人工智能实验报告 LAB1

（2021学年秋季学期）

课程名称：Artificial Intelligence

教学班级	计科2班	专业（方向）	计算机科学与技术
学号	19335174	姓名	施天予

Task1：TF-IDF

一、 实验题目

TF_IDF

二、 实验内容

1、算法原理

计算TF-IDF矩阵，需要先分别计算出TF矩阵和IDF向量。

TF矩阵为词频归一化后的概率表示，其公式为：

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

$tf_{i,d}$表示文档 d 单词 i 出现的次数除以该文档中的单词总数。TF矩阵表示某个文档中的特定单词的权重。

IDF向量为逆向文档频率，如果一个词语在每篇文章都出现过，说明它反应文本特性的能力不足，应当缩小权重。其公式为：

$$idf_i = \log \frac{|D|}{|D_i|}$$

i 为某个单词，D_i为单词 i 在多少篇文档中出现了，而D为文档总数。IDF向量衡量了每个单词在所有文档中出现的频率，能度量该词语的普遍重要性。

同时考虑两者，将TF和IDF相乘，产生了TF-IDF矩阵。其公式如下：

$$tfidf_{i,j} = \frac{n_{i,j}}{\sum_{k=1}^{V} n_{i,k}} * \log \frac{|D|}{|D_i| + 1}$$

IDF中，分母要加一是为了防止出现分母等于零的情况。

依据上述公式和原理计算TF、IDF，最后就能算出TF-IDF。

2、伪代码

```
/* input: data是文本内容 */
/* return: 返回TFIDF矩阵 */
Procedure TFIDF(data)
    word_list = count_words(data) /* 构建词表 */
    idf = IDF(data, word_list)  /* 通过IDF公式，按照词表计算idf向量 */
    for line in data:   /* 遍历文本每一行 */
        tf = TF(line)   /* 计算每行中的TF值，tf是一个字典 */
        for word in word_list:
            if word in tf:
                x = tf[word]*idf[index] /*用tf对应的值和idf对应的值相乘*/
                将x加入list中
            else
                将0加入list中
        将完成相乘的行向量list加入tfidf矩阵，以此构建整个矩阵
    return tfidf
```

3、关键代码展示

读取文件

```
def read_file():
    data = []
    with open("semeval.txt", "r") as f:
        total = len(f.readlines())      # 统计行数
        f.seek(0, 0)                    # 文件指针复位
        for sentence in f:
            # 拆分每一行并将第三个部分的句子赋值给sentence，取[0:-1]是为了
把句末的回车符去掉
            data.append(sentence.split('\t')[2][0:-1])
    return data, total
```

构建词表

```
def count_words(data):
    word_list = []
    for sentence in data:
        words =  sentence.split(' ')    # 将每一行用空格划分为多个单词
        for word in words:
            if word not in word_list:
                word_list.append(word)  # 如果单词不在词表中，则将单词加入词表
    return word_list
```

计算TF矩阵的每一行

```
def TF(words):
    tf = {}
    num = len(words)    # 计算一行中的单词个数
    for word in words:
        tf[word] = (tf.get(word, 0)*num+1)/num  # 按照公式计算TF
    return tf
```

计算IDF向量

```
def IDF(data, word_list):
    idf = []
    for word in word_list:  # 遍历词表中的每个词
        count = 0
        for sentence in data:
            words =  sentence.split(' ')    # 将每一行用空格划分为多个单词
            if word in words:
                count += 1  # 如果该单词在这一行中，计数器count加上1
        idf.append(math.log(total/(count+1)))   # 利用公式计算
    return idf
```

输出到文件

```
def write_file(tfidf):
    with open("19335174_ShiTianyu_TFIDF.txt", "w") as f:
        for line in tfidf:
            for x in line:
                if x != 0:  # 如果值不为0，输出该值
                    f.write(str(x))
                    f.write(' ')
            f.write('\n')
```

主函数

```
if __name__ ==  '__main__':
    data, total = read_file()       # data是从文件读入的数据，total是总行
数
    word_list = count_words(data)   # 由data构建词表word_list
    idf = IDF(data, word_list)      # 计算idf的值
    tfidf = []
    for sentence in data:
        words = sentence.split(' ')
        tf = TF(words)              # 计算每一行的tf的值
        line = []
        index = 0
        for word in word_list:
            x = tf.get(word, 0)*idf[index]  # 利用公式计算tf-idf
            line.append(x)                  # 在该行中添加一个值
            index += 1                      # idf的索引加1
        tfidf.append(line)          # tfidf矩阵添加一行
    write_file(tfidf)           # 输出到文件
```

三、 实验结果及分析

首先利用 semeval_sample.txt 作为输入文件，方便检查，得到如下结果

19335174_ShiTianyu_TFIDF.txt

```
-0.07192051811294523 0.1013662770270411
-0.14384103622589045 0.1013662770270411
-0.047947012075296815 0.06757751801802739 0.06757751801802739

```

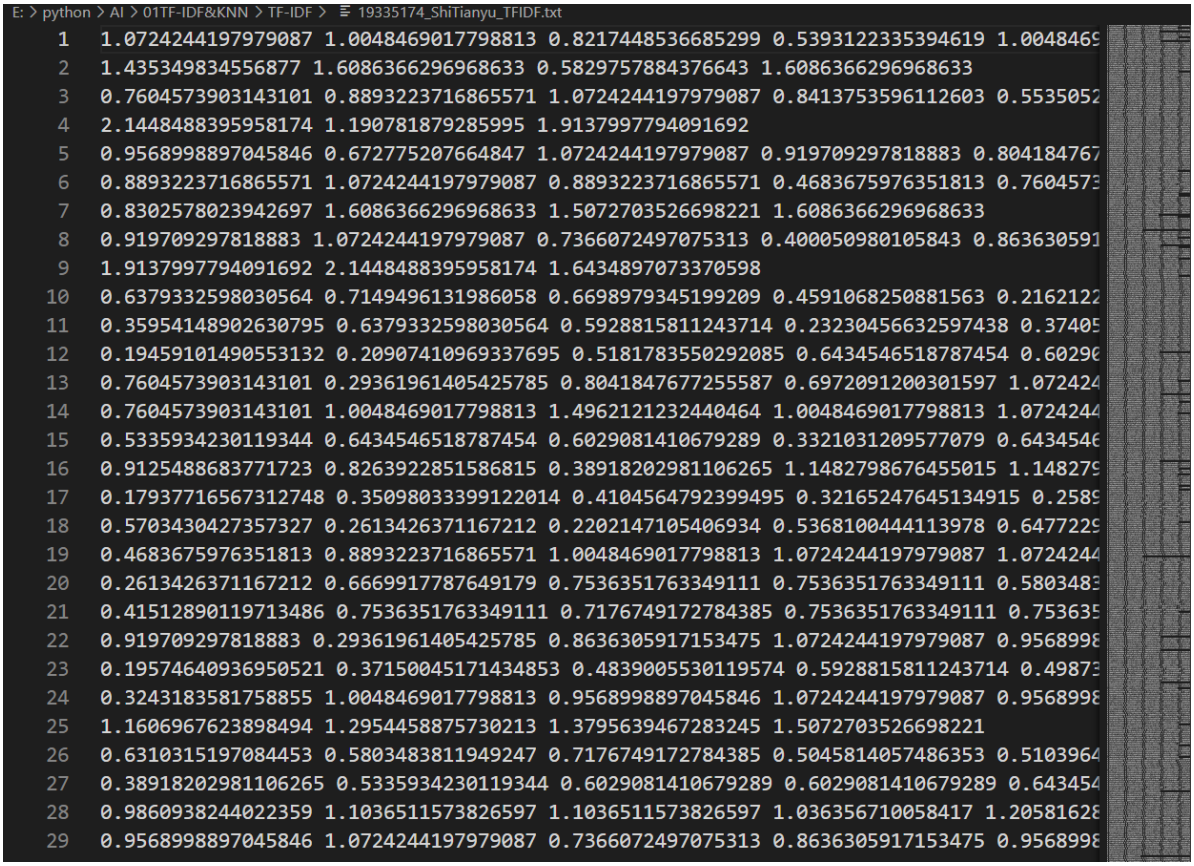
这与所给的19881234_Sample_TFIDF.txt 文件结果相同，只是精确位数更多，说明程序是正确的

19881234_Sample_TFIDF.txt.txt

```
-0.0719205 0.101366
-0.143841 0.101366
-0.047947 0.0675775 0.0675775

```

利用 semeval.txt 文件作为输入文件，可以得到如下结果



四、思考题

1、IDF的第二个计算公式中分母多了个1是为什么？

如果一个词向量在所有文本中都没有出现过，这样就会导致分母为0。加1防止分母为0，且不会对结果有较大影响

2、IDF数值有什么含义？TF-IDF数值有什么含义？

IDF就是指词向量的反文档频率，如果这个词语在越多的篇章中出现，那它可能是一个常用词，不能够反映文本的特征。
TF-IDF就是指将TF的值和IDF相乘后得到的结果，可以降低常用词的权重，更能反映文本的特征。

Task2： KNN_classification

一、 实验题目

KNN分类任务

二、 实验内容

1、算法原理

KNN算法将文本转为多个单词向量，再在所有向量中找出与目标文本最近的k个向量，按照投票选出这k个向量中最多的标签，就作为目标文本的标签。在计算距离时，有多种公式，常用的是Lp距离：

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

当 p = 1 时，为曼哈顿距离：

$$L_1(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|)$$

当 p = 2 时，为欧氏距离：

$$L_2(x_i, x_j) = (\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^2)^{\frac{1}{2}}$$

还可以用余弦距离：

$$\cos\left(\vec{A}, \vec{B}\right) = \frac{\vec{A} \cdot \vec{B}}{\left|\vec{A}\right| \cdot \left|\vec{B}\right|}$$

$$dist(\vec{A}, \vec{B}) = 1 - \cos\left(\vec{A}, \vec{B}\right) \geq 0$$

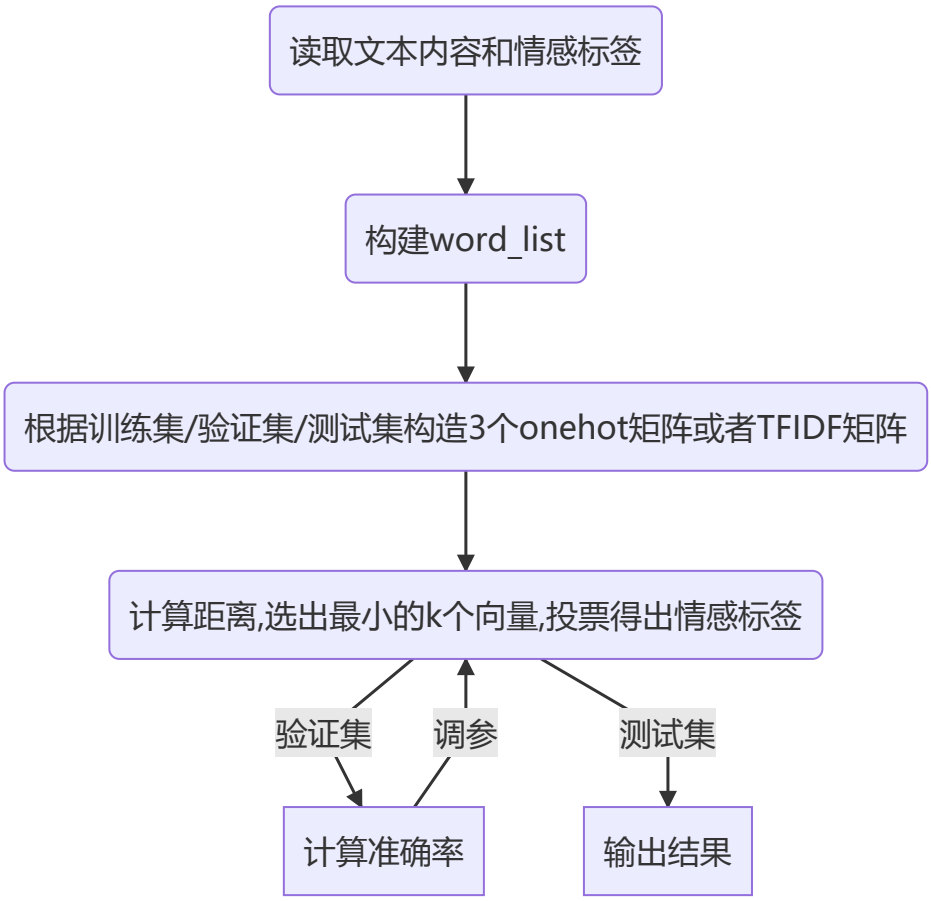
将文本转化为向量形式有很多方法，这次实验我用了onehot矩阵和TFIDF矩阵。

实验中我先用训练集、验证集、测试集的数据构建一个word_list词表。再计算这三个数据集的onehot矩阵或者TFIDF矩阵，根据不同的计算公式，选出距离最小的k个向量，按照众数，投票得出情感标签。

使用不同的距离计算公式，onehot矩阵或者TFIDF矩阵，就可以利用验证集调整k的参数。

最后选出最优的距离公式和文本转向量的方法，就可以得到最优的k值，以及最高的准确率。

2、流程图



3、关键代码展示

读取文件

```
def read_file(file_name):
    # 从file_name读取
    with open(file_name, 'r') as f:
        reader = csv.reader(f)
        first = next(reader)    # 跳过第一行
        data = []
        labels = []
        # 如果是train_set或validation_set，需要读取文本内容和情感
        if file_name == 'train_set.csv' or file_name ==
'validation_set.csv':
            for line in reader:
                data.append(line[0])     # train_set和validation_set的
0列是文本内容
                labels.append(line[1])   # train_set和validation_set的
1列是情感
            return data, labels
        # 如果是test_set，只需要读取文本内容
        else:
            for line in reader:
                data.append(line[1])     # test_data的1列是文本内容，0列
是索引!
```

```
            return data
```

构建词表

```
def count_words(train_data):
    word_list = []
    word_set = set()
    for line in train_data:              # 训练集
        words =  line.split(' ')         # 将每一行用空格划分为多个单词
        for word in words:
            if word not in word_set:     # 用集合加速查找
                word_list.append(word)  # 如果单词不在词表中，则将单词加
入词表
                word_set.add(word)
    return word_list
```

构造onehot矩阵

```
def onehot(data, word_list):
    # onehot矩阵的宽是文本内容data的行数，长是词表word_list中word的个数
    onehot_matrix = np.zeros(shape=(len(data), len(word_list)))
    # 遍历word_list的每个单词，i是word的索引
    for i,word in enumerate(word_list):
        # 遍历文本内容data的每一行，index是line的索引
        for index,line in enumerate(data):
            words = line.split(' ')      # 将每一行用空格划分为多个单词
            if word in words:
                onehot_matrix[index][i] = 1     # onehot矩阵对应位置的值
设为1
    return onehot_matrix
```

构造IF-IDF矩阵（函数TF和IDF与task1相同）

```
def TFIDF(data, word_list):
    idf = IDF(data, word_list)      # 计算idf的值
    tfidf = []
    for sentence in data:
        words = sentence.split(' ')
        tf = TF(words)              # 计算每一行的tf的值
        line = []
        index = 0
        for index, word in enumerate(word_list):
            x = tf.get(word, 0)*idf[index]  # 利用公式计算tf-idf
            line.append(x)                  # 在该行中添加一个值
        tfidf.append(line)          # tfidf矩阵添加一行
    tfidf = np.array(tfidf)
    return tfidf
```

计算Lp距离，可根据不同p值选取不同的距离计算方法

p = 1，曼哈顿距离

p = 2，欧氏距离

```
def cal_distance(train_matrix, line, p=2):
    # train_matrixs是整个训练集的矩阵
    # line是测试集矩阵的某一行
    # p是Lp距离中的p值，默认p=2欧氏距离
    line = line.reshape(1, -1) # 将line从列向量转为行向量
    lines = np.repeat(line, train_matrix.shape[0], axis=0)   # 将行向量
扩展成矩阵
    # Lp距离
    temp = np.abs(lines - train_matrix)     # 求出矩阵相减后的绝对值
    distances = np.sum(temp**p, axis=1)**(1/p)  # 行内求和，和为列向量
    return distances
```

利用KNN处理分类问题

```
def KNN_predict(train_matrix, matrix, k, train_labels):
    # train_matrix是训练集的矩阵
    # matrix是验证集或测试集的矩阵
    # k是超参数
    # train_labels是训练集的情感标签

    # 构造一个共有matrix的行数的个数的list
    result = [0 for i in range(matrix.shape[0])]
    # 遍历验证集/测试集矩阵的每一行
    for index in range(matrix.shape[0]):      # 按下标访问
        line = matrix[index]
        # 求出验证集或测试集矩阵中每一行与整个训练集矩阵的距离
        distances = cal_distance(train_matrix, line, 2)    # 默认欧氏距
离p=2
        sort_index = np.argsort(distances)    # 从小到大排序，返回原始下
标

        # 对距离最小的k组情感标签投票
        dict1 = {}
        i = 0
        while i < k:
            if train_labels[sort_index[i]] not in dict1:
                dict1[train_labels[sort_index[i]]] = 1      # 不在字典
中设为1
            else:
                dict1[train_labels[sort_index[i]]] += 1     # 在字典中
增加1
            i += 1
        # 找出字典中值最大对应的键
```

```
            result[index] = max(dict1, key=lambda x: dict1[x])
        return result   # 返回预测的情感标签集合
```

从验证集中计算模型的准确率

```
def cal_accuracy(real, predict):
    # predict是预测的标签
    # real是真实的标签
    total = len(real)
    count = 0
    for i in range(total):
        if real[i] == predict[i]:
            count += 1      # 如果预测正确，计数器count加1
    return count/total      # 返回准确率
```

主函数

```
if __name__ ==  '__main__':
    # 读取文本内容和标签
    train_data, train_labels = read_file("train_set.csv")
    valid_data, valid_labels = read_file("validation_set.csv")
    test_data = read_file("test_set.csv")


    # 构建词表
    word_list = count_words(train_data)

    # 构建训练集、验证集、测试集的onehot矩阵
    train_matrix = onehot(train_data, word_list)
    valid_matrix = onehot(valid_data, word_list)
    test_matrix = onehot(test_data, word_list)

    # # 构建训练集、验证集、测试集的tfidf矩阵
    # train_matrix = TFIDF(train_data, word_list)
    # valid_matrix = TFIDF(valid_data, word_list)
    # test_matrix = TFIDF(test_data, word_list)

    # 训练模型，保存准确率最高的k值
    k_best = 1
    accuracy_best = 0
    k = 1
    while k <= 20:
        # 在验证集上预测情感标签
        valid_predict = KNN_predict(train_matrix, valid_matrix, k,
train_labels)
        # 计算准确率
        accuracy = cal_accuracy(valid_labels, valid_predict)
        print('k = ', k, ', accuracy = ', accuracy)
```

```
            # 如果准确率最高，则保存k值
            if accuracy > accuracy_best:
                k_best = k
                accuracy_best = accuracy
            k += 1
        print('The best k is', k_best)                          # 输出最佳k值
        print('The highest accuracy is', accuracy_best)         # 输出最高的
accuracy值

        # 在测试集上预测情感标签，运用pandas输出csv文件
        test_predict = KNN_predict(train_matrix, test_matrix, k_best,
train_labels)
        test_output = pd.DataFrame({'Words (split by space)': test_data,
'label': test_predict})

     test_output.to_csv('19335174_ShiTianyu_KNN_classification.csv.csv',
index=None, encoding='utf8')
```

4、创新点&优化

- 在构造word_list时，用集合set加速查找，方便插入
- 在计算距离时，将测试集的单个test行向量扩展为矩阵，运用numpy直接和训练集的矩阵进行计算，一次得到单个test的所有距离的向量，加速运算

三、 实验结果及分析

1、实验结果展示示例

选取k从1到20

欧氏距离 p = 2，onehot矩阵

```
k =  1 , accuracy =  0.34405144694533376
k =  2 , accuracy =  0.34405144694533376
k =  3 , accuracy =  0.36012861736334406
k =  4 , accuracy =  0.3536977491961415
k =  5 , accuracy =  0.3536977491961415
k =  6 , accuracy =  0.3729903536977492
k =  7 , accuracy =  0.3408360128617363
k =  8 , accuracy =  0.34405144694533376
k =  9 , accuracy =  0.36977491961414793
k =  10 , accuracy =  0.38263665594855306
k =  11 , accuracy =  0.39228295819935693
k =  12 , accuracy =  0.40514469453376206
k =  13 , accuracy =  0.40836012861736337
k =  14 , accuracy =  0.3954983922829582
k =  15 , accuracy =  0.3987138263665595
k =  16 , accuracy =  0.39228295819935693
k =  17 , accuracy =  0.39228295819935693
k =  18 , accuracy =  0.39228295819935693
k =  19 , accuracy =  0.3954983922829582
k =  20 , accuracy =  0.3954983922829582
The best k is 13
The highest accuracy is 0.40836012861736337
```

曼哈顿距离 p = 1，onehot矩阵

```
k =  1 , accuracy =  0.34405144694533376
k =  2 , accuracy =  0.34405144694533376
k =  3 , accuracy =  0.36012861736334406
k =  4 , accuracy =  0.3536977491961415
k =  5 , accuracy =  0.3536977491961415
k =  6 , accuracy =  0.3729903536977492
k =  7 , accuracy =  0.3408360128617363
k =  8 , accuracy =  0.34405144694533376
k =  9 , accuracy =  0.36977491961414793
k =  10 , accuracy =  0.38263665594855306
k =  11 , accuracy =  0.39228295819935693
k =  12 , accuracy =  0.40514469453376206
k =  13 , accuracy =  0.40836012861736337
k =  14 , accuracy =  0.3954983922829582
k =  15 , accuracy =  0.3987138263665595
k =  16 , accuracy =  0.39228295819935693
k =  17 , accuracy =  0.39228295819935693
k =  18 , accuracy =  0.39228295819935693
k =  19 , accuracy =  0.3954983922829582
k =  20 , accuracy =  0.3954983922829582
The best k is 13
The highest accuracy is 0.40836012861736337
```

曼哈顿距离 p = 1，TFIDF矩阵

```
k =  1 , accuracy =  0.3954983922829582
k =  2 , accuracy =  0.3954983922829582
k =  3 , accuracy =  0.3890675241157556
k =  4 , accuracy =  0.3729903536977492
k =  5 , accuracy =  0.37942122186495175
k =  6 , accuracy =  0.3858520900321543
k =  7 , accuracy =  0.38263665594855306
k =  8 , accuracy =  0.36977491961414793
k =  9 , accuracy =  0.3890675241157556
k =  10 , accuracy =  0.3954983922829582
k =  11 , accuracy =  0.38263665594855306
k =  12 , accuracy =  0.3858520900321543
k =  13 , accuracy =  0.405144694533762O6
k =  14 , accuracy =  0.4115755627009646
k =  15 , accuracy =  0.40836012861736337
k =  16 , accuracy =  0.39228295819935693
k =  17 , accuracy =  0.4180064308681672
k =  18 , accuracy =  0.40192926045016075
k =  19 , accuracy =  0.3987138263665595
k =  20 , accuracy =  0.3987138263665595
The best k is 17
The highest accuracy is 0.4180064308681672
```

欧氏距离 p = 2，TFIDF矩阵

```
k =  1 , accuracy =  0.3215434083601286
k =  2 , accuracy =  0.3215434083601286
k =  3 , accuracy =  0.263665594855305 44
k =  4 , accuracy =  0.2508038585209003
k =  5 , accuracy =  0.2508038585209003
k =  6 , accuracy =  0.22186495176848875
k =  7 , accuracy =  0.19935691318327975
k =  8 , accuracy =  0.19292604501607716
k =  9 , accuracy =  0.1864951768488746
k =  10 , accuracy =  0.18006430868167203
k =  11 , accuracy =  0.18006430868167203
k =  12 , accuracy =  0.19935691318327975
k =  13 , accuracy =  0.19614147909967847
k =  14 , accuracy =  0.2090032154340836
k =  15 , accuracy =  0.2057877813504823
k =  16 , accuracy =  0.189710610932475 88
k =  17 , accuracy =  0.21221864951768488
k =  18 , accuracy =  0.21543408360128619
k =  19 , accuracy =  0.22508038585209003
k =  20 , accuracy =  0.2282958199356913
The best k is 1
The highest accuracy is 0.3215434083601286
```

2、评测指标展示及分析

	曼哈顿距离	欧氏距离	onehot矩阵	TFIDF矩阵	准确率	最优k值
初始	0	1	1	0	40.84%	13
优化1	1	0	1	0	40.84%	13
优化2	1	0	0	1	41.80%	17
优化3	0	1	0	1	32.15%	1
最优结果	1	0	0	1	41.80%	17

当选用曼哈顿距离，TFIDF矩阵时，准确率最高达41.80%，最优k值是17

奇怪地发现我使用onehot矩阵时，曼哈顿距离和欧氏距离的结果一样

Task3：KNN_regression

一、 实验题目

KNN回归任务

二、 实验内容

1、算法原理

与分类问题类似，先将文本转化为词向量，找到距离最近的k个向量，将距离的倒数作为权重，计算目标文本和最近的k个向量的每个情感在距离的加权求和后的结果。

以happy为例，公式如下：

$$P(test1ishappy) = \sum_{k=1}^{K} \frac{train_k\ probility}{d(train_k, test)}$$

这样得出来的结果之和可能不为1，需要进行归一化处理，用每种情感的比例除以6种情感比例的总和，公式如下：

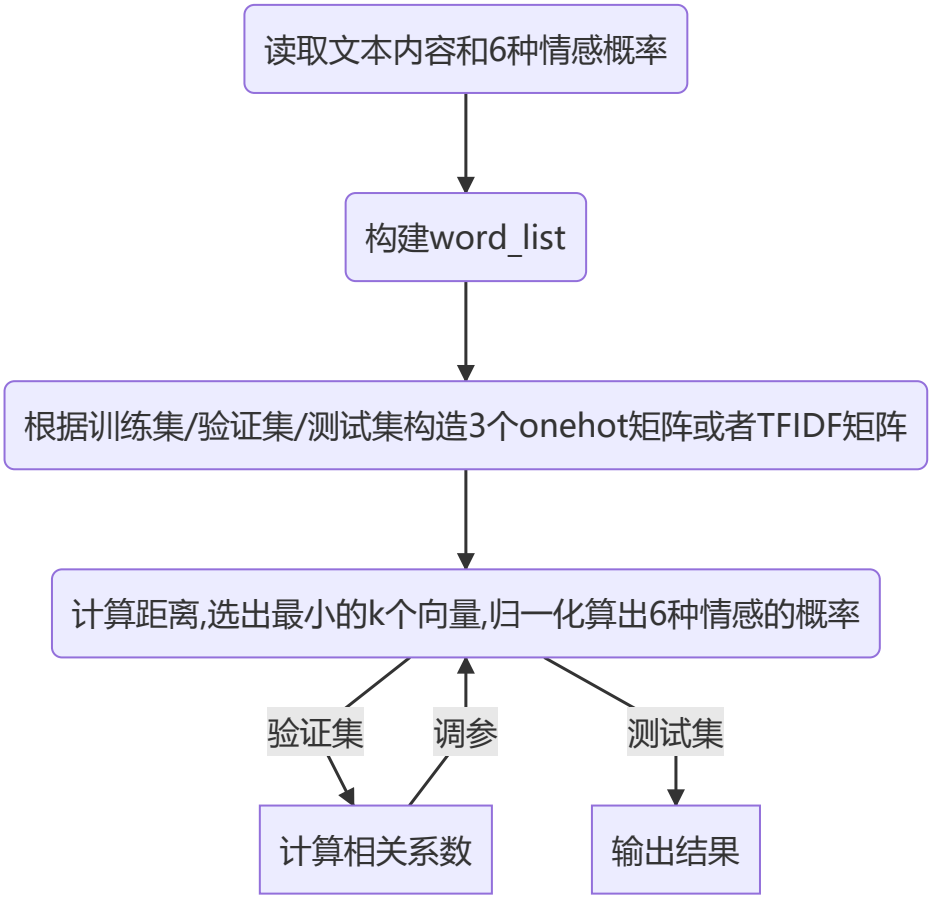
$$porb_i = \frac{prob_i}{\sum_{j=1}^{6} prob_j}$$

与分类问题不同的是，回归问题的预测结果不可能和实际结果完全准确。可以使用相关系数作为性能指标，判断预测结果和实际结果的差距。

$$COR(X,Y)=\frac{cov(X,Y)}{\sigma_X\sigma_Y}=\frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{\sqrt{\sum_{i=1}^{n}(X_i-\overline{X})^2\sum_{i=1}^{n}(Y_i-\overline{Y})^2}}$$

本题中的结果有六个概率值。先分别计算六个维度上的真实概率值和预测概率值的相关系数，然后对六个维度取平均，计算得到最终相关系数作为判断依据。相关系数的值越大，预测值和真实值的线性相关程度就越好。

2、流程图



3、关键代码展示

读取文件

```
def read_file(file_name):
    # 从file_name读取
    with open(file_name, 'r') as f:
        reader = csv.reader(f)
        first = next(reader)
        data = []
        labels = [[]for i in range(6)]
```

```
        # 如果是train_set或validation_set，需要读取文本内容和情感
        if file_name == 'train_set.csv' or file_name ==
'validation_set.csv':
            for line in reader:
                data.append(line[0])     # train_set和validation_set的
0列是文本内容
                for j in range(6):
                    labels[j].append(float(line[j+1]))    # j+1列是
labels[j]情感
            return data, labels
        # 如果是test_set，只需要读取文本内容
        else:
            for line in reader:
                data.append(line[1]) # test_data的1列是文本内容，0列是索
引！
            return data
```

构建词表、计算onehot矩阵、计算TFIDF矩阵、计算距离的函数与KNN_classification相同，这里就不再展示

利用KNN处理回归问题

```
def KNN_predict(train_matrix, matrix, k, train_labels):
    # train_matrix是训练集的矩阵
    # matrix是验证集或测试集的矩阵
    # k是超参数
    # train_labels是训练集的情感标签

    # 返回的是行数为6，列数为测试矩阵的行数的list
    result = [[0.0 for j in range(matrix.shape[0])]for i in range(6)]
    # 遍历矩阵的每一行
    for index in range(matrix.shape[0]):      # 按下标访问
        line = matrix[index]
        # 求出验证集或测试集TFIDF矩阵中每一行与整个训练集TFIDF矩阵的距离
        distances = cal_distance(train_matrix, line, 1)    # 默认欧氏距
离p=2
        sort_index = np.argsort(distances)    # 从小到大排序，返回原始下
标
        i = 0
        total = 0
        while i < k:
            for j in range(6):
                if distances[sort_index[i]] == 0:
                    distances[sort_index[i]] = 0.01     # 防止除数为0
                # 公式计算情感标签概率
                result[j][index] += train_labels[j][sort_index[i]] /
float(distances[sort_index[i]])
                # 加入总和，用于归一化
```

```
                total += train_labels[j][sort_index[i]] /
float(distances[sort_index[i]])
            i += 1
        for i in range(6):
            result[i][index] /= total    # 归一化
    return result
```

从验证集中计算模型的相关系数

```
def cal_cor(real, predict):
    # predict是预测的标签概率
    # real是真实的标签概率
    real = np.array(real)
    predict = np.array(predict)
    # 利用numpy返回一个12*12的相关系数矩阵
    correlation = np.corrcoef(real, predict)
    cor = 0
    for i in range(6):
        # correlation[i][6+i]代表real矩阵i行和predict矩阵i行的相关系数
        cor += correlation[i][6+i]
    return cor / 6.0
```

主函数

```
if __name__ ==  '__main__':
    # 读取文本内容和标签
    train_data, train_labels = read_file("train_set.csv")
    valid_data, valid_labels = read_file("validation_set.csv")
    test_data = read_file("test_set.csv")

    # 构建词表
    word_list = count_words(train_data)

    # # 构建训练集、验证集、测试集的onehot矩阵
    # train_matrix = onehot(train_data, word_list)
    # valid_matrix = onehot(valid_data, word_list)
    # test_matrix = onehot(test_data, word_list)

    # 构建训练集、验证集、测试集的tfidf矩阵
    train_matrix = TFIDF(train_data, word_list)
    valid_matrix = TFIDF(valid_data, word_list)
    test_matrix = TFIDF(test_data, word_list)

    # 训练模型，保存准确率最高的k值
    k_best = 1
    cor_best = 0
    k = 1
    while k <= 20:
```

```
        # 在验证集上预测情感标签概率
        valid_predict = KNN_predict(train_matrix, valid_matrix, k,
train_labels)
        # 计算相关系数
        cor = cal_cor(valid_labels, valid_predict)
        print('k = ', k, ', cor = ', cor)
        # 如果相关系数最大，则保存k值
        if cor > cor_best:
            k_best = k
            cor_best = cor
        k += 1
    print('The best k is', k_best)                          # 输出最佳k值
    print('The highest correlation coefficient is', cor_best)      #
输出最大的cor值

    # 在测试集上预测情感标签概率，运用pandas输出csv文件
    test_predict = KNN_predict(train_matrix, test_matrix, k_best,
train_labels)
    test_output = pd.DataFrame({'textid':
np.arange(len(test_predict[0]))+1, 'anger': test_predict[0],
'disgust': test_predict[1],'fear': test_predict[2], 'joy':
test_predict[3], 'sad': test_predict[4], 'surprise': test_predict[5]})
    test_output.to_csv('19335174_ShiTianyu_KNN_regression.csv',
index=None, encoding='utf8')
```

三、 实验结果及分析

1、实验结果展示示例

选取k从1到20

欧氏距离 p = 2，onehot矩阵

```
k =  1 , cor =  0.18865916204648345
k =  2 , cor =  0.2270697610979486
k =  3 , cor =  0.2144025084161314
k =  4 , cor =  0.24649989114114393
k =  5 , cor =  0.2587084440486992
k =  6 , cor =  0.25465041177936687
k =  7 , cor =  0.26248204153160987
k =  8 , cor =  0.25624631068659537
k =  9 , cor =  0.2679723613780796
k =  10 , cor =  0.26982328819077056
k =  11 , cor =  0.26085882032261604
k =  12 , cor =  0.2633449842779176
k =  13 , cor =  0.2610244425798825
k =  14 , cor =  0.2609013119989258
k =  15 , cor =  0.26297770143093
k =  16 , cor =  0.2578778389962139
k =  17 , cor =  0.2555194008498671
k =  18 , cor =  0.24803961276592576
k =  19 , cor =  0.2541288665506108
k =  20 , cor =  0.25260393120778735
The best k is 10
The highest correlation coefficient is 0.26982328819077056
```

曼哈顿距离 p = 1，onehot矩阵

```
k =  1 , cor =  0.18865916204648345
k =  2 , cor =  0.22925920568041733
k =  3 , cor =  0.21974826362655112
k =  4 , cor =  0.2528928380321679
k =  5 , cor =  0.2663785555026914
k =  6 , cor =  0.263318313240218
k =  7 , cor =  0.2714717319823646
k =  8 , cor =  0.2656278807735775
k =  9 , cor =  0.27776806082018174
k =  10 , cor =  0.27976705774816873
k =  11 , cor =  0.2705179025039323
k =  12 , cor =  0.273019444776408
k =  13 , cor =  0.27106363678821427
k =  14 , cor =  0.27086617441155486
k =  15 , cor =  0.27231506971910974
k =  16 , cor =  0.2670344405180649
k =  17 , cor =  0.26435581497833
k =  18 , cor =  0.25736975474837426
k =  19 , cor =  0.26322749644944055
k =  20 , cor =  0.2616978192810722
The best k is 10
The highest correlation coefficient is 0.27976705774816873
```

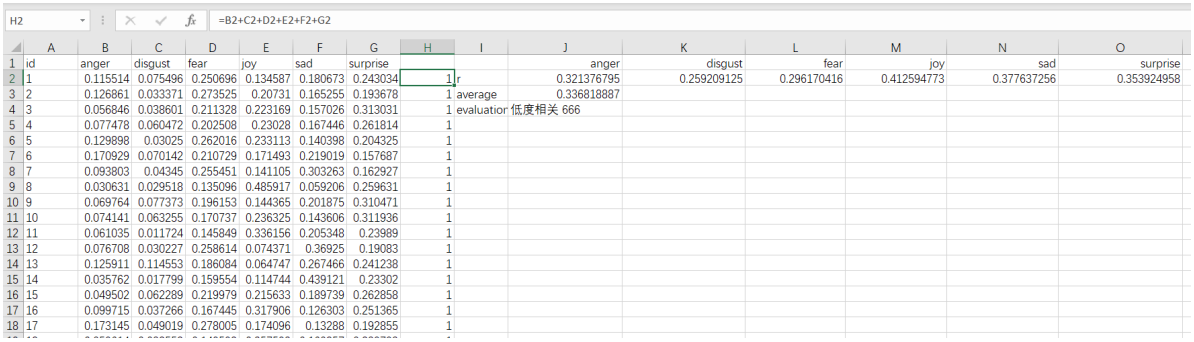
曼哈顿距离 p = 1，TFIDF矩阵

```
k =  1 , cor =  0.28919569868849376
k =  2 , cor =  0.32980421654924785
k =  3 , cor =  0.31595578820171666
k =  4 , cor =  0.32030201741652514
k =  5 , cor =  0.31666824341349603
k =  6 , cor =  0.31956016996420866
k =  7 , cor =  0.3224146250250189
k =  8 , cor =  0.3051333217991577
k =  9 , cor =  0.30921577409388745
k =  10 , cor =  0.30905414968677103
k =  11 , cor =  0.3095578828211379
k =  12 , cor =  0.32164976511698895
k =  13 , cor =  0.32692458660451684
k =  14 , cor =  0.33681888718003106
k =  15 , cor =  0.3293539992342474
k =  16 , cor =  0.327054855019883
k =  17 , cor =  0.326773394117529
k =  18 , cor =  0.32060601975566677
k =  19 , cor =  0.32303274763615364
k =  20 , cor =  0.3208871111235379
The best k is 14
The highest correlation coefficient is 0.33681888718003106
```

欧氏距离 p = 2，TFIDF矩阵

```
k =  1 , cor =  0.2509119184857549
k =  2 , cor =  0.26343038957735343
k =  3 , cor =  0.2867214602838781
k =  4 , cor =  0.2524217708363378
k =  5 , cor =  0.24392623437653127
k =  6 , cor =  0.24111735893591338
k =  7 , cor =  0.24413380608838465
k =  8 , cor =  0.2368611161107512
k =  9 , cor =  0.23503876229947393
k =  10 , cor =  0.24007007703641
k =  11 , cor =  0.2432568020178093
k =  12 , cor =  0.25829957584280827
k =  13 , cor =  0.24942166406305058
k =  14 , cor =  0.24863351710839252
k =  15 , cor =  0.22771788098414772
k =  16 , cor =  0.2375967729835032
k =  17 , cor =  0.26010701869094105
k =  18 , cor =  0.26315330458065755
k =  19 , cor =  0.25898629311157995
k =  20 , cor =  0.25869338079090426
The best k is 3
The highest correlation coefficient is 0.2867214602838781
```

选用曼哈顿距离和TFIDF矩阵后，将从验证集得出的结果粘贴到validation相关度评估.csv文件，可以得到“低相关666”的评价，并且可以检验出6种情感概率的和为1



H2 =B2+C2+D2+E2+F2+G2

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	id	anger	disgust	fear	joy	sad	surprise			anger	disgust	fear	joy	sad	surprise
2	1	0.115514	0.075496	0.250696	0.134587	0.180673	0.243034	1	r	0.321376795	0.259209125	0.296170416	0.412594773	0.377637256	0.353924958
3	2	0.126861	0.033371	0.273525	0.20731	0.165255	0.193678	1	average	0.336818887					
4	3	0.056846	0.038601	0.211328	0.223169	0.157026	0.313031	1	evaluation	低度相关 666					
5	4	0.077478	0.060472	0.202508	0.23028	0.167446	0.261814	1							
6	5	0.129898	0.03025	0.262016	0.233113	0.140398	0.204325	1							
7	6	0.170929	0.070142	0.210729	0.171493	0.219019	0.157687	1							
8	7	0.093803	0.04345	0.255451	0.141105	0.303263	0.162927	1							
9	8	0.030631	0.029518	0.135096	0.485917	0.059206	0.259631	1							
10	9	0.069764	0.077373	0.196153	0.144365	0.201875	0.310471	1							
11	10	0.074141	0.063255	0.170737	0.236325	0.143606	0.311936	1							
12	11	0.061035	0.011724	0.145849	0.336156	0.205348	0.23989	1							
13	12	0.076708	0.030227	0.258614	0.074371	0.36925	0.19083	1							
14	13	0.125911	0.114553	0.186084	0.064747	0.267466	0.241238	1							
15	14	0.035762	0.017799	0.159554	0.114744	0.439121	0.23302	1							
16	15	0.049502	0.062289	0.219979	0.215633	0.189739	0.262858	1							
17	16	0.099715	0.037266	0.167445	0.317906	0.126303	0.251365	1							
18	17	0.173145	0.049019	0.278005	0.174096	0.13288	0.192855	1							

2、评测指标展示及分析

	曼哈顿距离	欧氏距离	onehot矩阵	TFIDF矩阵	相关系数	最优k值
初始	0	1	1	0	0.2698	10
优化1	1	0	1	0	0.2798	10
优化2	1	0	0	1	0.3368	14
优化3	0	1	0	1	0.2867	3
最优结果	1	0	0	1	0.3368	14

当选用曼哈顿距离，TFIDF矩阵时，相关系数可达0.3368，最优k值是14

这个结果与KNN分类问题所得的评测结果相同，都是使用曼哈顿距离和TFIDF矩阵达到最佳

四、思考题

1、为什么是倒数？如果要求同一测试样本的各个情感概率总和为1，应该如何处理？

因为距离越大，和当前文本的相似度就越小，取倒数可以作为一个权值降低这种情感的影响力，分母越大，影响越小。

让各个情感概率总和为1，可以归一化处理。将所有情感概率加起来得到sum，再用每种情感概率除以sum得到归一化后的情感概率。