实验报告

学院：数据科学与计算机学院　　专业：计算机科学与技术　　科目：人工智能

学号：17341213　　姓名：郑康泽

文本数据集简单处理 & KNN（K 最近邻）

一. 算法原理

1. TF-IDF 实验任务：

   TF 矩阵计算公式如下：

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

   IDF 矩阵计算公式如下：

$$idf_i = log\frac{|D|}{1+\{j: t_i \in d_j\}}, |D| 表示文本总数, \{j: t_i \in d_j\}$$

$$表示出现单词i的文本总数,$$

   TF_IDF 矩阵计算公式如下：

$$tfidf_{i,j} = tf_{i,j} \times idf_i ,$$

   $tf_{i,j}$表示单词 *i* 在文本 *j* 中出现的频率，idf_i表示逆向文件频率。

   首先我们要先收集 *semeval.txt* 中的所有单词，然后在遍历每一行文本，计算出该文本中单词出现的频率，同时也要给该单词出现的文本数加一，但是要注意不要重复加了，因为同个文本可能会出现多次同一个单词。

2. KNN 分类：

   KNN 来做分类，就是参考相似度最高的 k 个记录，来预测当前记录的结果。这次作业是提供一定数量的文本以及对应的情感标签的训练集，来预测只有文本的测试集。首先我们要解决如何计算相似度的问题，我们可以生成训练集的 one_hot 矩阵 *matrix* 以及测试集中一个文本的 one_hot 向量$\vec{n}$，然后计算该向量与训练集的 one_hot 矩阵中每一个行向量的距离 *dis(matrix[i], $\vec{n}$)*，得到的距离就是相似度，然后选出相似度最高的 *k* 个记录，参照记录中的情感标签，选出其中的众数，作为该测试文本情感的预测值。

3. KNN 回归：

   KNN 来做回归，原理和上面一样，也是参考相似度最高的 k 个记录，来预测当前记录的结果，不过分类的预测值是离散的，而回归的预测值是连续的。所以对于每一条参考记录，可以利用相似度（相似度的计算同上）作为权重，乘上参考记录的预测值，作为预测值，最后把所有参考记录获得的预测值加起来，就是最后的预测值。

二. 伪代码

1. TF-IDF 实验任务：

Procedure get_tf_idf(*documents*):
input: *documents* which is a set of documents and *document[i]* denotes the i-th document.
output: *tf_idf* which is tf_idf matrix of *documents*.

```
vocabulary <- an empty list;
for every document in documents:
    for every word in document:
        if word not in vocabulary:
            insert word into vocabulary;

len_of_vocabulary <- length of vocabulary;
len_of_documents  <- the number of documents;
tf_idf <- an empty list;
tf <- an empty list;
idf <- an empty list;
for every document in documents:
    tmp <- a list with l 0s;
    num_of_words <- the number of words in documents;
    for every word in document:
        if tmp[the position of word in vocabulary] == 0:
            idf[the position of word in vocabulary] += 1;
        tmp[the position of word in vocabulary] = 1;
    for data in tmp:
        data /= num_of_words;
    insert tmp into tf;
for data in idf:
    data = log(len_of_documents/(1+data))
for data in tf:
    data *= id;          // "/=" denotes point division
return tf_idf;
```

2. KNN 分类：

Procedure KNN-classification(*train_set, test_set*):
input: *train_set* which is a set of documents and corresponding labels and *test_set* which is a set of documents only.
output: *predicition* which the prediction about labels in *test_set*.

```
vocabulary <- an empty list;
```

```
for every document in documents of train_set:
    for every word in document:
        if word not in vocabulary:
            insert word into vocabulary;

one_hot_train <- an empty list;
one_hot_test <- an empty list;
len_of_vacabulary <- length of vacabulary
for every document in documents of train_set:
    tmp <- a list with len_of_vacabulary 0s;
    for every word in document:
        if word in vocabulary:
            tmp[the position of word in vocabulary] = 1;
    insert tmp into one_hot_train;
for every document in documents of test_set:
    tmp <- a list with len_of_vacabulary 0s;
    for every word in document:
        if word in vocabulary:
            tmp[the position of word in vocabulary] = 1;
    insert tmp into one_hot_test;

prediction <- an empty list;
for every row1 in one_hot_test:
    distance <- an empty list;
    for every row2 in one_hot_train:
        tmp <- calculate_similarity between row1 and row2
        insert tmp into distance;
    labels <- select top k high similarity in distance and get their labels;
    label <- select the mode in labels;
    insert label into prediction;
return prediction;
```

3. KNN 回归：

Procedure KNN-classification(*train_set*, *test_set*):
input: *train_set* which is a set of documents and corresponding labels and *test_set* which is a set of documents only.
output: *predicition* which the prediction about labels in *test_set*.

```
vocabulary <- an empty list;
for every document in documents of train_set:
    for every word in document:
        if word not in vocabulary:
            insert word into vocabulary;
```

```
one_hot_train <- an empty list;
one_hot_test <- an empty list;
len_of_vacabulary <- length of vacabulary
for every document in documents of train_set:
    tmp <- a list with len_of_vacabulary 0s;
    for every word in document:
        if word in vocabulary:
            tmp[the position of word in vocabulary] = 1;
    insert tmp into one_hot_train;
for every document in documents of test_set:
    tmp <- a list with len_of_vacabulary 0s;
    for every word in document:
        if word in vocabulary:
            tmp[the position of word in vocabulary] = 1;
    insert tmp into one_hot_test;

prediction <- an empty list;
for every row1 in one_hot_test:
    distance <- an empty list;
    for every row2 in one_hot_train:
        tmp <- calculate_similarity between row1 and row2
        insert tmp into distance;
    labels <- select top k high similarity in distance and get their labels;
    pre <- 0;
    for label in labels:
        pre += label * corresponding distance in distance;
    insert pre into prediction;
return prediction;
```

三. 代码截图

1. TF_IDF 矩阵：

1）生成 TF_IDF 矩阵：

先收集出现的词汇，然后再遍历一遍，制作 TF 和 IDF 矩阵，之后就能制作 TF_IDF 矩阵。

```
def generate(read_path, save_path):
    vocabulary = []
    record = {}
    tf = []
    idf = {}
    article_num = 0

    f = open(read_path, 'r')
    # 收集词汇和文章数量
    for line in f:
        article_num += 1
        value = line.split()
        # print(value[8:])
        for word in value[8:]:
            if word.lower() not in vocabulary:
                vocabulary.append(word.lower())
    # print(vocabulary)
    f.seek(0)
```

```
    # 创建统计表
    for word in vocabulary:
        record[word] = 0
        idf[word] = 0

    # 创建tf和idf
    for line in f:
        num = 0
        value = line.split()
        for word in value[8:]:
            num += 1
            if record[word.lower()] == 0:
                idf[word.lower()] += 1
            record[word.lower()] += 1
        for word in value[8:]:
            record[word.lower()] /= float(num)
        tf.append(copy.deepcopy(record))
        for word in value[8:]:
            record[word.lower()] = 0
    f.close()
    for word in vocabulary:
        idf[word] = np.log10(article_num / (1.0 + idf[word]))
```

```
    # 写文件
    f = open(save_path, 'w')
    for rec in tf:
        for word in vocabulary:
            f.write(str(rec[word] * idf[word]))
            f.write(' ')
        f.write('\n')
    f.close()
```

2. KNN 分类：

1) 预测函数：

利用距离公式，算出对于训练集中每一条记录的距离，之后取出相似度最高的 k 条记录，进行参照，获得预测值。

```
def predict(sentence, matrix, k, tf_idf_train, tf_idf_validation):
    '''
    :param sentence: 预测文本
    :param matrix: 训练集的one_hot
    :param k: 取k个最相似
    :param tf_idf_train: 训练集的tf_idf矩阵
    :param tf_idf_validation: 测试集tf_idf向量（就是对应的那一条行）
    :return: 预测值
    '''
    record = [0 for i in range(len(VOCABULARY))]
    rank = [0 for i in range(len(matrix))]
    prediction = {}
    # 制作测试集的ont_hot
    words = sentence.split()
    for word in words:
        if word.lower() in VOCABULARY:
            record[VOCABULARY.index(word.lower())] = 1
    # 特殊情况
    if np.linalg.norm(np.array(record)) == 0:
        return 'joy'
```

```
    # 与训练集求距离
    for i in range(len(matrix)):
        array1 = np.array(record)
        array2 = np.array(matrix[i][:-1])
        array3 = np.array(tf_idf_train[i])
        # rank[i] = np.sqrt(np.sum(np.square(array1 - array2)))
        rank[i] = np.dot(array1 * tf_idf_validation, array2 * array3) / (np.linalg.norm(array1 * tf_idf_validation) * np.linal
        # rank[i] = np.dot(record * tf_idf_validation, tf_idf_train[i] * array2) / (
        #           np.linalg.norm(record * tf_idf_validation) * np.linalg.norm(tf_idf_train[i] * array2))
    # 取k个最相似的
    rank = np.array(rank)
    rank = (rank - np.mean(rank)) / (max(rank) - min(rank))
    top_k = np.argsort(-rank)[0: k]
    for index in top_k:
        key = matrix[index][-1]
        if key not in prediction.keys():
            # prediction[key] = 1.0 / (0.001 + rank[i][1])
            prediction[key] = 1.0 * rank[index]         # 加权
        else:
            prediction[key] += 1.0 * rank[index]        # 加权
            #prediction[key] += 1.0 / (0.001 + rank[i][1])
```

```
    # 排序找最可能的
    prediction = sorted(prediction.items(), key=lambda x: x[1], reverse=True)
    # print(prediction)
    # print()
    return prediction[0][0]
```

2）生成 one_hot 的函数：

首先获得所有词汇的列表，然后再遍历一遍制作 one_hot 矩阵。

```
def generate_matrix(path):
    matrix = []

    f = open(path, 'r')
    lines = f.readlines()
    for line in lines[1:]:
        record = [0 for i in range(len(VOCABULARY) + 1)]    # 零列表
        value = line.split(',')
        words = value[0].split()
        for word in words:
            record[VOCABULARY.index(word.lower())] = 1      # 出现就置一
        record[-1] = value[1].split()[0]
        matrix.append(record)                               # 加入一行的one_hot

    return matrix
```

3. KNN 回归：

1) 预测函数：

利用距离公式，算出对于训练集中每一条记录的距离，之后取出相似度最高的 k 条记录，进行参照，获得预测值。

```
def prediction(matrix, tf_idf_train, tf_idf_validation, path, k, istest):
    '''
    :param matrix: 训练集的one_hot
    :param tf_idf_train: 训练集的tf_idf矩阵
    :param tf_idf_validation: 测试机的tf_idf矩阵
    :param path: 测试集的路径
    :param k: 取k相似
    :param istest: 输入的是预测集集还是验证集
    :return: 返回预测值
    '''
    res = []

    f = open(path, 'r')
    lines = f.readlines()
    j = 0
    for line in lines[1:]:
        one_hot = [0 for i in range(len(VOCABULARY))]
        rank = [0 for i in range(len(matrix))]
```

```
        # 制作测试集的one_hot
        value = line.split(',')
        if istest:
            words = value[1].split()
        else:
            words = value[0].split()
        for word in words:
            if word.lower() in VOCABULARY:
                one_hot[VOCABULARY.index(word.lower())] = 1
        one_hot = np.array(one_hot) + 0.001          #防止为0
        # 计算与训练集的距离
        for i in range(len(matrix)):
            array_1 = np.array(matrix[i][:len(VOCABULARY)])
            array_2 = np.array(tf_idf_train[i])
            array_3 = np.array(tf_idf_validation[j])
            if np.sum(array_3) != 0:
                rank[i] = 1 - (np.dot(one_hot * array_3, array_1 * array_2) /
                               (np.linalg.norm(one_hot * array_3)*np.linalg.norm(array_1 * array_2)))
            else:
                rank[i] = 1 - (np.dot(one_hot, array_1 * array_2) /
                               (np.linalg.norm(one_hot)*np.linalg.norm(array_1 * array_2)))
```

```
        # 排序后找k个最相似的
        top_k = np.argsort(np.array(rank))[0: k]
        tmp = [0 for i in range(len(MOOD))]
        for index in top_k:
            train_mood = matrix[index][len(VOCABULARY): len(VOCABULARY)+len(MOOD)]
            for i in range(len(MOOD)):
                tmp[i] += train_mood[i] / (0.001 + rank[index])      # 加权
        # 处理后概率相加为1
        s = sum(tmp)
        for i in range(len(MOOD)):
            tmp[i] /= s
        res.append(tmp)
        j += 1
    return res
```

2) 计算相关系数：

利用相关系数的公式

$$\gamma = \frac{Cov(X,Y)}{Var(X)\times Var(Y)},$$

即可算出答案

```python
def cal_correl(res, path):
    '''
    :param res: 预测值矩阵
    :param path: 验证集的路径
    :return: 相关系数
    '''
    correlation_coefficient = []

    f = open(path, 'r')
    lines = f.readlines()
    for l in range(0, len(lines[1:])):
        answer = [0 for i in range(len(MOOD))]
        value = lines[l+1].split(',')
        for i in range(len(MOOD)):
            answer[i] = float(value[1 + i])          # 准确答案
```

```python
        per_res = res[l]
        correlation_coefficient.append(
            np.mean((answer - np.mean(answer)) * (per_res - np.mean(per_res))) / np.sqrt(np.var(answer) * np.var(per_res)))
        # 相关系数的公式
    return np.mean(correlation_coefficient)
```

四. 创新点

其实准确率高低最重要的是距离公式的正确程度，所以我一直尝试不同的距离公式，发现 L_p 距离公式无论是几次方，都没有较好的准确率或者相关系数，然后我们尝试用余弦公式，获得的结果比 L_p 公式好了许多，但是我还是想再提高，就想到了课件前面提到的 tf-idf 矩阵，如何去利用 tf-idf 矩阵呢？我也尝试了许多方法，最后发现直接点乘上 one_hot 能获得较好的结果，距离公式如下：

$$distance = \frac{\vec{a} \times \vec{b}}{|\vec{a}| \times |\vec{b}|},$$

$$其中\vec{a} = \overrightarrow{one\_hot\_train} \cdot \overrightarrow{tf\_idf\_train}, \overrightarrow{one\_hot\_test} \cdot \overrightarrow{tf\_idf\_test}.$$

五. 实验结果以及分析

1. 结果展示和分析：

   1）KNN 分类：

   预测结果如下：

```
预测结果分布如下：
anger: 6
disgust: 3
fear: 54
joy: 153
sad: 60
surprise: 35
准确率如下：
46.623794%
```

数据分布还是比较正常，不会出现只猜测一种的情况，但是猜测 joy 的次数还是挺多的，有可能是我预测方式导致的，因为我判断如果测试文本中没有出现训练文本中任何词的时候，就预测是 joy。

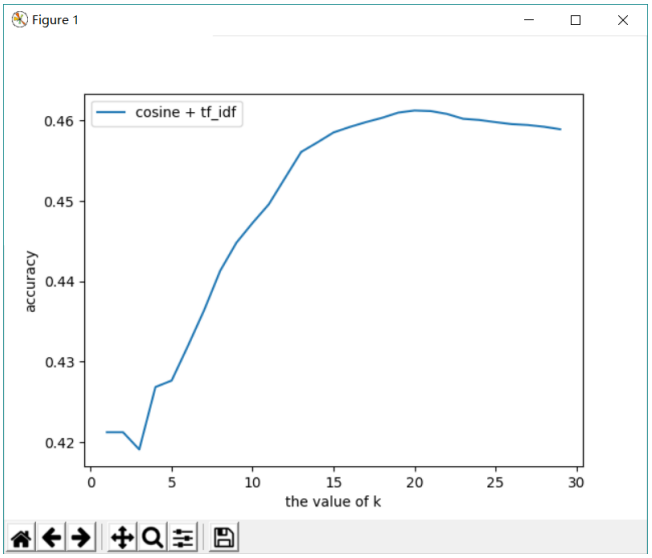
2) KNN 回归：

```
预测结果如下：
预测前10条所得到的结果与准确结果的相关系数：
0 61.491282%
1 57.123397%
2 77.990666%
3 67.479320%
4 43.862784%
5 10.314957%
6 91.652417%
7 87.349938%
8 82.715592%
9 22.970035%
平均相关系数如下：
48.536718%
```

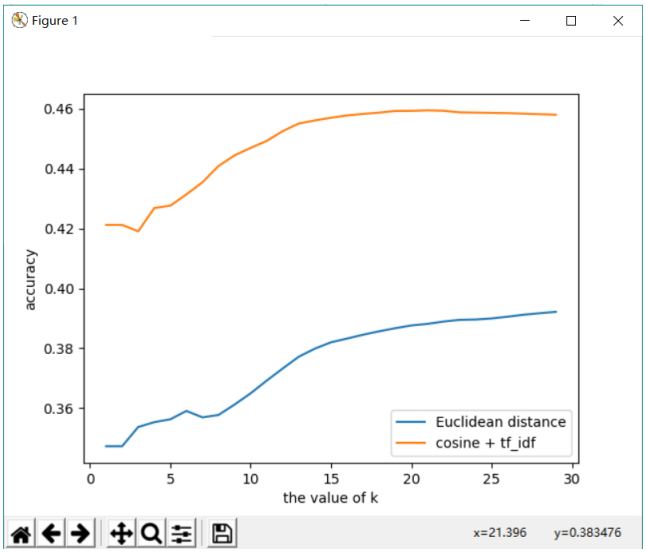
第 4、5、9 条的预测结果都比较差，可能原因是其中出现的单词在训练集中没有出现，导致预测结果较为差。KNN 的缺点就是如此，如果没有可以参照的，预测结果不会很好。

2. 模型性能展示和分析：

1）KNN 分类：

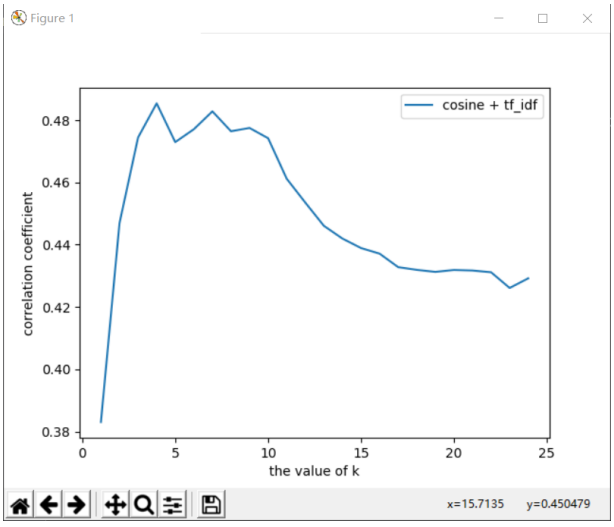


从图中可以看出，k 值大概为 20 左右的时候，准确率能达到最高；

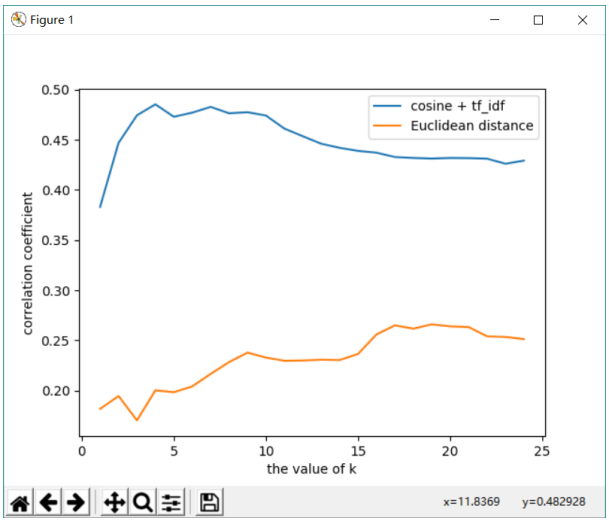


从图中可以看出利用 tf-idf 矩阵优化后的余弦相似度预测的准确率远高于直接用欧式距离的准确率，所以新的距离公式效果还是挺好的

2）KNN 回归：



从图中可以看出，k 值大概为 4 时，相关系数达到最高；



从图中可以看出，新的距离公式得到的相关系数比直接用欧氏距离得到的相关系数高得多。

六. 思考题

1. IDF 的第二个计算公式中分母多了个1是为什么？
   答：为了防止分母为 0 的情况，因为有可能一个单词在所有的文章中都没有出现过。

2. IDF 数值有什么含义？TF-IDF 数值有什么含义？

   答：IDF 数值代表某个单词对于某篇文章的代表程度，因为数值越小，说明分母越小，说明出现该单词的文章数越小，那么该单词就能代表性越强。TF-IDF 数值是 TF 与 IDF 的乘积，代表着该单词对于该文章的重要程度，频率越高，出现的文章数越少，说明单词对于该文章的重要程度越强。
3. 为什么是倒数？
   答：因为距离越小代表两者的相似度越高，相似度越高说明参考价值越高，即权重越大，所以就取距离的倒数作为权值。