



## 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 | 1511     | 专业（方向） | 软件工程（移动信息） |
| 学号   | 15352237 | 姓名     | 刘顺宇        |

## 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

#### 1.1. K 近邻：

1.1.1. 简述：KNN 是通过测量不同特征值之间的距离进行分类。它的思路是：如果一个样本在特征空间中的  $k$  个最相似(即特征空间中最邻近)的样本中的大多数属于某一个类别，则该样本也属于这个类别。KNN 算法中，所选择的邻居都是已经正确分类的对象。该方法在定类决策上只依据最邻近的一个或者几个样本的类别来决定待分样本所属的类别。

1.1.2. 影响因素： $k$  的取值、距离计算公式、文本矩阵形式、文本矩阵是否归一化处理

1.1.3. 分类思想：多数投票原则

1.1.4. 回归思想：计算距离，再取前  $k$  个距离作为权重乘以情感概率

#### 1.2. 朴素贝叶斯

1.2.1. 简述：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。朴素贝叶斯是一种简单但是非常强大的线性分类器。它之所以称为朴素，是因为它假设特征之间是相互独立的，但是在现实生活中，这种假设基本上是不成立的。那么即使是在假设不成立的条件下，它依然表现的很好，尤其是在小规模样本的情况下。但是，如果每个特征之间有很强的关联性和非线性的分类问题会导致朴素贝叶斯模型有很差的分类效果。

1.2.2. 假设：特征之间是相互独立的，随机变量具有独立性。

1.2.3. 定理：后验概率 = (条件概率 \* 先验概率) / 现象概率。

1.2.4. 影响因素：概率模型、拉普拉斯平滑、拉普拉斯平滑的系数

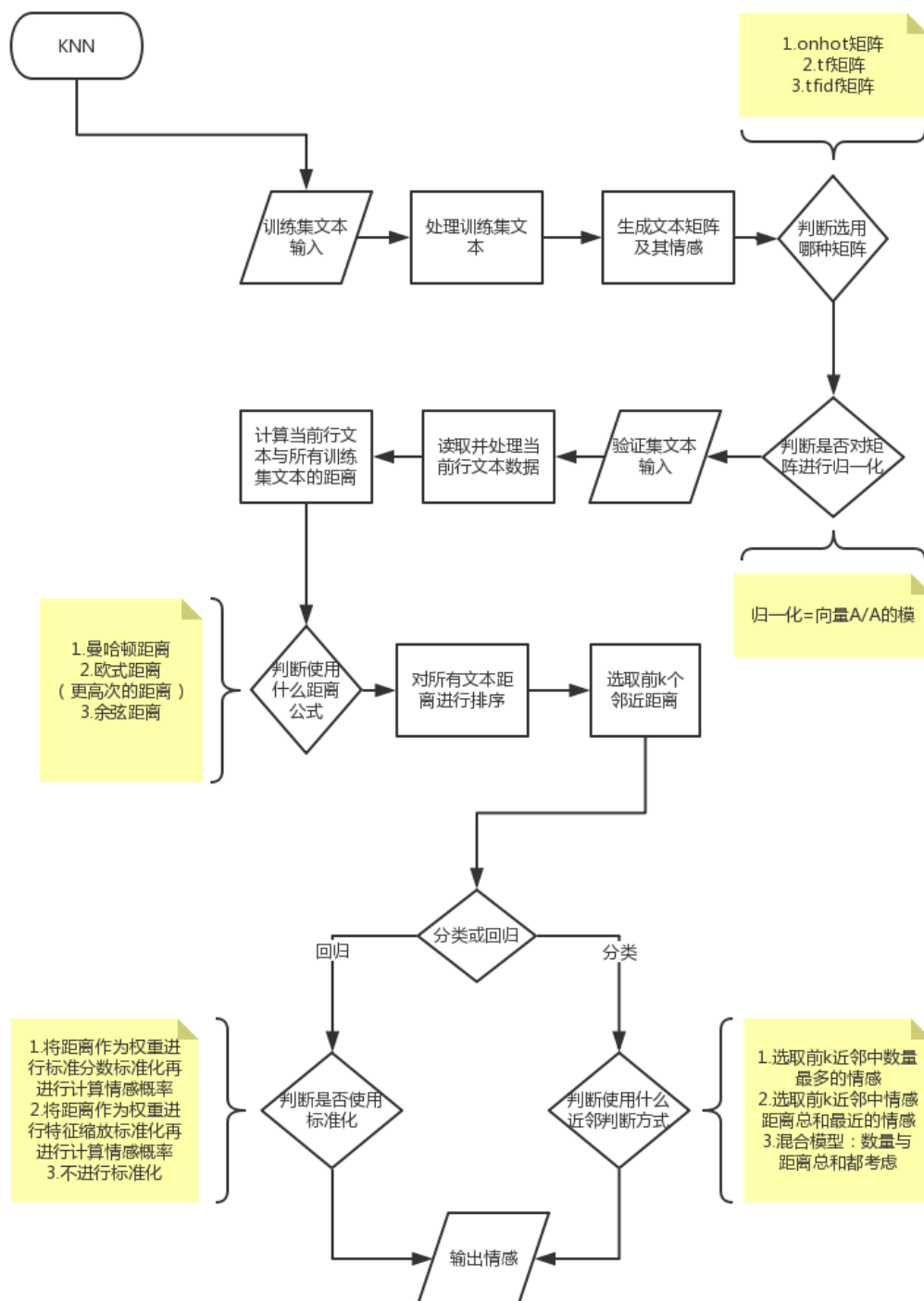
1.2.5. 分类思想：使用混合模型（伯努利与多项式模型混合使用）

1.2.6. 回归思想：计算所有句子中对于测试出现的词的概率情感之和

## 2. 伪代码流程图

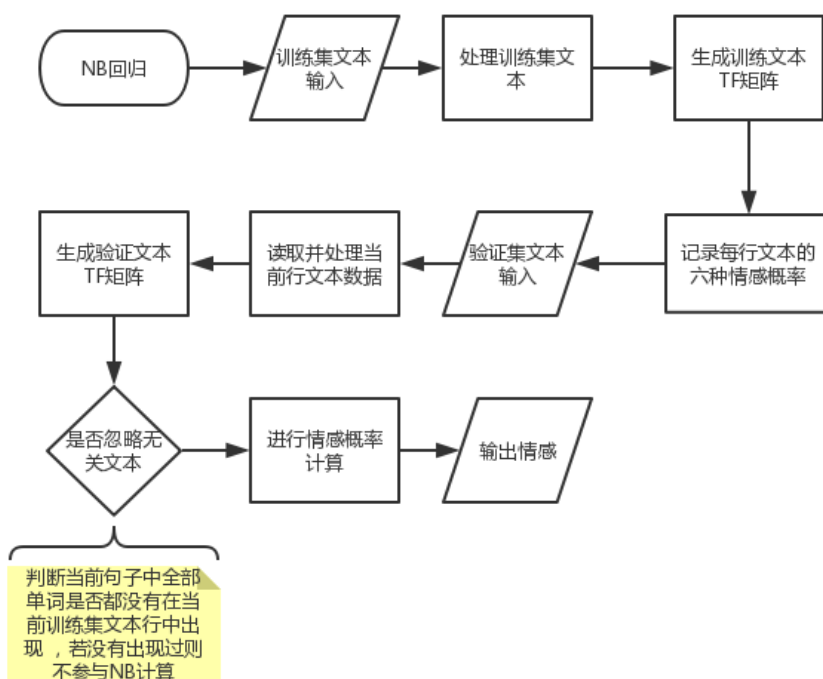
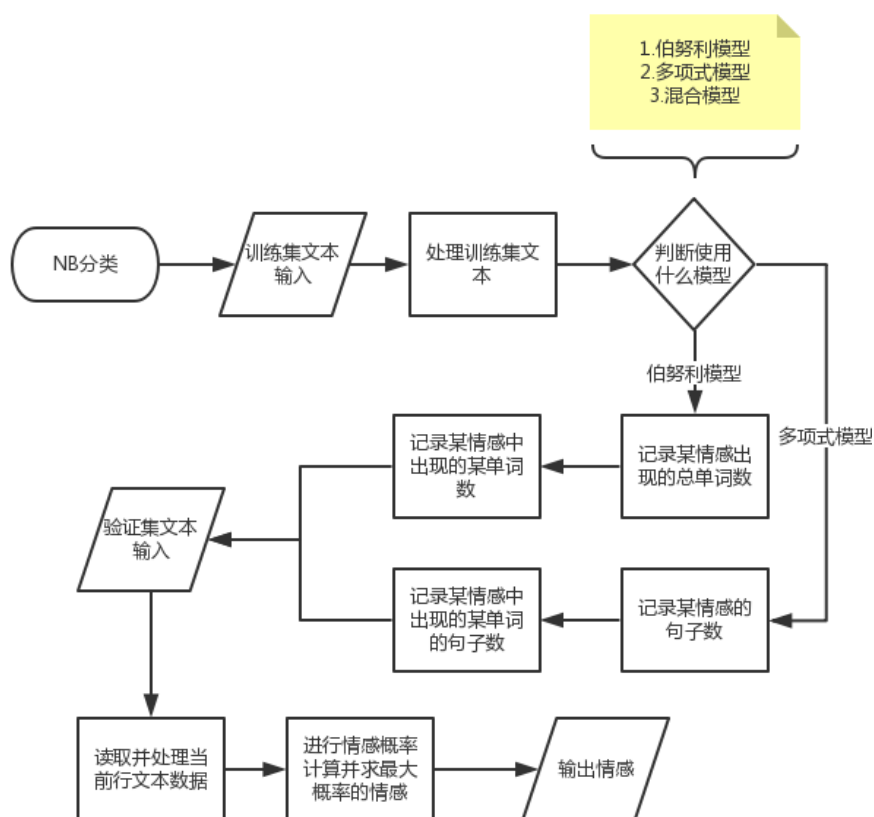
### 2.1. KNN 算法分类与回归代码流程图

首先读取训练集文本生成文本矩阵，然后将验证集的文本与全部训练集文本进行距离计算，并使用相应算法将前  $k$  个近邻文本的情感或情感概率作为验证文本的情感或情感概率的判断因素。





## 2.2. NB 算法分类与回归代码流程图





### 3. 关键代码截图（重复代码不再贴图解释，具体代码可看 cpp 文件，有详尽注释）

#### 3.1. KNN 分类

##### 3.1.1. TFIDF 矩阵定义及归一化

首先计算文本的 TFIDF 矩阵，然后对 TFIDF 矩阵进行向量除模归一化处理。

```
//输出 TFIDE 矩阵
void show_TFIDE_matrix(){
    int cnt = 0;
    for(int i = 0; i < rn; i++){
        double tem_Normalized = 0; //记录向量的模
        int tem_cnt = cnt;
        for(int j = 0; j < cn; j++){
            if(i == data[cnt].row && j == data[cnt].col){
                //计算逆向文件频率再计算 TFIDE 矩阵
                double TF_tem = double(data[cnt].value) / double(row_word_cnt[i]);
                double TFIDE_tem = TF_tem * (log(double(rn) / (1.0 + double(col_word_cnt[j]))));
                tem_Normalized += pow(TFIDE_tem, 2);
                TFIDE_matrix[i][j] = TFIDE_tem;
                cnt++;
            }else{
                TFIDE_matrix[i][j] = 0;
            }
        }
        tem_Normalized = sqrt(tem_Normalized);
        //对 TFIDF 矩阵进行向量除模归一化处理
        for(int j = 0; j < cn; j++){
            if(i == data[tem_cnt].row && j == data[tem_cnt].col){
                TFIDE_matrix[i][j] = TFIDE_matrix[i][j] / tem_Normalized;
                tem_cnt++;
            }else{
                TFIDE_matrix[i][j] = 0;
            }
        }
    }
}
```

##### 3.1.2. p 次距离与余弦距离

使用两种距离公式比较好坏，分别是万能的可输入 p 的 p 次距离公式，和余弦距离。

```
//使用TFIDF矩阵与任意p次距离
else if(method == 3){
    for(int j = 0; j < cnt_col; j++){
        words_dis_emotion[i].dis = words_dis_emotion[i].dis + pow(abs(test_TF[j] - TFIDE_matrix[i][j]), p);
    }
    words_dis_emotion[i].dis = pow(words_dis_emotion[i].dis, double(1.0/p));
}
//使用TFIDF矩阵与余弦距离
else if(method == 4){
    for(int j = 0; j < cnt_col; j++){
        words_dis_emotion[i].dis = words_dis_emotion[i].dis + (test_TF[j] * TFIDE_matrix[i][j]);
        cos_tem_test = cos_tem_test + pow(test_TF[j], 2);
        cos_tem_train = cos_tem_train + pow(TFIDE_matrix[i][j], 2);
    }
    cos_tem_test = sqrt(cos_tem_test);
    cos_tem_train = sqrt(cos_tem_train);
    words_dis_emotion[i].dis = words_dis_emotion[i].dis / (cos_tem_test * cos_tem_train);
}
}
```

##### 3.1.3. 分类 k 近邻情感判断：采用同时比较数量与比较总距离和的混合模型

使用 p 次距离时：采用先比较前 k 个中不同情感的数量，取数量最多的为输出情感，若是多个数量相同的情感，则再比较他们的距离和。



```
//若使用p次距离则距离越小越近
for(int i = 0; i < k; i++){
    knn_tem_num[words_dis_emotion[i].emotion]++; //记录每种情感的个数
    knn_tem_sum_dis[words_dis_emotion[i].emotion] += words_dis_emotion[i].dis; //记录每种情感的距离总和
    if(max_emotion != words_dis_emotion[i].emotion){
        //取最多个数的情感作为 max_emotion
        if(knn_tem_num[max_emotion] < knn_tem_num[words_dis_emotion[i].emotion]) max_emotion = words_dis_emotion[i].emotion;
        //若出现多个最多个数情感，则判断其距离和，近的为max_emotion
        else if (knn_tem_num[max_emotion] == knn_tem_num[words_dis_emotion[i].emotion]){
            if(knn_tem_sum_dis[max_emotion] > knn_tem_sum_dis[words_dis_emotion[i].emotion])
                max_emotion = words_dis_emotion[i].emotion;
        }
    }
}
```

使用余弦距离时：采用先比较距离和，取距离和最近的情感作为输出情感，但若是某一个情感数量大于  $k/2$  则直接将该情感作为输出情感。

```
//若使用余弦距离则余弦值越大越接近
for(int i = cnt_row - 1; i > cnt_row - k - 1; i--){
    knn_tem_num[words_dis_emotion[i].emotion]++; //记录每种情感的个数
    knn_tem_sum_dis[words_dis_emotion[i].emotion] += words_dis_emotion[i].dis; //记录每种情感的距离总和
    //取距离总和最近的情感作为 max_emotion
    if(knn_tem_sum_dis[max_emotion] < knn_tem_sum_dis[words_dis_emotion[i].emotion])
        max_emotion = words_dis_emotion[i].emotion;
}
```

```
//当前k个近邻中某情感个数大于一半时直接作为max_emotion
for(int i=1;i<=6;i++){
    if(knn_tem_num[i] > k/2) max_emotion = i;
}
```

## 3.2. KNN 回归

### 3.2.1. 回归 k 近邻情感判断

将前  $k$  个文本的距离作为权重乘以情感概率再求和，最后进行归一化处理得到输出情感概率，其中我们可以将距离权重做标准化处理。

```
//若使用p次距离则距离越小越近
if(method == 1 || method == 3){
    //如果距离为零则句子完全相同，直接将情感赋值
    if(words_dis_emotion[0].dis == 0){
        for(int i = 1; i <= 6; i++){
            knn_tem_pro[i] = words_dis_emotion[0].emotion[i];
            sum += knn_tem_pro[i];
        }
    }
    else{
        //计算各距离的倒数乘以情感概率
        for(int i = 1; i <= 6; i++){
            //记录最大距离与最小距离以使用标准化
            double max_dis = words_dis_emotion[0].dis;
            double min_dis = words_dis_emotion[0].dis;
            for(int j = 0; j < k; j++){
                if(words_dis_emotion[j].dis > max_dis) max_dis = words_dis_emotion[j].dis;
                if(words_dis_emotion[j].dis < min_dis) min_dis = words_dis_emotion[j].dis;
            }
            for(int j = 0; j < k; j++){
                knn_tem_pro[i] += words_dis_emotion[j].emotion[i] / words_dis_emotion[j].dis;
                //对距离权重进行标准化
                knn_tem_pro[i] += words_dis_emotion[j].emotion[i] / ((words_dis_emotion[j].dis - min_dis + 1) / (max_dis - min_dis + 1));
            }
            sum += knn_tem_pro[i];
        }
    }
}
```



```
// 如果句子中所有单词从未出现则平均附值情感概率
if(sum == 0){
    double tem = 0.166667;
    for(int i = 1; i <= 6; i++){
        outf << tem << ",";
    }
} else {
    // 对情感概率进行归一化后输出
    for(int i = 1; i <= 6; i++){
        outf << knn_tem_pro[i] / sum << ",";
    }
}
outf << endl;
```

### 3.3. NB 分类

#### 3.3.1. 两大概率模型定义

根据伯努利模型和多项式模型的定义构造两个结构体并在读取文本时统计需要用到的参数。

```
// 伯努利模型
struct Ber_emotion{
    int num; // 记录某情感的句子数
    int words[5000]; // 记录某情感中出现的某单词的句子数
    Ber_emotion(){
        num = 0;
        for(int i = 0; i < 5000; i++){
            words[i] = 0;
        }
    }
};
Ber_emotion bre_words_emotion[7];

// 多项式模型
struct Mul_emotion{
    int num; // 记录某情感出现的总单词数
    int words[5000]; // 记录某情感中出现的某单词数
    Mul_emotion(){
        num = 0;
        for(int i = 0; i < 5000; i++){
            words[i] = 0;
        }
    }
};
Mul_emotion mul_words_emotion[7];
```

#### 3.3.2. 混合模型

混合模型是伯努利模型和多项式模型的结合，重点在于它在统计训练集文本时考虑了词频（即使用多项式模型的思想），但是在对验证集的统计计算时不再考虑词频（即使用伯努利模型的思想）。

```
// 判断当前单词是否在当前行数据集第一次出现
it = word_pos.find(word_tem);
if(it == word_pos.end()){
    word_pos[word_tem] = Data_matrix.data.size(); // 记录新单词的序号
    // 将新单词位置放入矩阵中
    test_One_hot[word_num[word_tem]] = 1;
    test_TF[word_num[word_tem]] = 1;

    // 混合模型
    if(method == 3){
        for(int i = 1; i <= 6; i++){
            pro_emotion[i] *= double(mul_words_emotion[i].words[word_num[word_tem]] + a*1.0) / double(mul_words_emotion[i].num + a*train_cnt_col);
        }
    } else {
        test_TF[word_num[word_tem]]++; // 新单词所在TF矩阵值++
    }
}
```





### 3.3.3. 伯努利模型

统计训练集文本时不考虑词频，记录某情感中出现的某单词的句子数作为分子，记录某情感的句子数作为分母，在最后进行计算时会考虑验证文本的词频。

```
//伯努利模型
if(method == 1){
    for(int i = 1; i <= 6; i++){
        pro_emotion[i] += double(bre_words_emotion[i].words[word_num[word_tem]] + 1.0) / double(bre_words_emotion[i].num + 2.0);
    }
}
```

```
//伯努利模型
if(method == 1){
    for(int i = 1; i <= 6; i++){
        pro_emotion[i] = double(bre_words_emotion[i].num) / double(cnt_row);
    }
}
```

### 3.3.4. 多项式模型

统计训练集文本时考虑词频，记录某情感中出现的某单词数作为分子，记录某情感出现的总单词数作为分母，在最后进行计算时会考虑验证文本的词频。

```
//多项式模型
} else if(method == 2){
    for(int i = 1; i <= 6; i++){
        pro_emotion[i] *= double(mul_words_emotion[i].words[word_num[word_tem]] + a*1.0) / double(mul_words_emotion[i].num + a*train_cnt_col);
    }
}
```

### 3.3.5. 选择最大概率情感

```
int max_emotion = 1;
double max_pro = pro_emotion[1];
for(int i = 1; i <= 6; i++){
    if(max_pro < pro_emotion[i]){
        max_emotion = i;
        max_pro = pro_emotion[i];
    }
}
```

## 3.4. NB 回归

3.4.1. 定义 Pro\_emotion 记录每行文本的六种情感概率，在训练集文本读取时依次记录。

```
//定义 Pro_emotion 记录每行文本的六种情感概率
struct Pro_emotion{
    int num; //当前行号
    double emotion[7]; //六种情感概率
    Pro_emotion(){
        num = 0;
        for(int i = 1; i < 7; i++){
            emotion[i] = 0;
        }
    }
};
Pro_emotion words_pro_emotion[5000];
```



```
//记录每行文本的情感概率
string emotion_str = mydata_str.substr(pos + 1, mydata_str.length() - pos - 1); //截取情绪字符串
char* mydata_c_emotion = (char*)emotion_str.c_str();
const char* d_emotion = ","; //以,作截取词
char* word_emotion;
word_emotion = strtok(mydata_c_emotion, d_emotion); //截取有效单词
int i_emotion = 1;
while(word_emotion){
    string word_tem(word_emotion);
    words_pro_emotion[cnt_row].emotion[i_emotion] = atof(word_tem.c_str());
    i_emotion++;
    word_emotion = strtok(NULL, d_emotion);
}

//记录当前行行号
words_pro_emotion[cnt_row].num = cnt_row;
```

### 3.4.2. 进行 NB 概率运算

遍历所有训练集文本的某情感概率，如果某文本中不包含任何验证集中的单词，则不参与 NB 计算。

```
//进行NB 概率运算
double NB_emotion[7] = {0,};
double sum = 0;
//遍历六种情感
for(int i = 1; i <= 6; i++){
    //遍历所有训练集文本的某情感概率
    for(int j = 0; j < cnt_row; j++){
        double tem_emotion = 1;
        bool tem = false; //用于判断当前句子中全部单词是否都没有在当前训练集文本行中出现
        for(int m = 0; m < cnt_col; m++){
            if((test_One_hot[m] == 1) && (TF_matrix[j][m] != 0)){
                tem = true;
                continue;
            }
        }
        //如果出现则开始计算NB 情感概率
        if(tem == true){
            for(int m = 0; m < train_cnt_col; m++){
                if(test_One_hot[m] == 1){
                    tem_emotion *= ((TF_matrix[j][m] + 0.01) / (1.0 + 0.01 * train_cnt_col));
                }
            }
            NB_emotion[i] += tem_emotion * words_pro_emotion[j].emotion[i];
        }
    }
    sum += NB_emotion[i];
}
```

## 4. 创新点&优化（可能部分与关键代码截图重复，但为了阅读方便还是贴了过来）

### 4.1. KNN 创新：

#### 4.1.1. 对训练集与测试集使用 TFIDF 矩阵进行操作

```
for(int i = 0; i < cnt_col; i++){
    if(Data_matrix.col_word_cnt[i] == 0) test_TF[i] = 0;
    else{
        test_TF[i] = test_TF[i] / row_word_cnt * (log(double(cnt_row) / (1.0 + double(Data_matrix.col_word_cnt[i]))));
        tem_Normalized += pow(test_TF[i], 2);
    }
}
```





#### 4.1.2. 对训练集与测试集的 TFIDF 矩阵进行了向量除模归一化操作

```
tem_Normlized = sqrt(tem_Normlized);  
//对 TFIDF 矩阵进行向量除模的归一化操作  
for(int i = 0; i < cnt_col; i++){  
    test_TF[i] = test_TF[i] / tem_Normlized;  
}
```

#### 4.1.3. 针对分类：分类 k 近邻情感判断时采用同时比较数量与比较总距离和的混合模型 使用 p 次距离时：采用先比较前 k 个中不同情感的数量，取数量最多的为输出情感， 若是有多多个数量相同的情感，则再比较他们的距离和。

```
//若使用p次距离则距离越小越近  
for(int i = 0; i < k; i++){  
    knn_tem_num[words_dis_emotion[i].emotion]++; //记录每种情感的个数  
    knn_tem_sum_dis[words_dis_emotion[i].emotion] += words_dis_emotion[i].dis; //记录每种情感的距离总和  
    if(max_emotion != words_dis_emotion[i].emotion){  
        //取最多个数的情感作为 max_emotion  
        if(knn_tem_num[max_emotion] < knn_tem_num[words_dis_emotion[i].emotion]) max_emotion = words_dis_emotion[i].emotion;  
        //若出现多个最多个数情感，则判断其距离和，近的为max_emotion  
        else if (knn_tem_num[max_emotion] == knn_tem_num[words_dis_emotion[i].emotion]){  
            if(knn_tem_sum_dis[max_emotion] > knn_tem_sum_dis[words_dis_emotion[i].emotion])  
                max_emotion = words_dis_emotion[i].emotion;  
        }  
    }  
}
```

使用余弦距离时：采用先比较距离和，取距离和最近的情感作为输出情感，但若是某一个情感数量大于  $k/2$  则直接将该情感作为输出情感。

```
//若使用余弦距离则余弦值越大越接近  
for(int i = cnt_row - 1; i > cnt_row - k - 1; i--){  
    knn_tem_num[words_dis_emotion[i].emotion]++; //记录每种情感的个数  
    knn_tem_sum_dis[words_dis_emotion[i].emotion] += words_dis_emotion[i].dis; //记录每种情感的距离总和  
    //取距离总和最近的情感作为 max_emotion  
    if(knn_tem_sum_dis[max_emotion] < knn_tem_sum_dis[words_dis_emotion[i].emotion])  
        max_emotion = words_dis_emotion[i].emotion;  
}
```

```
//当前k个近邻中某情感个数大于一半时直接作为max_emotion  
for(int i=1;i<=6;i++){  
    if(knn_tem_num[i] > k/2) max_emotion = i;  
}
```



#### 4.1.4. 针对回归：对距离权重进行标准化操作，有两种办法：标准分数和特征缩放

```
//计算各距离乘以情感概率
for(int i = 1; i <= 6; i++){
    //记录最大距离与最小距离、方差与平均值以使用标准化
    double max_dis = words_dis_emotion[cnt_row - 1].dis;
    double min_dis = words_dis_emotion[cnt_row - 1].dis;
    double mean_dis = 0;
    double standard_deviation_dis = 0;
    for(int j = cnt_row - 1; j > cnt_row - k - 1; j--){
        if(words_dis_emotion[j].dis > max_dis) max_dis = words_dis_emotion[j].dis;
        if(words_dis_emotion[j].dis < min_dis) min_dis = words_dis_emotion[j].dis;
        mean_dis += words_dis_emotion[j].dis;
    }
    //计算平均值
    mean_dis = mean_dis / k;
    //计算方差
    for(int j = cnt_row - 1; j > cnt_row - k - 1; j--){
        standard_deviation_dis += pow((words_dis_emotion[j].dis - mean_dis), 2);
    }
    standard_deviation_dis = sqrt(standard_deviation_dis / k);

    for(int j = cnt_row - 1; j > cnt_row - k - 1; j--){
        knn_tem_pro[i] += words_dis_emotion[j].emotion[i] * words_dis_emotion[j].dis;
        //对距离权重进行两种不同的标准化
        knn_tem_pro[i] += words_dis_emotion[j].emotion[i] * (words_dis_emotion[j].dis - min_dis) / (max_dis - min_dis);
        knn_tem_pro[i] += words_dis_emotion[j].emotion[i] * (words_dis_emotion[j].dis - mean_dis + 1) / standard_deviation_dis;
    }
    sum += knn_tem_pro[i];
}
```

#### 4.1.5. 如果验证集句子曾在训练集中出现过，则直接赋值正确情感

```
//如果距离为零则句子完全相同，直接将情感赋值
if(words_dis_emotion[0].dis == 0){
    for(int i = 1; i <= 6; i++){
        knn_tem_pro[i] = words_dis_emotion[0].emotion[i];
        sum += knn_tem_pro[i];
    }
}
else{
```

## 4.2. NB 创新：

### 4.2.1. 针对分类：使用混合模型，混合模型是伯努利模型和多项式模型的结合，重点在于它在统计训练集文本时考虑了词频（即使用多项式模型的思想），但是在对验证集的统计计算时不再考虑词频（即使用伯努利模型的思想）。

```
//判断当前单词是否在当前行数据集第一次出现
it = word_pos.find(word_tem);
if(it == word_pos.end()){
    word_pos[word_tem] = Data_matrix.data.size(); //记录新单词的列号
    //将新单词位置放入矩阵中
    test_One_hot[word_num[word_tem]] = 1;
    test_TF[word_num[word_tem]] = 1;

    //混合模型
    if(method == 3){
        for(int i = 1; i <= 6; i++){
            pro_emotion[i] += double(mul_words_emotion[i].words[word_num[word_tem]] + a*1.0) / double(mul_words_emotion[i].num + a*train_cnt_col);
        }
    }
    else{
        test_TF[word_num[word_tem]]++; //新单词所在TF矩阵值++
    }
}
```



4.2.2. 针对回归：在进行 NB 概率运算时遍历所有训练集文本的某情感概率，如果某文本中不包含任何验证集中的单词，则不参与 NB 计算。

```
//进行NB概率运算
double NB_emotion[7] = {0,};
double sum = 0;
//遍历六种情感
for(int i = 1; i <= 6; i++){
    //遍历所有训练集文本的某情感概率
    for(int j = 0; j < cnt_row; j++){
        double tem_emotion = 1;
        bool tem=false; //用于判断当前句子中全部单词是否都没有在当前训练集文本行中出现
        for(int m = 0; m < cnt_col; m++){
            if((test_One_hot[m] == 1) && (TF_matrix[j][m]!=0)){
                tem = true;
                continue;
            }
        }
        //如果出现过则开始计算NB情感概率
        if(tem == true){
            for(int m = 0; m < train_cnt_col; m++){
                if(test_One_hot[m] == 1){
                    tem_emotion *= ((TF_matrix[j][m] + 0.01) / (1.0 + 0.01 * train_cnt_col));
                }
            }
            NB_emotion[i] += tem_emotion * words_pro_emotion[j].emotion[i];
        }
    }
    sum += NB_emotion[i];
}
```

### 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）

训练集为

| A                                | B     |
|----------------------------------|-------|
| Words                            | label |
| hello introduce these are TAs    | joy   |
| hello this is a test             | joy   |
| some TAs have no girlfriend      | sad   |
| some TAs have girlfriend         | joy   |
| TAs live happy                   | joy   |
| hello you all have no girlfriend | sad   |

测试集为

| A                              | B     |
|--------------------------------|-------|
| Words                          | label |
| some of you have no girlfirend | ?     |

TF 矩阵为

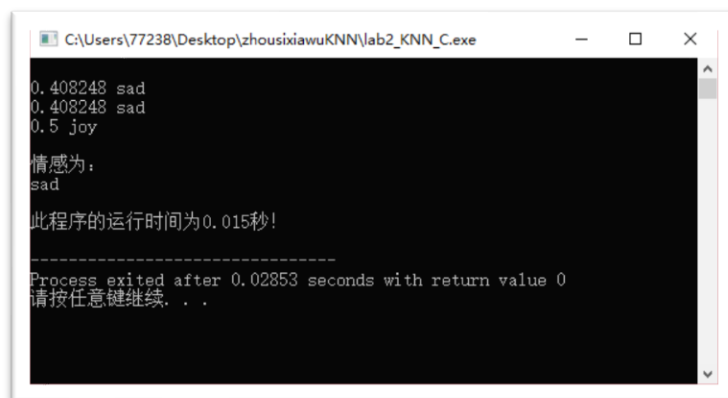
```

训练集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
0.2 0 0 0 0 0.2 0.2 0.2 0.2 0 0 0 0 0 0 0 0
0 0 0 0 0.2 0 0 0 0 0.2 0.2 0.2 0.2 0 0 0 0
0 0 0 0 0.25 0 0 0 0 0.25 0.25 0.25 0 0 0 0 0
0 0 0 0 0.333333 0 0 0 0 0 0 0 0.333333 0.333333 0 0 0
0.166667 0 0 0 0 0 0 0 0 0.166667 0.166667 0 0 0 0.166667 0.166667 0.166667

测试集TF矩阵:
0 0 0 0 0 0 0 0 0 0.166667 0.166667 0.166667 0 0 0 0.166667 0 0 0.166667 0.166667

测试集TF矩阵与训练集距离
0.408248 sad
0.408248 sad
0.5 joy
0.60553 joy
0.60553 joy
0.707107 joy
  
```

当 k 取 3 时，前三个距离中有两个 sad，所以程序输出情感 sad，正确



```

C:\Users\77238\Desktop\zhousixiawuKNN\lab2_KNN_C.exe
0.408248 sad
0.408248 sad
0.5 joy
情感为:
sad
此程序的运行时间为0.015秒!
-----
Process exited after 0.02853 seconds with return value 0
请按任意键继续. . .
  
```

由于回归模型与分类模型的思想一致，且使用的也同样是一套距离计算公式，所以不再重复展示实验结果

## 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）

### 2.1. 最优结果

| 问题     | 最优准确率       |
|--------|-------------|
| KNN 分类 | 48.5531%    |
| KNN 回归 | 42.3376636% |
| NB 分类  | 44.8718%    |
| NB 回归  | 38.4294246% |

## 2.2. 优化结果分析

由于优化较多，展示时使用控制变量法，即只改变某优化看结果，所以只有同一个优化分析中的结果有可比性，在不同的优化分析中由于没有很好控制变量，没有可比性。由于对同一种办法，如 KNN 或 NB，他们的分类与回归办法有相通之处，所以有些优化在分类中展示了便不再在回归中展示。

### 2.2.1. KNN 优化分类

| 优化前办法     | 优化前准确率   | 优化后办法    | 优化后准确率   |
|-----------|----------|----------|----------|
| Onehot 矩阵 | 42.7653% | TFIDF 矩阵 | 48.5531% |
| 矩阵无归一化    | 46.9483% | 矩阵归一化处理  | 48.5531% |
| 仅数量判断近邻   | 42.7653% | 先数量再距离判断 | 48.5531% |
| 仅数量判断近邻   | 42.7653% | 仅距离判断近邻  | 48.2315% |
| 仅距离判断近邻   | 48.2315% | 先距离再数量判断 | 48.5531% |

### 2.2.2. KNN 回归优化

| 优化前办法 | 优化前准确率      | 优化后办法   | 优化后准确率      |
|-------|-------------|---------|-------------|
| 无标准化  | 42.3376636% | 特征缩放标准化 | 41.4528211% |
| 无标准化  | 42.3376636% | 标准分数标准化 | 40.9495124% |

注：在 KNN 回归中的标准化优化办法并没有将我的准确率提高反而变低

### 2.2.3. NB 分类优化

| 优化前办法 | 优化前准确率   | 优化后办法 | 优化后准确率   |
|-------|----------|-------|----------|
| 伯努利模型 | 35.8974% | 混合模型  | 44.8718% |
| 多项式模型 | 44.5513% | 混合模型  | 44.8718% |

### 2.2.4. NB 回归优化

| 优化前办法  | 优化前准确率      | 优化后办法  | 优化后准确率      |
|--------|-------------|--------|-------------|
| 全部文本计算 | 36.4560147% | 去除无效文本 | 38.4294246% |

## 四、 思考题

1. 在处理 KNN 回归问题时，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。为什么是倒数呢？同一测试样本的各个情感概率总和应该为 1 如何处理？
  - 1.1. 因为在计算距离时如果采用曼哈顿距离或者欧式距离时，当两个文本距离的值越小时，证明两个文本越接近，相似度高，所以我们在计算概率时，距离作为权重乘以概率时就应该取距离的倒数，这样子才符合相似度越高，权重越大的原则，而且直接取倒数也符合单调性性质，在数学上是符合要求的。

但我们要注意当我们使用余弦距离的时候，就要直接使用余弦值作为权重，因为这时候余弦值越大，则代表两个文本越接近，相似度越高，权重越大。
  - 1.2. 当我们计算得到所有情感的概率后，为了使各个情感概率总和为一，可以对所有情感进行归一化处理，即每个情感概率除以情感概率的总和。
2. 曼哈顿距离和欧式距离，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？
  - 2.1. 矩阵的稀疏程度可以用范数向量来表示，向量的范数可以简单形象的理解为向量的长度，或者向量到零点的距离，或者相应的两个点之间的距离。我们定义  $L_p$  范数  $\|x\|_p$  为  $x$  向量各个元素绝对值  $p$  次方和的  $1/p$  次方
  - 2.2.  $L_1$  范数和  $L_2$  范数，用于机器学习的  $L_1$  正则化、 $L_2$  正则化。 $L_1$  正则化是指权值向量  $w$  中各个元素的绝对值之和，可以产生稀疏权值矩阵，即产生一个稀疏模型，可以用于特征选择； $L_2$  正则化是指权值向量  $w$  中各个元素的平方和然后再求平方根，可以防止模型过拟合（overfitting）；一定程度上， $L_1$  也可以防止过拟合。 $L_1$  正则化能增加稀疏性， $L_2$  正则化能防止过拟合。
  - 2.3. 采用哪种距离度量方法对最终结果有很大影响。例如，你的数据集有很多特征，但是如果任意一对个体之间的欧氏距离都相等，那么你就没法通过欧氏距离进行比较了！曼哈顿距离在某些情况下具有更高的稳定性，但是如果数据集中某些特征值很大，用曼哈顿距离的话，这些特征会掩盖其他特征间的邻近关系。最后，再来说说余弦距离，它适用于特征向量很多的情况，但是它丢弃了向量长度所包含的在某些场景下可能会很有用的一些信息。
3. 伯努利模型和多项式模型，这两个模型分别有什么优缺点？
  - 3.1. 伯努利模型：伯努利模型是一种简化的方法，直接将重复的词语都视为其只出现 1 次，这种方式更加简化与方便。当然它丢失了词频的信息，因此效果可能会差一些。
  - 3.2. 多项式模型：在多项式模型中我们考虑重复词语的情况，也就是说，重复的词语我们视为其出现多次，直接按条件独立假设的方式推导，然后多次出现的结果，出现在概率的指数/次方上
  - 3.3. 混合模型：在这次实验中我采用了混合模型，采取两家之长，在计算句子概率时，不考虑重复词语出现的次数，但是在统计计算词语的概率  $P(\text{"词语"}|S)$  时，却考虑重复词语的出现次数，这样的模型可以叫作混合模型。





#### 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如何解决？

4.1. 对于 KNN 问题，如果测试集中出现了一个之前全词典中没有出现过的词时，首先需要更新训练集的矩阵列表，增加多一全零列代表该新出现的单词。然后对于测试集的 TFIDF 矩阵来说，在计算 TF 和 IDF 时都要加上新单词进行计算，当然还有简单的做法时计算 TF 时加上新单词计算，但计算 IDF 时使用原测试集的 IDF。然后使用新的含有新单词列的矩阵进行距离运算。

4.2. 对于 NB 问题，我们可以使用拉普拉斯平滑来处理新出现的单词，这样就可以避免出现计算概率时乘以 0 的情况。

在处理分类问题的多项式模型时，使用拉普拉斯平滑使分子加一，分母加全文本中不重复单词个数。

在处理回归问题时，使用拉普拉斯平滑使分子加  $a$ ，分母加  $a \times$  全文本中不重复单词个数。