数据挖掘与最优化: Assignment 2

组长:	231502004 李子	完成内容	T4	贡献度	20%
组员:	221300066 李千焜	完成内容	T3, T5	贡献度	20%
组员:	221098024 李瑛琦	完成内容	T6	贡献度	20%
组员:	221098145 李傲雪	完成内容	T1	贡献度	20%
组员:	221098071 单佳仪	完成内容	T2	贡献度	20%

目录

实验进度

我们完成了所有内容。

T1

将所有大写字母变成小写，将每条文本去除停用词后进行分词，并计算每个词的 tf-idf 值

```
import string
import numpy as np
import itertools
import pprint

# 停用词表加载
def get_stopword_list():
    # 停用词表存储路径，每一行为一个词，按行读取进行加载
    # 进行编码转换确保匹配准确率
    stop_word_path = 'D:\Personal\Desktop\stopwords.txt'
    stopword_list = [sw.replace('\n', '') for sw in open(stop_word_path,encoding='utf-8').
    readlines()]
    return stopword_list

# 去除停用词
def word_filter(seg_list):
    stopword_list = get_stopword_list()
    filter_list = []
    for seg in seg_list:
        # 过滤停用词表中的词，以及长度为<2的词
        if not seg in stopword_list:
            filter_list.append(seg)
    return filter_list

# 处理数据，得到文档集
def load_data(document):
    doc_list = []
    for text in document:
        text=text.lower() # 小写
        text = text.translate(str.maketrans('', '', string.punctuation)) # 去除标点
        seg_list=text.strip().split()
        filter_list = word_filter(seg_list) # 去除停用词
        doc_list.append(filter_list)
    return doc_list

```

```python
# idf值
def train_idf(doc_list):
    idf_dic = {}
    # 总文档数
    tt_count = len(doc_list)

    # 每个词出现的文档数
    for doc in doc_list:
        for word in set(doc):
            idf_dic[word] = idf_dic.get(word, 0.0) + 1.0

    # 按公式转换为idf值，分母加1进行平滑处理
    for k, v in idf_dic.items():
        idf_dic[k] = np.log(tt_count / (1.0 + v))

    # 对于没有在字典中的词，默认其仅在一个文档出现，得到默认idf值
    default_idf = np.log(tt_count / (1.0))
    return idf_dic, default_idf

# tf值
def get_tf_dic(filter_list):
    tf_dic = {}
    for word in filter_list:
        tf_dic[word] = tf_dic.get(word, 0.0) + 1.0

    tt_count = len(filter_list)
    for k, v in tf_dic.items():
         tf_dic[k] = v / tt_count

    return tf_dic

# tf-idf值
def get_tfidf(doc_list,filter_list):
    idf_dic, default_idf = train_idf(doc_list)
    tf_dic = get_tf_dic(filter_list)
    tfidf_dic = {}
    for word in filter_list:
        idf = idf_dic.get(word, default_idf)
        tf = tf_dic.get(word, 0)

        tfidf = tf * idf
        tfidf_dic[word] = tfidf
    return tfidf_dic

```

```python

# 原始文本数据
documents = [
    "My dog has flea problems, please help.",
    "Maybe not take him to park.",
    "My dog is so cute and I love it.",
    "Stop posting stupid garbage.",
    "mr licks ate my steak.",
    "how to stop him.",
    "quit buying worthless dog food, stupid."
]

doc_list=load_data(documents)
filter_list=list(itertools.chain.from_iterable(doc_list))
tfidf_dic=get_tfidf(doc_list,filter_list)
# 计算最长的键的长度
max_key_length = max(len(key) for key in tfidf_dic.keys())
for key, value in tfidf_dic.items():
    # 格式化输出，使每个键后面的值对齐
    print(f"{key:<{max_key_length}}: {value}")
```

```
'ate'          : 0.0695979426941871
'buying'       : 0.0695979426941871
'cute'         : 0.0695979426941871
'dog'          : 0.0932692978923711
'flea'         : 0.0695979426941871
'food'         : 0.0695979426941871
'garbage'      : 0.0695979426941871
'licks'        : 0.0695979426941871
'love'         : 0.0695979426941871
'park'         : 0.0695979426941871
'posting'      : 0.0695979426941871
'quit'         : 0.0695979426941871
'steak'        : 0.0695979426941871
'stupid'       : 0.0941442067096893
'worthless'    : 0.0695979426941871
```

T2

我们通过如下代码，导入了 41 页的矩阵 A：

```
import numpy as np
A = np.array([[1,0,0,0],[0,0,0,4],[0,3,0,0],[0,0,0,0],[2,0,0,0]])
```

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix} \tag{1}$$

然后通过如下代码，定义截断奇异值分解函数 TSVD：

```
def TSVD(k):
    U,S,Vt = np.linalg.svd(A)
    U_k = U[:,:k]
    S_k = np.diag(S[:k])
    Vt_k = Vt[:k,:]
    A_k = U_k @ S_k @ Vt_k
    print("U_k:",U_k)
    print("S_k:",S_k)
    print("Vt_k:",Vt_k)
    print("A_k:",A_k)
```

利用函数，分别完成截断阶数 k=2，1 的截断奇异值分解：

```
TSVD(2)
TSVD(1)
```

k=2 时，可得到结果：

$$U_k = \begin{bmatrix} 0. & 0. \\ -1. & 0. \\ 0. & -1. \\ 0. & 0. \\ 0. & 0. \end{bmatrix} \tag{2}$$

$$S_k = \begin{bmatrix} 4. & 0. \\ 0. & 3. \end{bmatrix} \tag{3}$$

$$Vt_k = \begin{bmatrix} -0. & -0. & -0. & -1. \\ -0. & -1. & -0. & -0. \end{bmatrix} \tag{4}$$

$$A_k = \begin{bmatrix} 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 4. \\ 0. & 3. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \end{bmatrix} \tag{5}$$

k=1 时，可得到结果：

$$U_k = \begin{bmatrix} 0. \\ -1. \\ 0. \\ 0. \\ 0. \end{bmatrix} \tag{6}$$

$$S_k = \begin{bmatrix} 4. \end{bmatrix} \tag{7}$$

$$Vt_k = \begin{bmatrix} -0. & -0. & -0. & -1. \end{bmatrix} \tag{8}$$

$$A_k = \begin{bmatrix} 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 4. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \\ 0. & 0. & 0. & 0. \end{bmatrix} \tag{9}$$

显然有截断阶数 k=2 时的截断奇异值分解的近似效果更好。

T3

计算两个相同长度向量的余弦相似度的代码如下：

```
import math

def cosine_similarity(vector_a, vector_b):

    # 检查两个向量是否相同长度
    if len(vector_a) != len(vector_b):
        raise ValueError("两个向量必须具有相同的长度")

    epsilon = 1e-9

    # 计算模长平方和
    sum_a = sum(x ** 2 for x in vector_a)
    sum_b = sum(x ** 2 for x in vector_b)

    # 处理两个零向量的情况
    if sum_a < epsilon and sum_b < epsilon:
        return 1.0
    # 处理一个零向量的情况
    elif sum_a < epsilon or sum_b < epsilon:
        return 0.0

    # 计算点积
    dot_product = sum(a * b for a, b in zip(vector_a, vector_b))

    # 计算模长
    magnitude_a = math.sqrt(sum_a)
    magnitude_b = math.sqrt(sum_b)

    # 计算余弦相似度
    return dot_product / (magnitude_a * magnitude_b)
```

示例测试与输出：

```
# 示例1：相同向量
vec1 = [1, 2, 3]
vec2 = [1, 2, 3]
print(cosine_similarity(vec1, vec2))  # 输出：1.0

# 示例2：正交向量
vec3 = [1, 0]
vec4 = [0, 1]
```

```python
print(cosine_similarity(vec3, vec4))  # 输出: 0.0

# 示例3: 零向量
vec5 = [0, 0, 0]
vec6 = [0, 0, 0]
print(cosine_similarity(vec5, vec6))  # 输出: 1.0

# 示例4: 部分零向量
vec7 = [0, 0]
vec8 = [1, 1]
print(cosine_similarity(vec7, vec8))  # 输出: 0.0


# 示例5: 一般向量
vec9 = [1, 2, 3, 4]
vec10 = [ 5, 6, 7, 8]
print(cosine_similarity(vec9, vec10))  # 输出: 0.9688639316269662
```

T4

(1) 输入层的每个 x_j 会与第一个隐藏层的每个节点 $z_i^{(1)}$ 通过加权和相连。对于第一个隐藏层的第 i 个神经元 $z_i^{(1)}$，其计算公式为：

$$z_i^{(1)} = f\left(\sum_{j=1}^{p} w_{ij}^{(1)} x_j + b_i^{(1)}\right)$$

其中，$w_{ij}^{(1)}$ 是输入层到第一隐藏层的权重，$b_i^{(1)}$ 是偏置项，f 是激活函数。

第二个隐藏层的每个节点 $z_i^{(2)}$ 由第一层的输出加权和计算而来。对于第二个隐藏层的第 i 个神经元 $z_i^{(2)}$，其计算公式为：

$$z_i^{(2)} = f\left(\sum_{j=1}^{m} w_{ij}^{(2)} z_j^{(1)} + b_i^{(2)}\right)$$

其中，$w_{ij}^{(2)}$ 是第一隐藏层到第二隐藏层的权重，$b_i^{(2)}$ 是偏置项。

输出层的每个预测值 $\hat{y}_i$ 是由第二个隐藏层的每个节点 $z_i^{(2)}$ 经过加权和得到的。对于第 i 个输出值 $\hat{y}_i$，其计算公式为：

$$\hat{y}_i = \sum_{j=1}^{q} w_{ij}^{(3)} z_j^{(2)} + b_i^{(3)}$$

其中，$w_{ij}^{(3)}$ 是第二隐藏层到输出层的权重，$b_i^{(3)}$ 是输出层的偏置项。

因此，输出值 $\hat{y}_i$ 可以表示为输入值 $(x_1, x_2, \ldots, x_p)$ 的函数形式，如下所示：

$$\hat{y}_i = \sum_{j=1}^{q} w_{ij}^{(3)} f\left(\sum_{k=1}^{m} w_{jk}^{(2)} f\left(\sum_{l=1}^{p} w_{kl}^{(1)} x_l + b_k^{(1)}\right) + b_j^{(2)}\right) + b_i^{(3)}$$

(2)

- **输入层到第一个隐藏层的权重和偏置**：- 权重的数量：输入层有 p 个节点，第一个隐藏层有 m 个节点，因此权重参数个数为 $p \times m$。- 偏置的数量：第一个隐藏层有 m 个节点，因此偏置参数个数为 m。

- **第一个隐藏层到第二个隐藏层的权重和偏置**：- 权重的数量：第一个隐藏层有 m 个节点，第二个隐藏层有 q 个节点，因此权重参数个数为 $m \times q$。- 偏置的数量：第二个隐藏层有 q 个节点，因此偏置参数个数为 q。

- **第二个隐藏层到输出层的权重和偏置**：- 权重的数量：第二个隐藏层有 q 个节点，输出层有 k 个节点，因此权重参数个数为 $q \times k$。- 偏置的数量：输出层有 k 个节点，因此偏置参数个数为 k.

因此，总的未知参数个数为：

$$\text{总参数个数} = (p \times m) + m + (m \times q) + q + (q \times k) + k$$

(3)

损失函数为平方损失：

$$L = \frac{1}{2}(\hat{y} - y)^2$$

其中，y 是真实标签，$\hat{y}$ 是输出层的预测值。

1. **输出层的误差**：首先计算输出层的误差项 $\delta^{(3)}$（对应于输出层的偏置 $b^{(3)}$）：

$$\delta^{(3)} = \hat{y} - y$$

2. **第二个隐藏层的误差**：接着，计算第二个隐藏层的误差项 $\delta_j^{(2)}$（对应于第二个隐藏层的偏置 $b_j^{(2)}$）：

$$\delta_j^{(2)} = \delta^{(3)} w_j^{(3)} f'(z_j^{(2)})$$

其中，$f'(z_j^{(2)})$ 是第二个隐藏层激活函数的导数，$w_j^{(3)}$ 是第二个隐藏层到输出层的权重。

3. **第一个隐藏层的误差**：然后，计算第一个隐藏层的误差项 $\delta_i^{(1)}$（对应于第一个隐藏层的偏置 $b_i^{(1)}$）：

$$\delta_i^{(1)} = \sum_{j=1}^{q} \delta_j^{(2)} w_{ij}^{(2)} f'(z_i^{(1)})$$

其中，$w_{ij}^{(2)}$ 是第一个隐藏层到第二个隐藏层的权重，$f'(z_i^{(1)})$ 是第一个隐藏层激活函数的导数。

4. **梯度计算**：最后，计算损失函数对两个隐藏层偏置的梯度。- 对于第一个隐藏层的偏置 $b_i^{(1)}$：

$$\frac{\partial L}{\partial b_i^{(1)}} = \delta_i^{(1)}$$

- 对于第二个隐藏层的偏置 $b_j^{(2)}$：

$$\frac{\partial L}{\partial b_j^{(2)}} = \delta_j^{(2)}$$

T5

(i) 对于 S 型函数 $\Gamma(z) = \frac{e^z}{1+e^z}$
首先使用商的导数法则计算 $\Gamma(z)$ 的导数：

$$\Gamma'(z) = \frac{d}{dz}\left(\frac{e^z}{1+e^z}\right) = \frac{e^z(1+e^z) - e^z \cdot e^z}{(1+e^z)^2} = \frac{e^z(1+e^z-e^z)}{(1+e^z)^2} = \frac{e^z}{(1+e^z)^2}$$

接下来验证 $\Gamma(z)(1-\Gamma(z))$：

$$\Gamma(z)(1-\Gamma(z)) = \frac{e^z}{1+e^z}\left(1 - \frac{e^z}{1+e^z}\right) = \frac{e^z}{1+e^z} \cdot \frac{1}{1+e^z} = \frac{e^z}{(1+e^z)^2}$$

因此，$\Gamma'(z) = \Gamma(z)(1-\Gamma(z))$ 成立。
(ii) 对于双曲正切函数 $\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
首先计算 $2\Gamma(2z) - 1$：

$$2\Gamma(2z) - 1 = 2 \cdot \frac{e^{2z}}{1+e^{2z}} - 1 = \frac{2e^{2z}}{1+e^{2z}} - 1 = \frac{2e^{2z} - (1+e^{2z})}{1+e^{2z}} = \frac{e^{2z}-1}{1+e^{2z}}$$

将 $\tanh(z)$ 的表达式转换为指数形式：

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z}-1}{e^{2z}+1}$$

显然，$\tanh(z) = \frac{e^{2z}-1}{e^{2z}+1}$ 与 $2\Gamma(2z)-1$ 的结果一致，因此 $\tanh(z) = 2\Gamma(2z) - 1$ 成立。

T6

$\frac{\partial L}{\partial w2} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial a1} \times \frac{\partial a1}{\partial z1} \times \frac{\partial z1}{\partial w2}$
$= -2(y - \hat{y}) \times w5 \times \Gamma(z1)' \times x2$
$= -2 \times (4 - 1.755) \times 1 \times 0.173 \times 0.5 = -0.388$

$\frac{\partial L}{\partial w3} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial a2} \times \frac{\partial a2}{\partial z2} \times \frac{\partial z2}{\partial w3}$
$= -2(y - \hat{y}) \times w6 \times \Gamma(z2)' \times x1$
$= -2 \times (4 - 1.755) \times 1 \times 0.0214 \times 1 = -0.096$

$\frac{\partial L}{\partial w4} = \frac{\partial L}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial a2} \times \frac{\partial a2}{\partial z2} \times \frac{\partial z2}{\partial w4}$
$= -2(y - \hat{y}) \times w6 \times \Gamma(z2)' \times x2$
$= -2 \times (4 - 1.755) \times 1 \times 0.0214 \times 0.5 = -0.048$

假设 s=0.1
$w2' = w2 - s \times \frac{\partial L}{\partial w2} = 1.5388$
$w3' = w3 - s \times \frac{\partial L}{\partial w3} = 2.3096$
$w4' = w4 - s \times \frac{\partial L}{\partial w4} = 3.0048$

重新进行 L 的计算：
$z1' = w1'x1 + w2'x2 = 1.3474$
$z2' = w3'x1 + w4'x2 = 3.812$
$a1' = \Gamma(z1')' = 0.7937$
$a2' = \Gamma(z2')' = 0.9784$
$\hat{y} = w5'a1' + w6'a2' = 2.479$
$L' = (y - \hat{y})^2 = 2.31 < 5.04$

由此可见，经过一次梯度下降迭代后，损失函数的值比以前更小。