_of_trainlabel;
p_label.push_back(accur_disgust);
accur_fear=1.0*accur_fear/size_of_trainlabel;
p_label.push_back(accur_fear);
accur_joy=1.0*accur_joy/size_of_trainlabel;
p_label.push_back(accur_joy);
accur_sad=1.0*accur_sad/size_of_trainlabel;
p_label.push_back(accur_sad);
accur_surprise=1.0*accur_surprise/size_of_trainlabel;
p_label.push_back(accur_surprise);
```

优化：分子为每个 label 出现的次数+1，分母为中的 label 数加上 label 的种类。

```
accur_angry=1.0*(accur_angry+1)/(size_of_trainlabel+6);
p_label.push_back(accur_angry);
accur_disgust=1.0*(accur_disgust+1)/(size_of_trainlabel+6);
p_label.push_back(accur_disgust);
accur_fear=1.0*(accur_fear+1)/(size_of_trainlabel+6);
p_label.push_back(accur_fear);
accur_joy=1.0*(accur_joy+1)/(size_of_trainlabel+6);
p_label.push_back(accur_joy);
accur_sad=1.0*(accur_sad+1)/(size_of_trainlabel+6);
p_label.push_back(accur_sad);
accur_surprise=1.0*(accur_surprise+1)/(size_of_trainlabel+6);
p_label.push_back(accur_surprise);
```

《2》 运用多项式模型，计算出 在每种情感的前提下 Diff 中的每个单词出现的概率：

① 统计 出现该情感 label 的 train 文本的所有单词数 以及 所有不重复单词数：

```
all_words=0; //出现该label的文本的所有单词数
norepet_words=0; //出现该label的文本的所有不重复单词数
for(int k=0;k<label_row;k++){
    for(int g=0;g<size_diff;g++){
        if(NumOfWords[pos[i][k]][g]!=0){
            all_words+=NumOfWords[pos[i][k]][g];
            norepet_words+=1;
        }
    }
}
```

② 统计出 Diff 中的每个单词在该情感 label 前提下出现的次数：

```
for(int m=0;m<size_diff;m++){
    nwex=0; //记录Diff中每个单词在该情感label前提下出现的次数
    for(int j=0;j<label_row;j++){
        nwex+=NumOfWords[pos[i][j]][m];
    }
```

③ 通过公式 $p(x_k|e_i)=\frac{nw_{e_i}(x_k)}{nw_{e_i}}$ 计算出在每种情感的前提下 Diff 中的每个单词出现的概率。

```
nwex=1.0*nwex/all_words;
```

优化： 使用平滑： $\hat{P}(x_k|\omega_i)=\frac{|D_{i,x_k}|+\alpha}{|D_i|+\alpha c_k}$

```
double a=0.00001;
nwex=1.0*(nwex+1*a)/(all_words+norepet_words*a);
```

《3》 计算每种情感在 validation/test 每行文本的前提下出现的概率，选出概率最高的 label 作为预测 label。

① 将该文本中的每个单词在某个 label 前提下的概率相乘，再乘以该 label 出现的概率即可得到该文本的情感是这个 label 的概率。

```
possibility=p_label[i];
if(pos[i].size()==0) possibility=0;//train中不存在这种情感
else{
    for(int j=0;j<size_diff;j++){
        if(NumOfWords[m+size_train][j]!=0){
            possibility=1.0*possibility*p_words[i][j];
        }else{
            continue;
        }
    }
}
```

② 选出出现概率最高的 label 作为结果，即预测的 label。

```
int pos = (int) ( max_element(v.begin(),v.end())-v.begin());
result.push_back(emotion[pos]);
```

（4）NB 回归

总体思路：NB 回归和 KNN 回归的前部分相同（从分割单词到求 tf 矩阵），就不再重复。求得 tf 矩阵后，遍历 validation 的每一行文本，在 train 的每一行中将该文本出现的单词的 tf 矩阵中的数值（即 在该 train 文本 和 在该情感前提下出现该单词 一起发生的概率）相乘，再乘以情感系数，最后将 train 中每一行计算的结果相加，得到该行文本的情感系数。

《1》 优化：

运用平滑：$\hat{P}(x_k|\omega_i)=\frac{|D_{i,x_k}|+\alpha}{|D_i|+\alpha c_k}$ C_k 是 Diff 中所有单词的个数。分母的 D：该行文本的单词总数，分子的 D：该单词在该行文本中出现的次数。

```
double Denominator=Diff.size()*a;
//  int Denominator=0;
for(int k=0;k<size_of_column;k++){
    Denominator+=bridge[i][k];//计算tf公式的分母(文本的单词总数)
}
for(int j=0;j<size_of_column;j++){
    if(bridge[i][j]>0){
//  num=1.0*bridge[i][j]/Denominator;
      num=1.0*(bridge[i][j]+1*a)/Denominator;
    v.push_back(num);
    }
    else if(bridge[i][j]==0){
    /*  v.push_back(0);*/
       num = 1.0*a/Denominator;
       v.push_back(num);
    }
}
```

《2》 计算 validation/test 每行文本的每个情感系数：

① 找出 validation/test 该行文本中的单词在 Diff 中的位置。

```
for(int j=0;j<size_diff;j++){ //找出validation一行文本出现的单词在Diff的位置
    if(one_hot[i+row_train][j]!=0) Diff_pos.push_back(j);
}
```

② 将 train 对应的 tf 矩阵在 Diss_pos 位置对应的值相乘，再乘以情感系数；然后将所有的值相加，即可得到该行文本的该 label 系数。

```
for(int b=0;b<num_emotion;b++){ //遍历六种情感
    train_p=0;
    for(int m=0;m<row_train;m++){  //遍历train的每一行
        row_p=1.0;
        for(int k=0;k<size_pos;k++){
            row_p=1.0*row_p*tf[m][Diff_pos[k]];
        }
        row_p=1.0*row_p*train_emotion[m][b];
        train_p+=row_p;
    }
    v.push_back(train_p);
}
```

③ 将每行文本所有情感系数相加，然后让每个情感系数乘以总和，使得一行文本的所有情感系数的和为 1。

```
double all_p=0;
for(int j=0;j<num_emotion;j++){
    all_p+=v[j];
}
if(all_p!=0){
    for(int j=0;j<num_emotion;j++){
        v[j]=1.0*v[j]/all_p;
    }
}
result_emotion.push_back(v);
```

4. 创新点&优化（如果有）

（1）KNN 分类：

① 最开始找预测结果是通过在选出的 K 个 label 中找出出现次数最多的 label。但是这 K 行 train 文本到 validation 的该行文本的距离时不同的，距离近的肯定更有参考价值，而这里却把 K 个文本的价值等同看待，所以可能误差会比较大。因此，我们使用了另一种方法：将不同距离的 label 对该行文本产生的影响赋予不同的权重，即每找到一个 label，就把该 label 的权重增加，增加的值= （1/距离）^{2} ，这样距离越近即距离的值越小增加的权重就越大。而对于余弦相似度是：增加的值= （距离）^{2} ，因为余弦相似度是值越大说明两个文本差距越小，所以增加的权重等于距离的平方，而不是距离倒数的平方。

```
if(dis_way=="cos"){
    num[m].pos=m;
    //权重
    num[m].count += 1.0*((distance[q].dis)*(distance[q].dis));
}else{
    num[m].pos=m;
    //权重
    num[m].count += 1.0/((distance[q].dis)*(distance[q].dis));
}
```

② 采用了不同的距离度量方式：余弦相似度。余弦值作为衡量两个个体间差异的大小的度量，为正且值越大，表示两个文本差距越小。

余弦相似度：

$\cos\left(\vec{A},\vec{B}\right)=\frac{\vec{A}\cdot\vec{B}}{\|\vec{A}\|\|\vec{B}\|}$，其中 $\vec{A}$ 和 $\vec{B}$ 表示 两个文本特征向量；

```
}else if(dis_way=="cos"){ //余弦相似度
    double ab=0,a=0,b=0;
    for(int g=0;g<Diff.size();g++){
        ab=ab+1.0*two[i][g]*two[j][g];
        a=a+1.0*two[i][g]*two[i][g];
        b=b+1.0*two[j][g]*two[j][g];
    }

    dis = 1.0*ab/(1.0*sqrt(a)*sqrt(b));
```

（2）KNN 回归：

① 在获得 validation 某一行和 train 所有行的距离后，将距离的值规范化，这样有助于放置具有较大初始值域的距离比具有较小初始值域的距离的权重过大。

```
/*线性函数归一化*/
int row_of_dis=distance.size();
double min=100000,max=0;
for(int m=0;m< row_of_dis;m++){
    if(distance[m].dis>max) max=distance[m].dis;
    if(distance[m].dis<min) min=distance[m].dis;
}
for(int m=0;m<row_of_dis;m++){
    distance[m].dis = 1.0*(distance[m].dis-min)/(max-min) ;
}
```

② 采用了不同的距离度量方式：余弦相似度。

（3）NB 分类：

① 计算每种情感的出现概率时运用了拉普拉斯平滑公式：

```
accur_angry=1.0*(accur_angry+1)/(size_of_trainlabel+6);
p_label.push_back(accur_angry);
accur_disgust=1.0*(accur_disgust+1)/(size_of_trainlabel+6);
p_label.push_back(accur_disgust);
accur_fear=1.0*(accur_fear+1)/(size_of_trainlabel+6);
p_label.push_back(accur_fear);
accur_joy=1.0*(accur_joy+1)/(size_of_trainlabel+6);
p_label.push_back(accur_joy);
accur_sad=1.0*(accur_sad+1)/(size_of_trainlabel+6);
p_label.push_back(accur_sad);
accur_surprise=1.0*(accur_surprise+1)/(size_of_trainlabel+6);
p_label.push_back(accur_surprise);
```

② 计算在某种情感的前提下某个单词出现的概率时运用了 Lidstone 平滑。

```
double a=0.00001;
nwex=1.0*(nwex+1*a)/(all_words+norepet_words*a);
```

（4）NB 回归：

① 计算 tf 矩阵时运用了 Lidstone 平滑：

```
double a=0.01;
for(int i=0;i<size_of_row;i++){
    double Denominator=Diff.size()*a;
//  int Denominator=0;
    for(int k=0;k<size_of_column;k++){
        Denominator+=bridge[i][k];//计算tf公式的分母(文本的单词总数)
    }
    for(int j=0;j<size_of_column;j++){
        if(bridge[i][j]>0){
//      num=1.0*bridge[i][j]/Denominator;
            num=1.0*(bridge[i][j]+1*a)/Denominator;
        v.push_back(num);
        }
        else if(bridge[i][j]==0){
        /*  v.push_back(0);*/
            num = 1.0*a/Denominator;
            v.push_back(num);
        }
    }
```

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）

（1）KNN 分类

<1> 用小数据测试自己的模型是否正确：

① train：

Words	label
I can not find test	sad
hello this is my girlfriend	sad
She is luhan's girlfriend	sad
I am not his girfriend	sad
I can not stand his girlfriend	anger

test：

Words	label
I can not stand I am not luhan's grilfriend	?

② tf 矩阵：

```
0.2 0.2 0.2 0.2 0.2 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0.2 0.2 0.2 0.2 0.2 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0.25 0 0.25 0.25 0.25 0 0 0 0 0
0.2 0 0 0.2 0 0 0 0 0 0 0 0 0 0.2 0.2 0.2 0 0
0.166667 0.166667 0.166667 0 0 0 0 0 0 0.166667 0 0 0 0.166667 0 0.166667 0
0.222222 0.111111 0.222222 0 0 0 0 0 0 0 0 0 0.111111 0.111111 0 0 0.111111 0.111111
```

③ tf—3-欧式距离：

```
ror=4  dis=0.323942  label=anger
ror=0  dis=0.371849  label=sad
ror=3  dis=0.371849  label=sad
```

〈2〉最优结果：

```
准确率=0.434084
k=9
dis way=CityBlock
```

（2）KNN 回归：

最优结果：

	anger	disgust	fear	joy	sad	surprise
r	0.402040662	0.349123	0.45123525	0.451016785	0.43761967	0.423690417
average	0.419120965					
evaluation	低度相关 666					

1	anger	disgust	fear	joy	sad	surprise
2	0.0896983	0.100955	0.227858	0.218501	0.124772	0.238216
3	0.0616184	0.0211718	0.251058	0.248634	0.197197	0.220321
4	0.0536897	0.0328258	0.230663	0.21343	0.150016	0.319376
5	0.0865584	0.0667209	0.231172	0.109189	0.374371	0.131989
6	0.0369429	0.0265863	0.127448	0.398402	0.181554	0.229066
7	0.25834	0.0725724	0.20682	0.0326029	0.263191	0.166474
8	0.173082	0.0759794	0.229444	0.118474	0.225687	0.177334
9	0.0559259	0.06372	0.106693	0.497441	0.0825895	0.193631
10	0.0537391	0.0868339	0.202278	0.183107	0.20041	0.273631
11	0.0513095	0.0645575	0.176968	0.290307	0.155628	0.261229
12	0.0535228	0.0074005	0.151451	0.461543	0.0560925	0.269991
13	0.028188	0.0242119	0.255671	0.129785	0.340364	0.221779
14	0.157074	0.135692	0.11145	0.0671052	0.288428	0.24025
15	0.0264717	0.0208472	0.0721253	0.0428806	0.557183	0.280492
16	0.104862	0.0644184	0.164589	0.380457	0.144125	0.141549
17	0.0809696	0.0338758	0.069752	0.44709	0.110816	0.257497
18	0.199182	0.146739	0.110061	0.297695	0.0797217	0.166602
19	0.0154913	0.0118861	0.147635	0.493286	0.124064	0.207637
20	0	0	0.313369	0.0718491	0.154062	0.460721
21	0.0796876	0.144579	0.169039	0.220806	0.178184	0.207704
22	0.175324	0.158741	0.230192	0.113995	0.233959	0.0877884
23	0.0441031	0.05848	0.173136	0.274479	0.161183	0.288619
24	0.0277842	0.0282021	0.135719	0.405387	0.143998	0.258909
25	0.112028	0.0579414	0.18456	0.0885928	0.329094	0.227784
26	0.0720502	0.0205009	0.205808	0.281762	0.199768	0.220111

（3）NB 分类：

最优解：

```
正确率=0.385852
```

（4）NB 回归：

最优解：

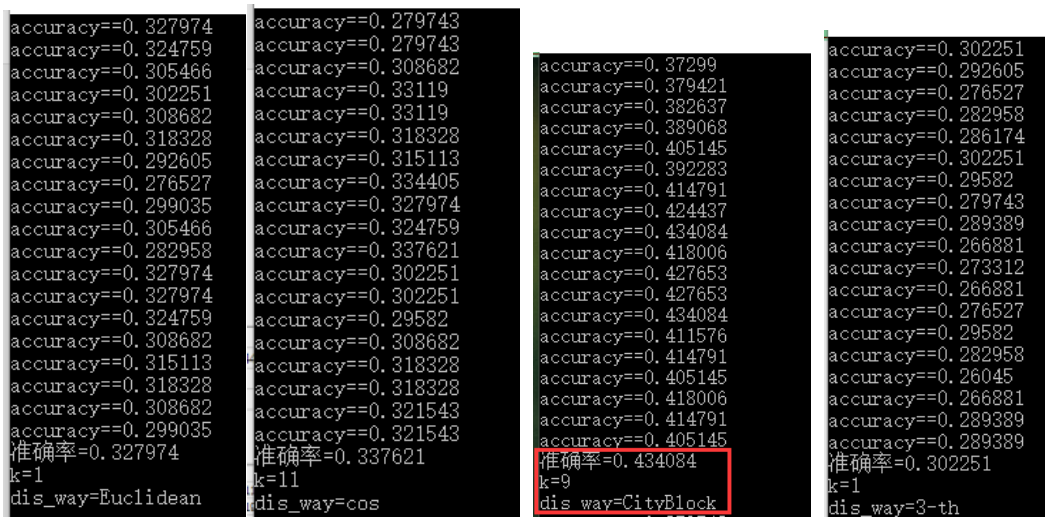
	anger	disgust	fear	joy	sad	surprise
r	0.318781373	0.238780103	0.37377527	0.419576893	0.40437542	0.387525816
average	0.357135813					
evaluation	低度相关 666					

id	anger	disgust	fear	joy	sad	surprise
1	0.097395	0.086463	0.209676	0.231382	0.18899	0.186092
2	0.045867	0.043769	0.191068	0.234655	0.137684	0.346958
3	0.070364	0.044281	0.235744	0.209459	0.184902	0.25525
4	0.077976	0.045318	0.187289	0.223677	0.275484	0.190257
5	0.05478	0.031264	0.172016	0.312921	0.207739	0.22128
6	0.108608	0.057426	0.121438	0.28362	0.192231	0.236676
7	0.158615	0.077963	0.322293	0.054294	0.270526	0.116309
8	0.08699	0.08699	0.043499	0.521645	0.108689	0.152188
9	0.010788	0.047331	0.243485	0.174168	0.03015	0.49408
10	0.076555	0.077887	0.236751	0.116096	0.282089	0.210622
11	0.037463	0.0022	0.107113	0.320183	0.189938	0.343103

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）

（1）KNN 分类：

不同的 K 值，不同的距离度量方式的不同结果：



（2）KNN 回归：

《1》 one-hot 矩阵，曼哈顿距离:

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.122119874	0.123124238	0.244011309	0.210610473	0.273875501	0.26536877
average	0.206518361					
evaluation	极弱相关 加油哦小辣鸡					

② k=6

	anger	disgust	fear	joy	sad	surprise
r	0.094744111	0.162928959	0.248479975	0.194989669	0.191453814	0.23612884
average	0.188120895					
evaluation	极弱相关 加油哦小辣鸡					

③ k=10

	anger	disgust	fear	joy	sad	surprise
r	0.083410095	0.111595564	0.261624235	0.217201219	0.217611873	0.194870519
average	0.181052251					
evaluation	极弱相关 加油哦小辣鸡					

《2》 one-hot 矩阵，欧式距离：

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.126500609	0.124016476	0.24488571	0.213169168	0.276785419	0.26742331
average	0.208796782					
evaluation	极弱相关 加油哦小辣鸡					

② k=6

	anger	disgust	fear	joy	sad	surprise
r	0.099554929	0.164562819	0.250885908	0.199061214	0.196786687	0.23997501
average	0.191804428					
evaluation	极弱相关 加油哦小辣鸡					

③ k=10

	anger	disgust	fear	joy	sad	surprise
r	0.08808405	0.113799289	0.263823536	0.221066262	0.222061071	0.199258546
average	0.184682126					
evaluation	极弱相关 加油哦小辣鸡					

《3》one-hot 矩阵，余弦相似度：

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.329556476	0.292671792	0.354958863	0.37348578	0.386138977	0.336665048
average	0.345579489					
evaluation	低度相关 666					

② k=6:

	anger	disgust	fear	joy	sad	surprise
r	0.281927069	0.271552489	0.359913591	0.404957381	0.399547425	0.341637825
average	0.343255963					
evaluation	低度相关 666					

③ k=11:

	anger	disgust	fear	joy	sad	surprise
r	0.293893708	0.295324381	0.36187347	0.381473495	0.37089425	0.3083736
average	0.335305491					
evaluation	低度相关 666					

《4》 tf 矩阵，曼哈顿距离

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.28285003	0.291981754	0.32899627	0.38608093	0.295140181	0.315931632
average	0.316830133					
evaluation	低度相关 666					

② k=10

	anger	disgust	fear	joy	sad	surprise
r	0.267812535	0.250465373	0.327818157	0.361957999	0.33709408	0.300935017
average	0.307680527					
evaluation	低度相关 666					

③ k=15

	anger	disgust	fear	joy	sad	surprise
r	0.28262598	0.232140391	0.326636733	0.367447372	0.34047828	0.276267342
average	0.304266016					
evaluation	低度相关 666					

《5》 tf 矩阵，欧式距离

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.263029882	0.137982797	0.226905943	0.295057571	0.350130721	0.314525801
average	0.264605452					
evaluation	极弱相关 加油哦小辣鸡					

② k=10

	anger	disgust	fear	joy	sad	surprise
r	0.175253309	0.17844673	0.267319119	0.321875873	0.311636498	0.275554941
average	0.255014412					
evaluation	极弱相关 加油哦小辣鸡					

③ k=15

	anger	disgust	fear	joy	sad	surprise
r	0.186925957	0.129352699	0.260894236	0.298929723	0.294948008	0.256545294
average	0.237932653					
evaluation	极弱相关 加油哦小辣鸡					

《6》 tf 矩阵，余弦相似度

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.337430379	0.269137607	0.325707527	0.338353347	0.388535346	0.343275027
average	0.333739872					
evaluation	低度相关 666					

② k=10

	anger	disgust	fear	joy	sad	surprise
r	0.325819418	0.294578849	0.352173948	0.3914703	0.359359861	0.314665559
average	0.339677989					
evaluation	低度相关 666					

③ k=15

	anger	disgust	fear	joy	sad	surprise
r	0.324339726	0.327070755	0.356053675	0.36540118	0.334338549	0.305825237
average	0.335504854					
evaluation	低度相关 666					

《7》 tf_idf 矩阵，曼哈顿距离

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.281369042	0.180639485	0.263073234	0.397348033	0.472510629	0.302528038
average	0.316244743					
evaluation	低度相关 666					

② k=10

	anger	disgust	fear	joy	sad	surprise
r	0.241541634	0.234050875	0.26115877	0.35214738	0.39087518	0.325242131
average	0.300835995					
evaluation	低度相关 666					

③ k=15

	anger	disgust	fear	joy	sad	surprise
r	0.236924083	0.270341935	0.294549836	0.392584948	0.372622128	0.313692801
average	0.313452622					
evaluation	低度相关 666					

《8》 tf_idf 矩阵，欧式距离

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.18743925	0.129394241	0.249062926	0.334421229	0.372629358	0.282516336
average	0.25924389					
evaluation	极弱相关 加油哦小辣鸡					

② k=10

	anger	disgust	fear	joy	sad	surprise
r	0.181779809	0.007907486	0.146943193	0.246747226	0.382671264	0.273833779
average	0.206647126					
evaluation	极弱相关 加油哦小辣鸡					

③ k=15

	anger	disgust	fear	joy	sad	surprise
r	0.152754009	0.149243171	0.151086433	0.232259905	0.362967614	0.294547651
average	0.223809797					
evaluation	极弱相关 加油哦小辣鸡					

《9》 tf_idf 矩阵，余弦相似度

① k=3

	anger	disgust	fear	joy	sad	surprise
r	0.360571814	0.292932021	0.389935533	0.407359463	0.396840507	0.40658258
average	0.375703653					
evaluation	低度相关 666					

② k=11

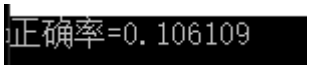
	anger	disgust	fear	joy	sad	surprise
r	0.386684683	0.347962365	0.450446686	0.443331447	0.43604351	0.417544638
average	0.413668888					
evaluation	低度相关 666					

③ k=15

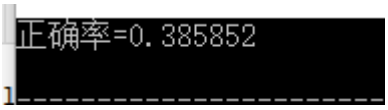
	anger	disgust	fear	joy	sad	surprise
r	0.362830091	0.359534498	0.430686979	0.436427345	0.418088135	0.40792395
average	0.402581833					
evaluation	低度相关 666					

（3）NB 分类：

未优化：



优化：



（4）NB 回归：

未优化：

	anger	disgust	fear	joy	sad	surprise
r	0.212477641	0.125579137	0.044792392	0.176761208	0.116761986	0.155301798
average	0.138612361					
evaluation	极弱相关 加油哦小辣鸡					

优化：

	anger	disgust	fear	joy	sad	surprise
r	0.318781373	0.238780103	0.37377527	0.419576893	0.40437542	0.387525816
average	0.357135813					
evaluation	低度相关 666					

四、 思考题

1. 为什么是倒数？

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：

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

答：因为距离近的训练数据会更有参考价值，所以距离越近的训练数据，权重就应该越大，因此取距离的倒数作为权重。

2. 在矩阵稀疏程度不同的时候，曼哈顿距离和欧式距离这两者表现有什么区别，为什么？

答：在矩阵系数程度越高时，曼哈顿距离变化比较大，而欧式距离变化比较小。因为（a+b）> $\sqrt{a^2+b^2}$ 。

3.伯努利模型和多项式模型分别有什么优缺点？

答：伯努利模型的优点：

伯努利模型中，对于一个样本来说，其特征用的是全局的特征，而且每个特征的取值是布尔值，所以当出现多个测试样本属性不平衡时，不会对该样本特征的估计产生太大的影响。

缺点：

伯努利模型中，对于每个特征的取值都是布尔值，忽略了每个特征的权重。

多项式模型的优点：

多项式模型中，对于每个特征的取值不是布尔值，而是特征出现的次数，即关注了每个特征的权重。

缺点： 多项式模型中，对于一个样本来说，其特征相当于用的是测试样本的特征的叠加，所以当有多个测试样本不平衡时（例如有多个测试样本中某个单词出现特别多次），可能会导致正确率很低。

4.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怎么处理？

答：如果测试集中出现了一个之前全词典中都没有出现过的词就运用 Listone 平滑，

$\hat{P}(x_k|\omega_i)=\frac{|D_{i,x_k}|+\alpha}{|D_i|+\alpha c_k}$ a 取 0~1 之间（不能为 0，不能为 1）。