人工智能实验一

18340146 计算机科学与技术 宋渝杰

任务一：

将数据集“semeval.txt”的数据表示成 TF-IDF 矩阵，并保存为“学号_姓名拼音_TFIDF.txt”文件。其中词表的顺序要做一个排序。

算法原理：

该任务主要是文件数据处理任务，先分析源文件和任务，源文件有文本编号、总情感权重、各情感权重，最后是文本内容（单词），任务需要做的是单词的 TF-IDF 矩阵，因此除了单词之外的数据都是无效数据，弃之不用。

从文本中筛选出每行单词之后，对该行的单词做 split() 操作，提取出该行的单词，然后遍历单词：

- 该单词属于该行的 TF 分子+1
- 如果该行第一次出现该单词，则 IDF 分母+1

读取完所有行单词之后（过程当中同时记录有多少行数据，每行多少个单词），开始生成 TF-IDF 矩阵：

IDF 矩阵每个单词的数据为：

$$IDF_i = log\frac{|C|}{|C_i|+1}$$

其中 $|C|$ 为数据行数；$|C_i|$ 为第 i 个单词在多少行数据中出现，即上述计算出的 IDF 分母

TF 矩阵每个单词的数据为：

$$TF_{i,d} = \frac{n_{i,d}}{\sum_v n_{v,d}}$$

其中 $n_{i,d}$ 为第 i 个单词在第 d 行出现的次数，即上述计算出的 TF 分子；$\sum_v n_{v,d}$ 为第 d 行单词总数

最后，根据以下 TF-IDF 的计算公式即可得到 TF-IDF 矩阵的每个元素：

$$TF\_IDF_{i,d} = TF_{i,d} * IDF_i$$

计算出所有的数据之后，按照表格样式生成表格即可

伪代码/流程图：

一行行读取单词 -> 去除无用数据 -> split() 取出该行所有单词 -> 遍历单词，计算 TF 和 IDF（计算过程已在上文说明） -> 生成表格文件

代码展示：

报告只展示关键代码，全部代码请移步 lab1_1.cpp 文件

计算 TF-IDF 表格数据代码：

```
map<int,map<string,int>> TF;
map<string,int> IDF;
```

```
vector<int> v; // 存放每行单词数
while (getline(infile,s)) {
    c++; // 动态计算数据个数
    j = 0; // 动态计算每行单词个数
    int n = s.find_last_of("\t"); // 前面的文本编号和总情感权重是无效数据
    int x = n+1;
    while (s.find(' ',x) != string::npos) { // s.split()
        if (TF[c].count(s.substr(x,s.find(' ',x)-x)) == 0) // 某行数据尚未出现过这个
单词
            IDF[s.substr(x,s.find(' ',x)-x)]++; // IDF++
        TF[c][s.substr(x,s.find(' ',x)-x)]++; // TF++
        x = s.find(' ',x)+1;
        j++;
    }
    if (TF[c].count(s.substr(x)) == 0) // 最后一个单词
        IDF[s.substr(x)]++;
    TF[c][s.substr(x)]++;
    v.push_back(j+1);
}
```

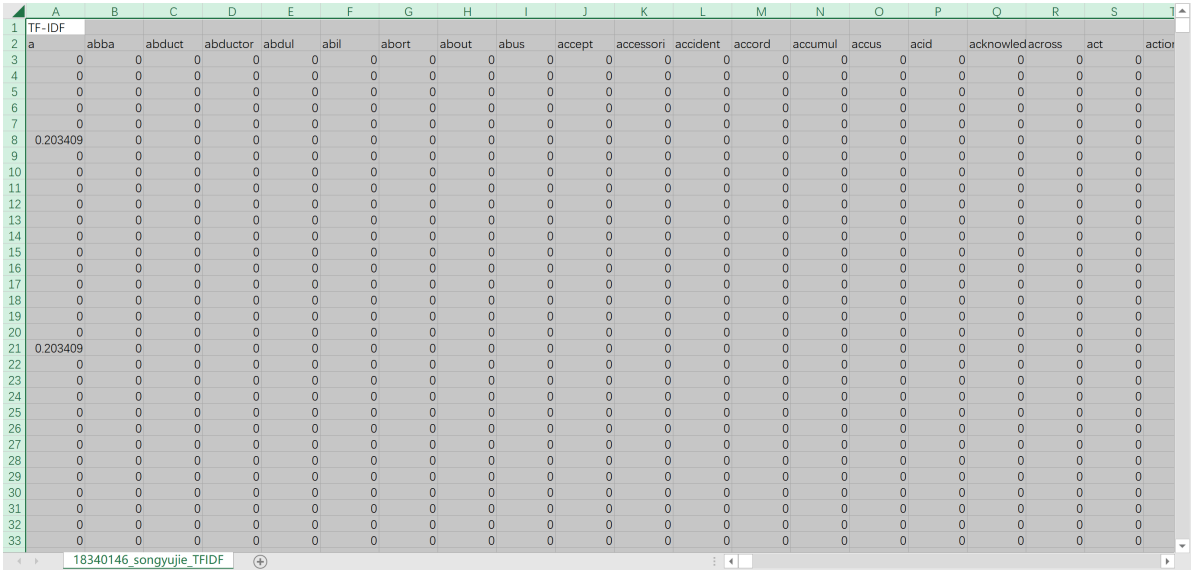
生成表格文件代码：

```
outfile.open("18340146_songyujie_TFIDF.txt");
outfile << "TF-IDF" << endl;
for (auto it = IDF.begin(); it != IDF.end(); ++it) { // 输出所有单词（map自带字典序升
序
    if (it != IDF.begin()) outfile << " ";
    outfile << it->first;
}
outfile << endl;
for (i=1; i<=c; i++) {
    for (auto it = IDF.begin(); it != IDF.end(); ++it) { // 输出每行数据的TF-IDF
        if (it != IDF.begin()) outfile << " ";
        outfile << (double)(TF[i].count(it->first) ? TF[i][it->first] : 0)/v[i-
1]*log10((double)c/(it->second+1));
    }
    outfile << endl;
}
```

实验结果以及分析：

运行 lab1_1.exe 之后，生成 18340146_songyujie_TFIDF.txt 结果文件，由于 txt 结果文件观感很不好，因此**额外生成 18340146_songyujie_TFIDF.csv 文件用于结果分析。**

用 Excel 打开 18340146_songyujie_TFIDF.csv 文件部分数据如下图：



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
1	TF-IDF																			
2	a	abba	abduct	abductor	abdul	abil	abort	about	abus	accept	accessori	accident	accord	accumul	accus	acid	acknowled	across	act	actio
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
8	0.203409	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
21	0.203409	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
26	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
27	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
29	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
30	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
31	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
32	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
33	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

由表格可以得出一些信息：总共有 1246 行数据、单词“a”在其中 74 行数据中出现过，其中在第 6 行数据出现…等等。

观察第 6 行数据，可以看出单词“a”出现了 1 次，该行有 6 个单词，符合 1/6*log_10(1246/(74+1)) = 0.203409 的结果。

```
6        all:62 anger:0 disgust:0 fear:0 joy:34 sad:0 surprise:28       pm havana deal a good experi
```

任务二：

使用KNN进行分类任务。数据文件为classification_dataset，其中train_set用于训练。validation_set是验证集，通过调节K值、不同距离度量等参数来筛选准确率最好的一组参数。在测试集test上应用该参数做预测，输出结果保存为“学号_姓名拼音_KNN_classification.csv”

算法原理：

该任务主要是文件数据处理+数据分类任务。根据 KNN 分类算法，我们需要先对训练数据生成其词频表示，之后读入其它数据时，计算其与其它所有训练数据的距离（与参数 p 有关）。

距离的计算公式：

$$L_p(x,y) = (\sum_{l=1}^{n} |x_l - y_l|^p)^{1/p}$$

其中 p 为参数，$L_p(x,y)$ 为数据 x 和 y 的距离，x_l 为数据 x 中单词 l 的个数，n 为单词总数

之后取 k 个距离最近的数据，统计其标签并取数量最大的那个标签，作为测试数据的标签。

伪代码/流程图：

读取训练数据：一行行读取数据 -> split()分割出所有单词和标签 -> 生成其词频表示，记录数据标签

读取验证数据：一行行读取数据 -> split()分割出所有单词和标签 -> 生成其词频表示 -> 根据其词频，计算它与所有训练数据的距离 -> 距离排序，统计前 k 个数据标签 -> 取标签最大值并与验证数据标签对比 -> 统计正确率

读取测试数据：一行行读取数据 -> 去除无用数据（问号）-> split()分割出所有单词和标签 -> 生成其词频表示 -> 根据其词频，计算它与所有训练数据的距离 -> 距离排序，统计前 k 个数据标签 -> 取标签最大值，成为该测试数据的标签

代码展示：

报告只展示关键代码，全部代码请移步 lab1_2_1.cpp 文件

计算距离代码：

```
unordered_map<int,unordered_map<string,int>> train; // 训练集的词频

double counting(int i,unordered_map<string,int> m) { // 计算测试数据m和训练数据i的距离
    double sum2 = 0;
    for (auto it = train[i].begin(); it != train[i].end(); ++it) // i有的单词
        sum2 += pow(abs(it->second-m[it->first]),p);
    for (auto it = m.begin(); it != m.end(); ++it) // m有的单词
        if (train[i].count(it->first) == 0) sum2 += pow(it->second,p); // 防止重复计算
    return pow(sum2,1.0/p);
}
```

训练代码：

```
string label[6]={"anger","disgust","fear","joy","sad","surprise"};
vector<int> v;
v.push_back(0);
// training...
infile.open("train_set.csv");
getline(infile,s);
while (getline(infile,s)) {
    n++; // 动态计算训练数据个数
    int l = s.find(','),x = 0;
    string s1 = s.substr(0,l),s2 = s.substr(l+1); // s1：句子，s2：标签
    // split句子
    while (s1.find(' ',x) != string::npos) {
        train[n][s1.substr(x,s1.find(' ',x)-x)]++;
        x = s1.find(' ',x)+1;
    }
    train[n][s1.substr(x)]++;
    for (i=0; i<6; i++) // 记录该训练数据标签
        if (s2 == label[i]) {
            v.push_back(i);
            break;
        }
}
infile.close();
```

验证代码：

```
// validating...
infile.open("validation_set.csv");
int correct=0,sum=0; // 正确数、总数
node N[n+1];
getline(infile,s); // 去除无用数据
while (getline(infile,s)) {
    sum++; // 动态计算验证集数据个数
    int l = s.find(','),x = 0;
    unordered_map<string,int> m;
```

```
    string s1 = s.substr(0,l),s2 = s.substr(l+1); // s1：句子，s2：标签
    // split句子
    while (s1.find(' ',x) != string::npos) {
        m[s1.substr(x,s1.find(' ',x)-x)]++;
        x = s1.find(' ',x)+1;
    }
    m[s1.substr(x)]++;
    // 初始化验证数据与训练数据距离
    for (i=1; i<=n; i++) {
        N[i].index = i;
        N[i].l = 0;
    }
    // 计算距离
    for (i=1; i<=n; i++) N[i].l = counting(i,m);
    sort(N+1,N+n+1,cmp); // 按距离升序排序
    // 多数投票
    int a[6] = {0,0,0,0,0,0},max=0,j;
    for (i=1; i<=k; i++) a[v[N[i].index]]++;
    for (i=0; i<6; i++)
        if (max < a[i]) {
            max = a[i];
            j = i;
        }
    // 判断验证是否正确
    if (label[j] == s2) correct++;
}
cout << "验证结果：" << correct << ":" << sum << "(" << correct*100.0/sum << "%)"
<< endl;
infile.close();
```

测试代码：

```
infile.open("test_set.csv");
outfile.open("18340146_songyujie_KNN_classification.csv");
getline(infile,s); // 去除无用数据
while (getline(infile,s)) {
    int l1 = s.find(',');
    int l2 = s.find(',',l1+1),x = 0;
    string s1 = s.substr(l1+1,l2-l1);// split句子
    unordered_map<string,int> m;
    while (s1.find(' ',x) != string::npos) {
        m[s1.substr(x,s1.find(' ',x)-x)]++;
        x = s1.find(' ',x)+1;
    }
    m[s1.substr(x)]++;
    // 初始化测试数据与训练数据距离
    for (i=1; i<=n; i++) {
        N[i].index = i;
        N[i].l = 0;
    }
    // 计算距离
    for (i=1; i<=n; i++) N[i].l = counting(i,m);
    sort(N+1,N+n+1,cmp); // 按距离升序排序
    // 多数投票
    int a[6] = {0,0,0,0,0,0},max=0,j;
    for (i=1; i<=k; i++) a[v[N[i].index]]++;
    for (i=0; i<6; i++)
```

```
            if (max < a[i]) {
                max = a[i];
                j = i;
            }
        outfile << s.substr(0,l1) << "," << label[j] << endl;
    }
    infile.close();
    outfile.close();
```

实验结果以及分析：

调参过程：我先对参数 k 调为最优值后，再调整参数 p 为最优值：

调 k 过程（暂定 p = 2)

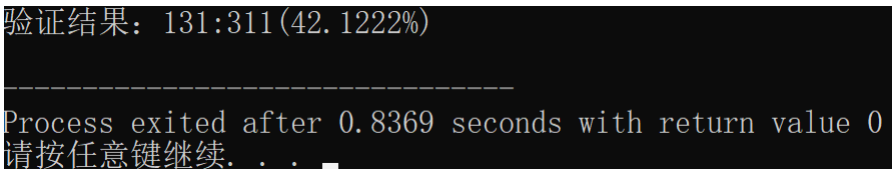
参数 k 取值	验证正确数	验证正确率
1	100	32.2%
5	96	30.8%
10	**130**	**41.8%**
15	119	38.3%
20	121	38.9%
25	119	38.3%
30	117	37.6%

由表知，参数 k 定为 10，下面继续调 p：

参数 p 取值	验证正确数	验证正确率
1	114	36.7%
2	130	41.8%
3	**131**	**42.1%**
4	123	39.5%
5	127	40.8%

因此，最终的参数为 **k = 10，p = 3**， 测试验证率为 **42.1%。**

运行 lab1_2_1.cpp 之后，输出验证结果：



个人认为准确率仅为 42.1%（已调参），有几个方面的原因：

- 算法中每个单词权重相同：这意味着一些具有强烈情感意识的单词和一些无关紧要的单词对最终标签的影响是相同的。举个例子，They got married 和 They got divorced 的情感是完全相反的，但是算法中无论 p 的取值，这两句话的距离都十分接近。

- 否定词的影响：距离算法显然完全无法处理否定词对情感的转变
- 训练数据的一些问题：首先是数据不够细致，数据之间关系不大，某些无明显情感意识的单词的 IDF 过小，以致测试数据某一个单词只能与很少的训练数据建立近距离关系。举个例子，训练集中“allow”单词只在 465 行出现一次，而这行数据的标签为 joy，这会导致测试数据如果存在该单词，会有较大概率被判定为 joy（而 allow 基本没有明显的情感意识）
- 另一个是标签过多数据量较少（或者说训练数据与标签种类不够匹配），而且 joy 训练数据占比较大，导致在基数上 joy 占优，验证数据集时发现大部分验证数据都被判定为 joy，这也是产生误差的原因之一。

因此个人认为这些原因会带来准确率并没有想象中高的原因。

创新点：

（我重写距离计算，自创相似度计算方式，修改投票方式，得到更优的结果需要和原算法结果数据作比较，因此创新点写在实验结果和分析之后）

我打算凭一些经验之类的东西，改写距离函数和投票方式，试图以一个新的方式寻求更高的准确率。因此自创一种**相似度**，公式如下：

$$simi(x,y)=\frac{\sum_{l=1}^{n}(F_{l,x}*F_{l,y}*IDF_l)}{len(x)*len(y)}$$

其中 $F_{l,x}$ 为单词 l 在数据 x 的词频（不进行归一化，因为测试发现归一化结果偏差），IDF_l 为单词 l 的逆向文档频率，n 为单词总数，$len(x)$ 为数据 x 单词种类数

按相似度取最高的 k 个训练数据，并修改原本的多数投票为**带权投票**，即相似度高的权重大，设权重为相似度的 q 次方（引入新的参数 q），最后带权投票后取最大值为验证/测试数据的标签

新的距离计算方法比之前的方法解决了一些问题：

- 通过加入 IDF 使得每个单词的权重不同
- 数据的长度被考虑在内
- 相似度越小的数据具有更低的权重（解决了 joy 基数大的问题）

具体修改的关键代码如下：（全部代码请移步 lab1_2_2.cpp 文件）

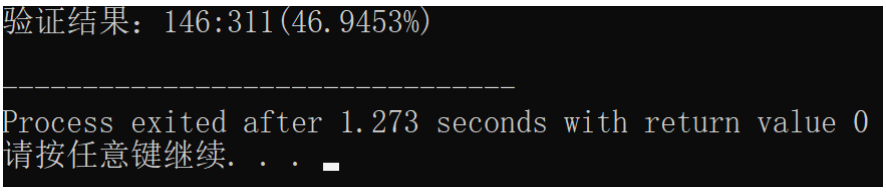
```
unordered_map<int,unordered_map<string,double>> train; // 训练集的词频
unordered_map<string,int> IDF; // 逆向文档频率

double counting(int i,unordered_map<string,int> m) { // 计算测试数据m和训练数据i的相似度
    double sum2 = 0;
    for (auto it = train[i].begin(); it != train[i].end(); ++it)
        sum2 += abs(it->second*log10((double)n/(IDF[it->first]+1))*m[it->first]);
    return sum2/train[i].size()/m.size();
}

int cmp(node a,node b) {
    return a.l>b.l; // 相似度降序排序
}

// 多数投票
double a[6] = {0,0,0,0,0,0},max=0;
for (i=1; i<=k; i++) a[v[N[i].index]] += pow(N[i].l,q); // 权重与相似度有关
```

新的相似度函数和投票方式的结果如下：通过调最优参 **k = 18，q = 2**（这里已无参数 p）时，验证集正确率上升至 **47%**，表现更优。



除了验证集正确率，还有一些值得说明的优势：

- 如果对训练集进行验证操作，正确率超过 99%（旧方法正确率仅为 50%），这意味着对于某些十分接近训练数据的测试数据，该方法具有极高正确率
- Debug 过程中发现，**原方法对于验证/测试数据会大概率判断成 joy（约为 80%），通过 joy 的大基数来达到 42% 的正确率**，而新方法判断成 joy 的概率大幅下降（即消除了盲目判断成 joy 的情况）
- 对于助教的小数据集，判断十分优秀（结果如下表），而原方法判断出 4 个 joy

原文	翻译	判断结果
cohn gunman **jailed** for years	科恩被判**入狱**数年	sad
bill would **strip** convicted legislators of pensions	该法案将**剥夺**被定罪的立法者的养老金	anger
amish **killings** school demolished	阿米什人**杀害**学校拆毁	sad
flooding **washes out** road to valdez alaska	洪水**冲毁**了通往阿拉斯加瓦尔迪兹的道路	sad
russian bird flu outbreak is **deadly** asian strain	俄罗斯爆发的禽流感是**致命的**亚洲毒株	fear

因此，最终的 18340146_songyujie_KNN_classification.csv 文件通过新算法计算产生

下图为生成的 18340146_songyujie_KNN_classification.csv 部分数据：

	A	B	C
1	1	senator carl krueger thinks ipods can kill you	sad
2	2	who is prince frederic von anhalt	disgust
3	3	prestige has magic touch	sad
4	4	study female seals picky about mates	joy
5	5	no e book for harry potter vii	joy
6	6	blair apologises over friendly fire inquest	fear
7	7	vegetables may boost brain power in older adults	surprise
8	8	afghan forces retake town that was overrun by taliban	sad
9	9	skip the showers male sweat turns women on study says	surprise
10	10	made in china irks some burberry shoppers	joy
11	11	britain to restrict immigrants from new eu members	joy
12	12	canadian breakthrough offers hope on autism	surprise
13	13	russia to strengthen its military muscle	fear
14	14	alzheimer s drugs offer no help study finds	joy

任务三：

使用KNN进行回归任务。数据文件为regression_dataset，其中train_set用于训练。validation_set是验证集，通过调节K值、不同距离度量等参数来筛选相关系数最好的一组参数。在测试集test上应用该参数做预测，输出结果保存为"学号_姓名拼音_KNN_regression.csv"。

算法原理：

该任务主要是文件数据处理+数据回归任务，总体操作与任务二相似：根据 KNN 分类算法，我们需要先对训练数据生成其词频表示，之后读入其它数据时，计算其与其它所有训练数据的距离（与参数 p 有关，距离的计算和任务二相同），取 k 个距离最近的数据，根据以下公式得到该数据各标签概率：

$$P_{i,x} = \frac{\sum_{j=1}^{k} \frac{P_{j,x}}{d_{i,j}}}{\sum_{j=1}^{k} \frac{1}{d_{i,j}}}$$

其中 $P_{i,x}$ 为测试数据 i 标签 x 的概率，$P_{j,x}$ 为与测试数据 i 距离第 j 近的训练数据的标签 x 的概率，$d_{i,j}$ 为测试数据 i 和与测试数据 i 距离第 j 近的训练数据的距离。

可以看出，这个公式和 pdf 课件的公式不同，因为课件的公式的每一种标签的概率的和不为 1，因此这里的概率计算乘上了一个**概率变化系数** $1/(\sum_{j=1}^{k} 1/d_{i,j})$，以保证每一种标签的概率的和为 1。

伪代码/流程图：

读取训练数据：一行行读取数据 -> split()分割出所有单词和标签 -> 生成其词频表示，记录数据各标签概率

读取验证数据：一行行读取数据 -> split()分割出所有单词和标签 -> 生成其词频表示 -> 根据其词频，计算它与所有训练数据的距离 -> 距离排序，统计前 k 个数据标签 -> 根据公式计算各标签概率 -> 读取并计算完所有数据之后，将计算出的各标签概率与验证数据各标签概率计算相关系数

读取测试数据：一行行读取数据 -> 去除无用数据（问号）-> split()分割出所有单词和标签 -> 生成其词频表示 -> 根据其词频，计算它与所有训练数据的距离 -> 距离排序，统计前 k 个数据标签 -> 根据公式计算各标签概率

代码展示：

报告只展示关键代码，全部代码请移步 lab1_3_1.cpp 文件

计算距离代码：

```
unordered_map<int,unordered_map<string,int>> train; // 训练集的词频

double counting(int i,unordered_map<string,int> m) { // 计算测试数据m和训练数据i的距离
    double sum2 = 0;
    for (auto it = train[i].begin(); it != train[i].end(); ++it)
        sum2 += pow(abs(it->second-m[it->first]),p);
    for (auto it = m.begin(); it != m.end(); ++it)
        if (train[i].count(it->first) == 0) sum2 += pow(it->second,p); // 防止重复计算
    return pow(sum2,1.0/p);
}
```

训练代码：

```
unordered_map<int,vector<double>> v;
```

```
// training...
infile.open("train_set.csv");
getline(infile,s);
while (getline(infile,s)) {
    n++; // 动态计算训练数据个数
    int l = s.find(','),x = 0;
    string s1 = s.substr(0,l),s2 = s.substr(l+1); // s1：句子，s2：概率
    // split句子
    while (s1.find(' ',x) != string::npos) {
        train[n][s1.substr(x,s1.find(' ',x)-x)]++; // 词频+1
        x = s1.find(' ',x)+1;
    }
    train[n][s1.substr(x)]++;
    x = 0;
    while (s2.find(',',x) != string::npos) {
        v[n].push_back(strtod(s2.substr(x,s2.find(',',x)-x).c_str(),NULL)); // 存储训练数据各标签概率
        x = s2.find(',',x)+1;
    }
    v[n].push_back(strtod(s2.substr(x).c_str(),NULL));
}
infile.close();
```

验证代码：

```
unordered_map<int,vector<double>> v2,v3;
// validating...
infile.open("validation_set.csv");
node N[n+1];
int sum=0;
getline(infile,s);
while (getline(infile,s)) {
    sum++; // 动态计算验证集数据个数
    int l = s.find(','),x = 0;
    unordered_map<string,int> m;
    string s1 = s.substr(0,l),s2 = s.substr(l+1); // s1：句子，s2：标签
    // split句子
    while (s1.find(' ',x) != string::npos) {
        m[s1.substr(x,s1.find(' ',x)-x)]++;
        x = s1.find(' ',x)+1;
    }
    m[s1.substr(x)]++;
    x = 0;
    while (s2.find(',',x) != string::npos) {
        v2[sum].push_back(strtod(s2.substr(x,s2.find(',',x)-x).c_str(),NULL));
// 记录验证集各标签概率
        x = s2.find(',',x)+1;
    }
    v2[sum].push_back(strtod(s2.substr(x).c_str(),NULL));
    // 初始化验证数据与训练数据距离
    for (i=1; i<=n; i++) {
        N[i].index = i;
        N[i].l = 0;
    }
    // 计算距离
    for (i=1; i<=n; i++) N[i].l = counting(i,m)+1;
    sort(N+1,N+n+1,cmp); // 按距离升序排序
```

```
    double a[6] = {0,0,0,0,0,0},dis=0;
    for (i=1; i<=k; i++) { // 计算比例
        dis += 1.0/N[i].l; // 把比例归一化
        for (j=0; j<6; j++) a[j] += v[N[i].index][j]/N[i].l;
    }
    for (i=0; i<6; i++) v3[sum].push_back(a[i]/dis); // 记录验证集计算出的概率
}
infile.close();
```

计算相关系数:

```
double COR[6] = {0,0,0,0,0,0},X[6] = {0,0,0,0,0,0},Y[6] = {0,0,0,0,0,0};
// 计算平均值 X, Y
for (i=1; i<=sum; i++)
    for (j=0; j<6; j++) {
        X[j] += v2[i][j];
        Y[j] += v3[i][j];
    }
for (i=0; i<6; i++) {
    X[i] /= sum;
    Y[i] /= sum;
}
// 计算相关系数
for (j=0; j<6; j++) {
    double cov=0,w1=0,w2=0;
    for (i=1; i<=sum; i++) {
        cov += (v2[i][j]-X[j])*(v3[i][j]-Y[j]);
        w1 += (v2[i][j]-X[j])*(v2[i][j]-X[j]);
        w2 += (v3[i][j]-Y[j])*(v3[i][j]-Y[j]);
    }
    COR[j] = cov/sqrt(w1*w2);
}
cout << "相关系数: " << (COR[0]+COR[1]+COR[2]+COR[3]+COR[4]+COR[5])/6 << endl;
```

测试代码:

```
// testing...
infile.open("test_set.csv");
outfile.open("18340146_songyujie_KNN_regression.csv");
getline(infile,s);
while (getline(infile,s)) {
    int l1 = s.find(',');
    int l2 = s.find(',',l1+1),x = 0;
    string s1 = s.substr(l1+1,l2-l1);
    unordered_map<string,int> m;
    // split句子
    while (s1.find(' ',x) != string::npos) {
        m[s1.substr(x,s1.find(' ',x)-x)]++;
        x = s1.find(' ',x)+1;
    }
    m[s1.substr(x)]++;
    x = 0;
    // 初始化验证数据与训练数据距离
    for (i=1; i<=n; i++) {
        N[i].index = i;
        N[i].l = 0;
```

```
    }
    // 计算距离
    for (i=1; i<=n; i++) N[i].l = counting(i,m)+1; // 防止距离为0导致归一出现除0
    sort(N+1,N+n+1,cmp); // 按距离升序排序
    double a[6] = {0,0,0,0,0,0},dis=0;
    for (i=1; i<=k; i++) { // 计算比例
        dis += 1.0/N[i].l; // 把比例归一化
        for (j=0; j<6; j++) a[j] += v[N[i].index][j]/N[i].l;
    }
    outfile << s.substr(0,12) << "," << setprecision(4) << a[0]/dis << "," <<
a[1]/dis << "," << a[2]/dis << "," << a[3]/dis << "," << a[4]/dis << "," <<
a[5]/dis << endl; // 文件输出概率
}
infile.close();
outfile.close();
```

实验结果以及分析：

调参过程：我先对参数 k 调为最优值后，再调整参数 p 为最优值：

调 k 过程（暂定 p = 3）

参数 k 取值	相关系数
1	0.165547
2	0.210430
3	0.230659
4	0.238068
5	**0.251251**
10	0.232187
15	0.235035

由表知，参数 k 定为 5，下面继续调 p：

参数 p 取值	相关系数
1	0.249052
2	0.237369
3	**0.251251**
4	0.230523
5	0.218291

因此，最终的参数为 **k = 5，p = 3**， 相关系数为 **0.251251**。

运行 lab1_3_1.exe 之后，输出验证结果（如下图）并生成 18340146_songyujie_KNN_regression.csv 结果文件：

```
相关系数：0.251251

--------------------------------
Process exited after 0.8671 seconds with return value 0
请按任意键继续. . .
```

当然，这个相关系数也有点小，我寻找了一下原因：因为训练数据、测试数据的句子和任务二是完全一样的，因此也存在和上文同样的问题：算法中每个单词权重相同、否定词的影响、数据不够细致、标签过多数据量较少...

创新点：

和任务二相同，我也对应修改了任务三的相似度和带权投票，相似度计算公式和任务二一样，而带权投票也相应从原算法的倒数改为正数（相似度越高，权重应该越大），当然概率变化系数也要修改成 $1/\sum_{i=1}^{k} simi(x, y_i)$，最终的代码为 lab1_3_2.cpp 文件。调最优参 k = 7,q = 1 （同样无参数 p）时，相关系数优化至 0.411811，远优于原算法

```
相关系数：0.411811

--------------------------------
Process exited after 1.251 seconds with return value 0
请按任意键继续. . .
```

因此，最终的 18340146_songyujie_KNN_regression.csv 文件通过新算法计算产生

下图为生成的 18340146_songyujie_KNN_regression.csv 部分数据：

	A	B	C	D	E	F	G	H
1	1	senator carl krueger thinks ipods can kill you	0.08299	0.0664	0.1983	0.07472	0.2966	0.2809
2	2	who is prince frederic von anhalt	0.09871	0.1736	0.05472	0.188	0.2607	0.2242
3	3	prestige has magic touch	0	0	0.03784	0.3639	0.2568	0.3415
4	4	study female seals picky about mates	0.09145	0.1247	0.06831	0.3657	0.06828	0.2816
5	5	no e book for harry potter vii	0.08554	0.0821	0	0.4635	0.09429	0.2746
6	6	blair apologises over friendly fire inquest	0.07278	0.06903	0.2561	0.02784	0.3811	0.1932
7	7	vegetables may boost brain power in older adults	0.009018	0.003505	0.1337	0.4001	0.07104	0.3826
8	8	afghan forces retake town that was overrun by taliban	0.09054	0.01642	0.3527	0.03693	0.3705	0.1329
9	9	skip the showers male sweat turns women on study says	0.04084	0.1724	0.01051	0.1177	0.02102	0.6376
10	10	made in china irks some burberry shoppers	0.06071	0.05046	0.1617	0.3564	0.07009	0.3006
11	11	britain to restrict immigrants from new eu members	0.08088	0.01538	0.2119	0.2995	0.2666	0.1257
12	12	canadian breakthrough offers hope on autism	0.1251	0.03139	0.1737	0.3357	0.1006	0.2336
13	13	russia to strengthen its military muscle	0.1069	0.08418	0.2364	0.2818	0.1367	0.1541
14	14	alzheimer s drugs offer no help study finds	0.1128	0.1368	0.1445	0.2595	0.1563	0.1901
15	15	uk police slammed over terror raid	0.1016	0.09408	0.4633	0.09553	0.1278	0.1176
16	16	no rejects police state claim	0.1811	0.1382	0.3688	0.0762	0.1296	0.1061
17	17	oprah announces new book club pick	0.06479	0.09327	0.1129	0.428	0.1315	0.1695
18	18	smith can t be buried until hearing	0.1052	0.03335	0.138	0.1299	0.3046	0.289
19	19	african nation hopes whoopi can help	0.02125	0.01783	0.09813	0.3986	0.14	0.3242
20	20	tourism lags in bush s hometown	0.0697	0.02964	0.1202	0.4458	0.1474	0.1873
21	21	air france klm profit rises	0.194	0.06802	0.2036	0.2467	0.1675	0.1201
22	22	au regrets sudan s expulsion of un envoy	0.07139	0.05554	0.1091	0.4076	0.2434	0.113
23	23	taiwan s mr clean indicted steps down	0.04503	0.02269	0.114	0.3759	0.2838	0.1586
24	24	martian life could have evaded detection by viking landers	0.1121	0.0292	0.09328	0.03011	0.1951	0.5401
25	25	turner pays for boston bombing	0.0811	0.05056	0.02751	0.515	0.107	0.2189
26	26	serena misses bangalore tournament	0.003671	0.03857	0.07042	0.01178	0.5195	0.3561

思考题：

为什么是倒数？ 因为距离越长，权重应该越小，因此取倒数满足该性质

如果要求得到的每一种标签的概率的和等于 1，应该怎么处理？ 对每一个概率乘上一个概率变化系数 $1/(\sum_{j=1}^{k} 1/d_{i,j})$，即把所有概率等额放大或缩小一个特定的倍数，使得它们的和为 1.