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 秋季学期）

教学班：教务 2 班

专业：计算机科学与技术（超算方向）

学号：16337113

姓名：劳马东

2018 年 9 月 18 日

1 实验题目

a. 给定一个含有多篇文本的文本集，建立对应的 TF-IDF 矩阵。

b. 给定一个含有多篇文本的文本集（包括训练集和测试集），使用 KNN 算法解决测试文本的分类和回归问题。

2 实验内容

2.1 算法原理

2.1.1 TF-IDF 矩阵

- TF 词频（Term Frequency）表示词语 w 在文本 d 中出现的频率。例如对于文本集合 D:“苹果 手机 好用 销售”,“市民 买 手机 手机”,“市民 觉得 苹果 手机 贵 好用”，构建词汇表：

$$W = \begin{array}{|c|c|c|c|c|c|c|c|} \hline 贵 & 好用 & 觉得 & 买 & 苹果 & 市民 & 手机 & 销售 \\ \hline \end{array} \tag{1}$$

其 TF 矩阵为：

$$tf = \begin{bmatrix} 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{1}{4} \\ 0 & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{4} & \frac{2}{4} & 0 \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & 0 \end{bmatrix} \tag{2}$$

- IDF 逆向文件频率 (Inverse Document Frequency) 是一个词语在文本集合 D 普遍重要性的度量。对于 D 中的每个词语 w_i，其 idf 计算公式为：

$$\underset{D_j \in D}{idf_i} = \log \frac{|D|}{|\{j : w_i \in D_j\}|} \tag{3}$$

D_j 是 D 中的文本。按照公式，对于上文提到的文本集合，其 IDF 向量为：

$$idf = \begin{bmatrix} \log \frac{3}{1} & \log \frac{3}{2} & \log \frac{3}{1} & \log \frac{3}{1} & \log \frac{3}{2} & \log \frac{3}{2} & \log \frac{3}{3} & \log \frac{3}{1} \end{bmatrix} \tag{4}$$

- TF-IDF 是一个词语的分类能力的度量，其计算公式为：

$$\underset{w_i \in W, D_j \in D}{tfidf_{ij}} = tf_{ij} \times idf_i \tag{5}$$

对于文本 D_j 中的词语 w_i，它在 D_j 中出现的次数越多，tf 就越大；出现在文本集合中的文本数越少，即 $|\{j : w_i \in D_j\}|$ 越小，idf 就越大，因此 tfidf 就越大。值得注意的是，一个词语在某篇文本中集中出现，说明这个词语能很好地把这篇文本和其他文本区分开。TF-IDF 矩阵为：

	贵	好用	觉得	买	苹果	市民	手机	销售
训练文本1	0	(1/4)*log(3/2)	0	0	(1/4)*log(3/2)	0	(1/4)*log(3/3)	(1/4)*log(3/1)
训练文本2	0	0	0	(1/4)*log(3/1)	0	(1/4)*log(3/2)	(2/4)*log(3/3)	0
训练文本3	(1/6)*log(3/1)	(1/6)*log(3/2)	(1/6)*log(3/1)	0	(1/6)*log(3/2)	(1/6)*log(3/2)	(1/6)*log(3/3)	0

图 1: TF-IDF 矩阵

2.1.2 KNN 算法

KNN 的基本思想是，一个样本的特性与它在样本空间中的 k 个最接近的邻居的特性相近。对于分类问题，k 个邻居的特性就是大多数邻居所属的类别；对于回归问题，为了表达这种相似性，就用加权的方法计算概率，越接近的就越相似。

2.2 关键代码截图

如图 2、3、4。

```
def __init__(self, documents, sep=' '):
    # 按出现顺序存储每个单词的idf值，以便在对应tf矩阵每个元素
    self._idf = collections.OrderedDict()
    for i, document in enumerate(documents):
        for word in self._to_words(document, sep):
            # 存储每个单词所出现的文本的集合（不重复）
            self._idf.setdefault(word, set())
            self._idf[word].add(i)
    i += 1
    # idf计算公式
    for word, exists in self._idf.items():
        self._idf[word] = math.log(i / (1 + len(exists)))
```

图 2: 计算 idf 向量

```
def get_tf_idf(self, document, sep=' '):
    counter = collections.Counter()
    words = self._to_words(document, sep)
    # 统计每个单词在document文本中出现的次数
    for word in words:
        counter[word] += 1
    total = len(words)
    # tf-idf计算公式
    for word in counter:
        if word in self._idf:
            counter[word] = counter[word] / total * self._idf[word]
        else:
            # 用0.000001是防止出现零向量（余弦距离不支持）
            counter[word] = 0.000001
    return counter
```

图 3: 计算 tf-idf 向量

```python
def predict(self, test_x):
    # (余弦距离，label)二元组的最小堆
    k_neighbors = PriorityQueue()
    for other_x, y in self._trained_datas:
        dis = self._distance_policy(test_x, other_x)
        if math.fabs(dis - self._distance_policy(other_x, other_x)) < 0.000001:
            return y
        k_neighbors.put((dis, y))
        # 至多存储k个邻居
        if len(k_neighbors) > self._k:
            k_neighbors.get()
    # 从k个最近邻居中决定测试样本的label的策略，用户代码中自定义
    # 好处在于可以针对不同类型问题（如分类和回归），不同加权方法得到label
    label = self._label_policy(k_neighbors)
    return label
```

图 4: KNN 预测

2.3 创新点 & 优化

2.3.1 TF-IDF 矩阵优化

不难发现，TF-IDF 矩阵是一个很大的稀疏矩阵，如果把它完完全全建出来，不仅浪费内存，还会严重影响存取速度, 复杂度是 $O(n^2)$。因此，只需要存储其中的非零元素，考虑用 dict 来替换列表，即 TF-IDF 矩阵是 dict 的列表，这样时间复杂度就变成了 $O(n\log(n))$。

2.3.2 距离优化

实验过程中一共使用了三种距离：曼哈顿距离、欧氏距离和余弦距离。使用前两种时分类的准确率在 0.38 左右，回归的相关系数在 0.29 左右；而使用余弦距离分类的准确率提高到了 0.42，回归的相关系数提高到 0.38。

需要特别注意的是，前两种距离衡量的是两个样本之间的相异度，越大表示越不同，余弦距离表示相似度，越大表示越相同。因此，回归计算概率时应该乘余弦距离而不是除，或者除以余弦的负数。此外，为了防止出现负概率，可以给余弦加 1。

$$\begin{aligned} P(test1\ is\ happy) =& (train1\ probability) \times [cos(train1, test1) + 1] \\ &+ (train2\ probability) \times [cos(train2, test1) + 1] \\ &+ (train3\ probability) \times [cos(train3, test1) + 1] \end{aligned} \tag{6}$$

$$P(test1\ is\ happy) = \frac{train1\ probability}{[-cos(train1, test1) + 1]} + \frac{train2\ probability}{[-cos(train2, test2) + 1]} + \frac{train3\ probability}{[-cos(train3, test1) + 1]} \tag{7}$$

3 实验结果及分析

3.1 实验结果展示

图 5、6 是计算 tf-idf 矩阵的测试文本和结果；分类和回归的正确性个人觉得不能用小测试集证明，直接在验证集上计算正确率即可，因此放在” 评测指标展示” 部分。

```
1   all:148 anger:22 disgust:2 fear:60 joy:0 sad:64 surprise:0  apple phone gooduse sale
2   all:148 anger:22 disgust:2 fear:60 joy:0 sad:64 surprise:0  shimin buy phone phone
3   all:148 anger:22 disgust:2 fear:60 joy:0 sad:64 surprise:0  shimin juede apple phone expensive gooduse
```

图 5: tf-idf 测试文本

```
0    -0.0719205   0    0.101366     0    0    0    0
0    -0.143841    0    0    0    0.101366     0    0
0    -0.047947    0    0    0    0    0.0675775    0.0675775
```

图 6: tf-idf 矩阵计算

3.2 评测指标展示

```
with open(output, 'w') as file:
    results = iter(test(knn, idf, prefix + "validation_set.csv"))
    print(*next(results), sep=',', file=file)
    for x, y in results:
        print(x, *y, sep=',', file=file)
f1 = open(output).readlines()
f2 = open(prefix + "validation_set.csv").readlines()
cnt = 0
total = len(f1)
for i in range(total):
    if f1[i] == f2[i]:
        cnt += 1
print(cnt / total)
```

图 7: 分类-统计正确率的方法

```
0.42628205128205127

Process finished with exit code 0
```

图 8: 优化后-余弦距离分类正确率

	anger	disgust	fear	joy	sad	surprise
r	0.273181313	0.297426484	0.466763263	0.426385154	0.409057324	0.432165655
average	0.384163199					
evaluation	低度相关 666					

图 9: 优化后-余弦距离回归相关系数

4 思考题

在算法原理部分已经分点回答。