

中山大学数据科学与计算机学院

计算机科学与技术专业-人工智能

本科生实验报告

(2018-2019 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	16337341	姓名	朱志儒

• 实验题目

TF-IDF 实验

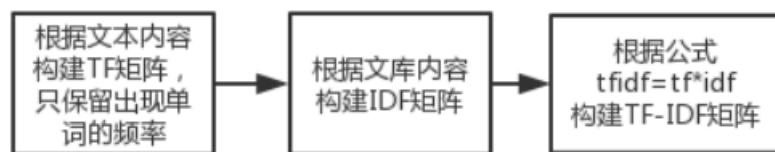
• 实验内容

• 算法原理

由于 semeval 数据集中每一行语句中单词个数远小于整个数据集中的单词个数, 所以构成的 TF-IDF 矩阵是一个稀疏矩阵, 矩阵中存在许多 0, 因此在生成矩阵时, 我选择只保留不为 0 的数据。

算法的整个过程: 首先遍历整个文本来生成不重复词汇向量, 接着根据词汇是否在每行文本中出现来生成 ONE-HOT 矩阵, 再将 ONE-HOT 矩阵中的数据归一化生成 TF 矩阵, 接着计算每个单词出现的文章总数, 再生成 IDF 矩阵, 最后根据公式: $tfidf_{i,j} = tf_{i,j} * idf_i$ 生成 TF-IDF 矩阵。

• 流程图



• 关键代码截图

```
for line in article:
    #生成 TF 矩阵, 只保留出现单词归一化后的频率
    words = line.split(' ')
    tmp = {}
    for word in words:
        if word not in tmp.keys():
            tmp[word] = 1 / len(words)
        else:
            tmp[word] += 1 / len(words)
    TF.append(tmp)
```

```

#去除每句文本中的重复单词
tmp = []
for element in words:
    if element not in tmp:
        tmp.append(element)

#记录单词在每句文本中出现的次数
for word in tmp:
    if word not in IDF.keys():
        IDF[word] = 1
    else:
        IDF[word] += 1

#生成 IDF 矩阵
for key in IDF.keys():
    IDF[key] = math.log(len(article) / (IDF[key] + 1))

#根据公式生成 TF-IDF 矩阵
TF_IDF = []
for i in range(len(TF)):
    tmp = {}
    for key in TF[i].keys():
        tmp[key] = TF[i][key] * IDF[key]
    TF_IDF.append(tmp)

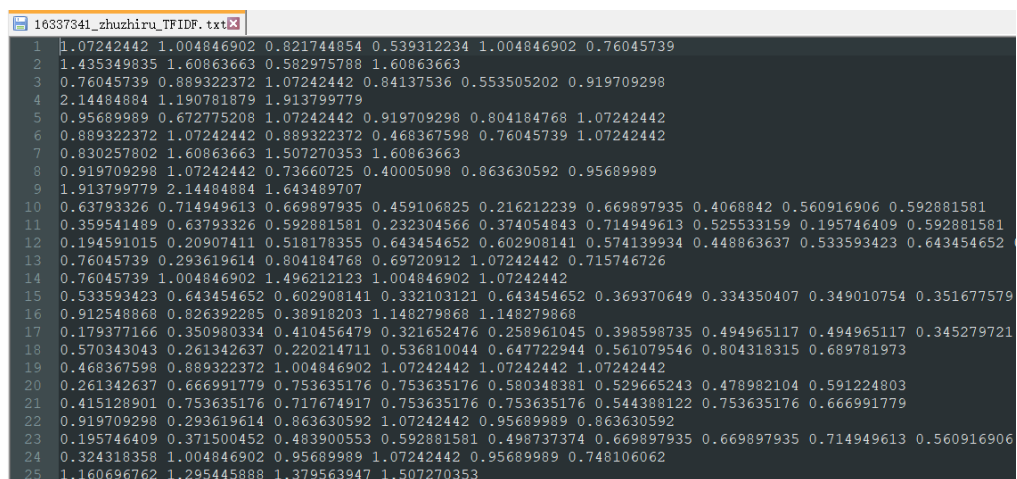
```

• 创新点&优化

TF 和 TF-IDF 矩阵均为稀疏矩阵，在生成 TF 和 TF-IDF 矩阵时没有保存 0，这样减少了占用的内存空间和运行时间。

• 实验结果

实验结果展示示例



```

16337341_zhuzhiru_TFIDF.txt
1 1.07242442 1.004846902 0.821744854 0.539312234 1.004846902 0.76045739
2 1.435349835 1.60863663 0.582975788 1.60863663
3 0.76045739 0.889322372 1.07242442 0.84137536 0.553505202 0.919709298
4 2.14484884 1.190781879 1.913789779
5 0.95689989 0.672775208 1.07242442 0.919709298 0.804184768 1.07242442
6 0.889322372 1.07242442 0.889322372 0.468367598 0.76045739 1.07242442
7 0.830257802 1.60863663 1.507270353 1.60863663
8 0.919709298 1.07242442 0.73660725 0.40005098 0.863630592 0.95689989
9 1.913789779 2.14484884 1.643489707
10 0.63793326 0.714949613 0.669897935 0.459106825 0.216212239 0.669897935 0.4068842 0.560916906 0.592881581
11 0.359541489 0.63793326 0.592881581 0.232304566 0.374054843 0.714949613 0.525533159 0.195746409 0.592881581
12 0.194591015 0.20907411 0.518178355 0.643454652 0.602908141 0.574139934 0.448863637 0.533593423 0.643454652
13 0.76045739 0.293619614 0.804184768 0.69720912 1.07242442 0.715746726
14 0.76045739 1.004846902 1.496212123 1.004846902 1.07242442
15 0.533593423 0.643454652 0.602908141 0.332103121 0.643454652 0.369370649 0.334350407 0.349010754 0.351677579
16 0.912548868 0.826392285 0.38918203 1.148279868 1.148279868
17 0.179377166 0.350980334 0.410456479 0.321652476 0.258961045 0.398598735 0.494965117 0.494965117 0.345279721
18 0.570343043 0.261342637 0.220214711 0.536810044 0.647722944 0.561079546 0.804318315 0.689781973
19 0.468367598 0.889322372 1.004846902 1.07242442 1.07242442 1.07242442
20 0.261342637 0.666991779 0.753635176 0.753635176 0.580348381 0.529665243 0.478982104 0.591224803
21 0.415128901 0.753635176 0.717674917 0.753635176 0.753635176 0.544388122 0.753635176 0.666991779
22 0.919709298 0.293619614 0.863630592 1.07242442 0.95689989 0.863630592
23 0.195746409 0.371500452 0.483900553 0.592881581 0.498737374 0.669897935 0.669897935 0.714949613 0.560916906
24 0.324318358 1.004846902 0.95689989 1.07242442 0.95689989 0.748106062
25 1.160696762 1.295445888 1.379563947 1.507270353

```

• 思考题

1. IDF 的第二个计算公式中分母多了个 1 是为什么？

分母多了个 1 是为了防止分母为 0。

2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

IDF 是指逆向文档频率，IDF 数值越小，说明该词在文库中越常见。TF-IDF 数值越大，说明该词在文章中的重要性越高。

• 实验题目

k-NN 处理分类问题

• 实验内容

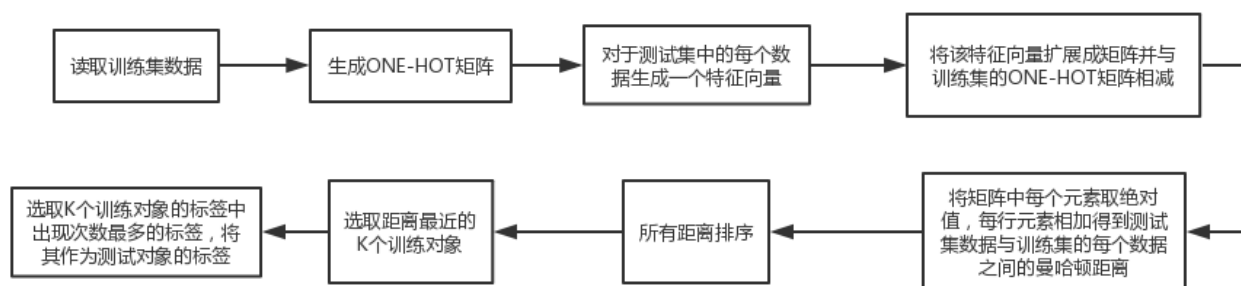
• 算法原理

KNN 算法，即 K 近邻分类算法，根据不同特征值之间的距离来进行分类的一种简单而又懒惰的算法。

KNN 算法的步骤：对于每个测试集中的不带标签的对象，通过使用曼哈顿距离、欧式距离或夹角余弦来计算它与训练集中的每个对象的距离，接着选取距离最近的 K 个训练对象，然后在这 K 个训练对象的标签中选出出现次数最多的标签作为该测试集对象的标签。

在本次实验中，我选择使用曼哈顿距离计算测试集向量与训练集向量的距离。

• 流程图



• 关键代码截图

(1) 对训练集数据的处理：

```
#读取训练集中的数据
train_set = read_csv('lab1_data\\classification_dataset\\train_set.csv')
for i in range(len(train_set)):
    train_set[i][0] = train_set[i][0].split(' ')
```

```

#生成不重复词向量
words_vector = []
for line in train_set:
    words_vector += line[0]
words_vector = list(set(words_vector))

#生成训练集数据的 ONE-HOT 矩阵
xmatrix = np.zeros((len(train_set), len(words_vector)))
for i in range(len(train_set)):
    for word in train_set[i][0]:
        xmatrix[i][words_vector.index(word)] += 1
xmatrix = np.matrix(xmatrix)

```

(2) 实现 KNN 算法:

```

def classification_KNN(test_set, k):
    labels = []
    for line in test_set:
        words = line[0].split(' ')
        test_vector = np.zeros(len(words_vector))

        # 对于测试集中的每个数据生成一个特征向量
        for word in words:
            if word in words_vector:
                test_vector[words_vector.index(word)] += 1

        # 将该特征向量扩展成矩阵并与训练集的 ONE-HOT 矩阵相减，再对得到的矩阵
        # 中每个元素取绝对值
        test_matrix = np.abs((np.tile(test_vector, (len(train_set), 1))
        - xmatrix))

        # 将矩阵中每行元素相加得到测试集数据与训练集的每个数据之间的曼哈顿距离
        distance = np.sum(test_matrix, axis=1).T.tolist()[0]

        maps = {}
        for i in range(len(distance)):
            if distance[i] not in maps.keys():
                maps[distance[i]] = [train_set[i][1]]
            else:
                maps[distance[i]].append(train_set[i][1])

        # 将得到的所有距离排序
        distance.sort()

        # 选取距离最近的 K 个训练对象

```

```

count = 0
mood = []
for index in range(len(distance)):
    i = distance[index]
    if len(maps[i]) + count >= k:
        mood += maps[i]
        break
    else:
        mood += maps[i]
        count += len(maps[i])

# 选取 K 个训练对象的标签中出现次数最多的标签
top = Counter(mood).most_common(1)[0][0]

labels.append(top)
return labels

```

• 创新点&优化

由于计算两个向量之间的距离有多种方式，本次实验我使用曼哈顿距离和欧式距离对比选择最优，试用不同的 K 值来选择最优的。

使用曼哈顿距离时：

```

k = 1 验证集正确率：0.3633440514469453
k = 2 验证集正确率：0.40836012861736337
k = 3 验证集正确率：0.3954983922829582
k = 4 验证集正确率：0.3858520900321543
k = 5 验证集正确率：0.4212218649517685
k = 6 验证集正确率：0.40836012861736337
k = 7 验证集正确率：0.40836012861736337
k = 8 验证集正确率：0.40836012861736337
k = 9 验证集正确率：0.40836012861736337
k = 10 验证集正确率：0.3954983922829582
k = 11 验证集正确率：0.40192926045016075
k = 12 验证集正确率：0.40514469453376206
k = 13 验证集正确率：0.40836012861736337
k = 14 验证集正确率：0.41479099678456594
k = 15 验证集正确率：0.41479099678456594
k = 16 验证集正确率：0.41479099678456594
k = 17 验证集正确率：0.4180064308681672
k = 18 验证集正确率：0.4180064308681672
k = 19 验证集正确率：0.4212218649517685
k = 20 验证集正确率：0.4212218649517685
k = 21 验证集正确率：0.41479099678456594
k = 22 验证集正确率：0.4115755627009646
k = 23 验证集正确率：0.4115755627009646

```

使用欧式距离时：

```
k = 1 验证集正确率: 0.36012861736334406
k = 2 验证集正确率: 0.3987138263665595
k = 3 验证集正确率: 0.3858520900321543
k = 4 验证集正确率: 0.36977491961414793
k = 5 验证集正确率: 0.40192926045016075
k = 6 验证集正确率: 0.3987138263665595
k = 7 验证集正确率: 0.3987138263665595
k = 8 验证集正确率: 0.3987138263665595
k = 9 验证集正确率: 0.3987138263665595
k = 10 验证集正确率: 0.3858520900321543
k = 11 验证集正确率: 0.39228295819935693
k = 12 验证集正确率: 0.3954983922829582
k = 13 验证集正确率: 0.3987138263665595
k = 14 验证集正确率: 0.40514469453376206
k = 15 验证集正确率: 0.40514469453376206
k = 16 验证集正确率: 0.40514469453376206
k = 17 验证集正确率: 0.40836012861736337
k = 18 验证集正确率: 0.40836012861736337
k = 19 验证集正确率: 0.4115755627009646
k = 20 验证集正确率: 0.4115755627009646
k = 21 验证集正确率: 0.40514469453376206
k = 22 验证集正确率: 0.40192926045016075
k = 23 验证集正确率: 0.40192926045016075
```

经过多次实验测试，最终发现使用曼哈顿距离，K=5 时效果最佳。

• 实验结果及分析

1. 实验结果展示示例

对测试集数据进行分类结果如图：

```
16337341_zhuzhiru_KNN_classification.csv
1 Words (split by space),label
2 senator carl krueger thinks ipods can kill you,joy
3 who is prince frederic von anhalt,joy
4 prestige has magic touch,joy
5 study female seals picky about mates,joy
6 no e book for harry potter vii,joy
7 blair apologises over friendly fire inquest,fear
8 vegetables may boost brain power in older adults,surprise
9 afghan forces retake town that was overrun by taliban,fear
10 skip the showers male sweat turns women on study says,surprise
11 made in china irks some burberry shoppers,joy
12 britain to restrict immigrants from new eu members,joy
13 canadian breakthrough offers hope on autism,joy
14 russia to strengthen its military muscle,fear
15 alzheimer s drugs offer no help study finds,joy
16 uk police slammed over terror raid,fear
17 no rejects police state claim,fear
18 oprah announces new book club pick,joy
19 smith can t be buried until hearing,joy
20 african nation hopes whoopi can help,joy
21 tourism lags in bush s hometown,joy
22 air france klm profit rises,joy
23 au regrets sudan s expulsion of un envoy,joy
24 taiwan s mr clean indicted steps down,sad
25 martian life could have evaded detection by viking landers,joy
```

2. 评测指标展示即分析

选择使用曼哈顿距离，K=5 时，验证集的准确率如图所示：

```
k = 5 验证集正确率: 0.4212218649517685
```

• 实验题目

k-NN 处理回归问题

• 实验内容

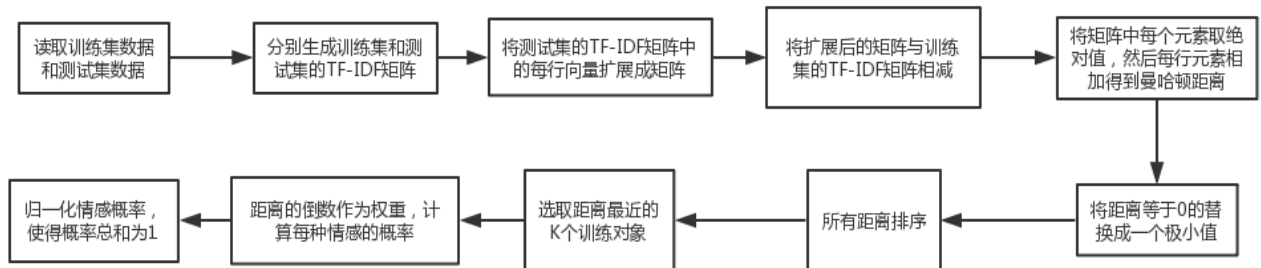
• 算法原理

KNN 算法，即 K 近邻分类算法，根据不同特征值之间的距离来进行分类的一种简单而又懒惰的算法。

KNN 算法的步骤：对于每个测试集中的不带标签的对象，通过使用曼哈顿距离、欧式距离或夹角余弦来计算它与训练集中的每个对象的距离，接着选取距离最近的 K 个训练对象，然后根据距离大小对这 K 个训练对象标签的概率加权求和得到测试对象的标签概率。

在本次实验中，我再次选择曼哈顿距离计算测试集向量与训练集向量之间的距离。与上一个实验不同的是，本次实验我为训练集数据和测试集数据分别生成了一个 TF-IDF 矩阵，使用 TF-IDF 矩阵中的向量计算距离。

• 流程图



• 关键代码截图

(1) 生成 TF-IDF 矩阵的函数：

```
def sub_TF_IDF(test_word_vector, test_set):
    test_TF = []
    test_IDF = [0 for i in range(len(test_word_vector))]
    idf = {}
    for line in test_set:
        words = line[0].split(' ')

        # 生成 TF 矩阵
        test_vector = np.zeros(len(test_word_vector))
        for word in words:
            test_vector[test_word_vector.index(word)] += 1 / len(words)
        test_TF.append(test_vector)
```

```

        # 记录单词在每句文本中出现的次数
        unique = list(set(words))
        for word in unique:
            if word not in idf.keys():
                idf[word] = 1
            else:
                idf[word] += 1

        # 生成 IDF 矩阵
        for key in idf.keys():
            test_IDF[test_word_vector.index(key)] = math.log(len(test_set)
/ (idf[key] + 1))

        # 根据公式生成 TF-IDF 矩阵
        test_TFIDF = test_TF * np.tile(test_IDF, (len(test_set), 1))
        return test_TFIDF

```

(2) KNN 算法实现:

```

def regression_KNN(test_set, k):
    labels = []

    # 生成不重复词向量
    test_word_vector = []
    for row in test_set:
        test_word_vector += row[0].split(' ')
    for row in train_set:
        test_word_vector += row[0].split(' ')
    test_word_vector = list(set(test_word_vector))

    # 生成训练集文本数据的 TF-IDF 矩阵
    train_TFIDF = sub_TF_IDF(test_word_vector, train_set)

    # 生成测试集文本数据的 TF-IDF 矩阵
    test_TFIDF = sub_TF_IDF(test_word_vector, test_set)

    for num in range(len(test_set)):
        test_vector = test_TFIDF[num]

        # 将测试集中一行文本数据的 TF-IDF 向量扩展成矩阵，并与训练集的 TF-IDF
        # 矩阵相减，再取绝对值
        test_matrix = np.abs(np.tile(test_vector, (len(train_set), 1))
- train_TFIDF)

```



```

# 将矩阵中每行元素相加得到测试集数据与训练集的每个数据之间的曼哈顿距离
distance = np.sum(test_matrix, axis=1).T.tolist()

# 将距离等于 0 的替换为一个极小值
for i in range(len(distance)):
    if distance[i] == 0:
        distance[i] = 0.00000000000001

maps = {}
for i in range(len(distance)):
    if distance[i] not in maps.keys():
        maps[distance[i]] = [i]
    else:
        maps[distance[i]].append(i)

# 将得到的所有距离排序
dist = sorted(distance)

count = 0
moods = []
# 选取距离最近的 k 个训练对象
for index in range(len(dist)):
    dd = dist[index]
    if len(maps[dd]) + count >= k:
        moods += maps[dd]
        break
    else:
        moods += maps[dd]
        count += len(maps[dd])

# 计算测试对象的情感概率
sentiments = []
for i in range(1, 7):
    sentiment = 0
    for index in moods:
        sentiment += float(train_set[index][i]) /
distance[index]
    sentiments.append(sentiment)

# 归一化测试对象的情感概率，使得各个概率的总和为 1
summ = sum(sentiments)
for i in range(len(sentiments)):
    sentiments[i] = sentiments[i] / summ

```

```
labels.append(sentiments)
return labels
```

- 创新点&优化

之前在处理训练集和测试集的数据时，像上一个实验一样，分别生成一个 ONE-HOT 矩阵，再使用曼哈顿距离计算向量之间的距离，选取不同的 K 值分别预测验证集，然而最终的平均相关系数都没有高于 0.3，如图所示。

```
k = 1 相关系数: 0.20464471058563802
k = 2 相关系数: 0.2687391524803116
k = 3 相关系数: 0.2709896314851869
k = 4 相关系数: 0.2768189182481399
k = 5 相关系数: 0.3021564851100477
k = 6 相关系数: 0.28919787780216993
k = 7 相关系数: 0.295089339553262
k = 8 相关系数: 0.2890590019288396
k = 9 相关系数: 0.28474547975540626
k = 10 相关系数: 0.2836284851529569
k = 11 相关系数: 0.28031895857130334
k = 12 相关系数: 0.28955056590618483
k = 13 相关系数: 0.2824755023042891
k = 14 相关系数: 0.278988925117308
k = 15 相关系数: 0.27784509842484906
k = 16 相关系数: 0.285042325170962
k = 17 相关系数: 0.27533531136224443
k = 18 相关系数: 0.277440150998496
k = 19 相关系数: 0.28384131233761034
k = 20 相关系数: 0.2783381024242278
k = 21 相关系数: 0.26903570071113264
k = 22 相关系数: 0.27181995990187224
k = 23 相关系数: 0.27083699101861997
```

于是，我决定为训练集和测试集的数据分别生成一个 TF-IDF 矩阵来替代 ONE-HOT 矩阵，最后测试的效果有显著提高。

使用 TF-IDF 矩阵后，再将曼哈顿距离和欧式距离对比，试用不同的 K 值来选择最优。使用曼哈顿距离时：

```
k = 1 相关系数: 0.28969729983982595
k = 2 相关系数: 0.3294775116625027
k = 3 相关系数: 0.31423517625272424
k = 4 相关系数: 0.31411946751119424
k = 5 相关系数: 0.30574527436437066
k = 6 相关系数: 0.30381258023105606
k = 7 相关系数: 0.31006574571393725
k = 8 相关系数: 0.29295157039885183
k = 9 相关系数: 0.2960181406926932
k = 10 相关系数: 0.2987913822072354
k = 11 相关系数: 0.2981043837330668
k = 12 相关系数: 0.3021473977882521
k = 13 相关系数: 0.3017507062428836
k = 14 相关系数: 0.3123037923473683
k = 15 相关系数: 0.3166896023176366
k = 16 相关系数: 0.3113506684016309
k = 17 相关系数: 0.3130507397802984
k = 18 相关系数: 0.30707520646496883
k = 19 相关系数: 0.3087005350028094
k = 20 相关系数: 0.30737418008698864
k = 21 相关系数: 0.3056319756549093
k = 22 相关系数: 0.29849810112037806
k = 23 相关系数: 0.2975435908959683
```

使用欧式距离时：

```
k = 1 相关系数: 0.25091191848575484
k = 2 相关系数: 0.26141392617589543
k = 3 相关系数: 0.2867214602838782
k = 4 相关系数: 0.24915200716856514
k = 5 相关系数: 0.23414199367309965
k = 6 相关系数: 0.22780263987199387
k = 7 相关系数: 0.23626281876912938
k = 8 相关系数: 0.2281019358794417
k = 9 相关系数: 0.22848348590750503
k = 10 相关系数: 0.23500385467727747
k = 11 相关系数: 0.23759959682138607
k = 12 相关系数: 0.25117870095305184
k = 13 相关系数: 0.24595158431064731
k = 14 相关系数: 0.24616502712078855
k = 15 相关系数: 0.22671647131530856
k = 16 相关系数: 0.23410925668376348
k = 17 相关系数: 0.2554153225079412
k = 18 相关系数: 0.26003996036579013
k = 19 相关系数: 0.255537265568
k = 20 相关系数: 0.2556044965858106
k = 21 相关系数: 0.2622488629119173
k = 22 相关系数: 0.258141654892628
k = 23 相关系数: 0.2744654012619818
```

经过多次实验测试，最终发现使用曼哈顿距离，K=2 时效果最佳。

• 实验结果及分析

1. 实验结果展示示例

对测试集数据进行预测结果如图：

textid	anger	disgust	fear	joy	sad	surprise
1	0.052387	0.14885	0.262034	0.040619	0.076844	0.419266
2	0.131536	0.025054	0.081427	0.066351	0.388638	0.306994
3	0	0	0.025835	0.23269	0.40031	0.341165
4	0	0	0	0.7108	0	0.2892
5	0.103668	0.069434	0.08342	0.278034	0.092529	0.372914
6	0.072828	0.088583	0.260397	0	0.421313	0.156879
7	0	0	0.100112	0.435862	0.000949	0.463077
8	0.153362	0	0.398982	0.069495	0.299445	0.078717
9	0	0.14597	0	0.135136	0	0.718893
10	0.015488	0.136697	0.185709	0.249676	0.054159	0.35827
11	0.202281	0	0.041152	0.384306	0.112965	0.259296
12	0.14398	0.002632	0.286662	0.106049	0.225532	0.235146
13	0.196598	0.122402	0.38611	0.049146	0.097487	0.148257
14	0.100027	0.121598	0.111798	0.284902	0.113719	0.267956
15	0.219931	0.209636	0.288269	0	0.282164	0
16	0.157509	0.037854	0.392083	0	0.159539	0.253014
17	0	0	0	0.723829	0	0.276171
18	0.219542	0.041534	0.155751	0.057125	0.315174	0.210874
19	0.109896	0.033683	0.192135	0.212663	0.286779	0.164844
20	0	0	0	0.72166	0	0.27834
21	0.128898	0.07566	0.086849	0.434548	0.148524	0.125521
22	0.103194	0.08897	0	0.072282	0.085951	0.649603
23	0.021123	0	0.140115	0.197728	0.327875	0.313159
24	0.121074	0	0.061109	0	0.119922	0.697895
25	0.028584	0	0.185846	0.496811	0.231591	0.057168

2. 评测指标展示即分析

使用曼哈顿距离，K=2 时验证集结果的相关系数如图所示：

```
k = 2 相关系数: 0.3294775116625027
```

- **思考题**

- 1. 为什么时倒数呢？**

因为距离越近，所对应的概率所占的比重越大。

- 2. 同一测试样本的各个情感概率总和应该为 1，如何处理？**

对各个情感概率进行归一化操作，使得他们的总和为 1。