华东师范大学软件学院课程作业

课程名称：软件质量分析	年级：2023 级本科	姓名：张梓卫
作业主题：游戏服务器系统的可信度量	学号：10235101526	作业日期：2024/12/04
指导老师：陈仪香	组号：	

目录

一 基于组件的软件可信性度量模型

1 权重向量的计算

首先需要计算 EV、LLSM、CSM，我们可以拿 Homework 6 的算法直接使用：

```python
import numpy as np

A = np.array([
    [1, 2, 1/2, 2, 1/4],
    [1/2, 1, 2, 3, 1/2],
    [2, 1/2, 1, 1, 1/2],
    [1/2, 1/3, 1, 1, 1/2],
    [4, 2, 2, 2, 1]
])

def calculate_ev_weights(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)  # 返回值为元组，第一个元素为特征值，第二个元素为特征向量
    max_index = np.argmax(eigenvalues)  # 取最大特征值对应的特征向量
    weights = eigenvectors[:, max_index].real
    # 归一化
    return weights / np.sum(weights)

def calculate_llsm_weights(matrix):
    # 分子 = 每行的乘积 × 开维度数次方
    numerator = np.prod(matrix, axis=1) ** (1 / matrix.shape[0])
    # 分母 = 每行的分子相加
    denominator =  np.sum(numerator)
    # 归一化
    return numerator / denominator

def calculate_csm_weights(matrix: np.array, epsilon: float, max_iterations: int) -> np.ndarray:
    '''计算 CSM，传入：matrix, precision, maximum iterations'''
    n = matrix.shape[0]
    # 初始化初始解
    W = np.ones(n) / n

    for k in range(max_iterations):
```

```
        e = np.zeros(n)
        for i in range(n):
            e[i] = np.sum([(1 + matrix[j, i] ** 2) * (W[i] / W[j]) - (1 + matrix[i, j] ** 2) * (W[j] / W[i])
                           for j in range(n) if j != i])

        max_e = np.max(np.abs(e))
        if max_e <= epsilon: # 若精度已到达，那么停止迭代
            print(f"CSM 算法在迭代次数为 {k} 次时收敛")
            break

        m = np.argmax(np.abs(e)) # 查找最大无差的索引

        # 计算 T(k)
        up = np.sum([(1 + matrix[m, j] ** 2) * (W[j] / W[m]) for j in range(n) if j != m])
        bottom = np.sum([(1 + matrix[j, m] ** 2) * (W[m] / W[j]) for j in range(n) if j != m])
        T = np.sqrt(up / bottom)

        # 更新矩阵向量，归一化
        X = W.copy()
        X[m] *= T
        W = X / np.sum(X)

    return W

def calculate_td(matrix: np.array, weight_vector: np.ndarray) -> float:
    n = len(weight_vector)
    td = 0.0
    for i in range(n):
        for j in range(n):
            td += abs(matrix[i, j] - (weight_vector[i] / weight_vector[j]))
    return td

weights_ev = calculate_ev_weights(A)
print("判断矩阵:\n", A)
print("权重向量 EV:", weights_ev)

weights_llsm = calculate_llsm_weights(A)
print("权重向量 LLSM:", weights_llsm)

weights_csm = calculate_csm_weights(A, 1e-10, 1000)
print("CSM 权重向量:", weights_csm)

TD_EV = calculate_td(A, weights_ev)
TD_LLSM = calculate_td(A, weights_llsm)
TD_CSM = calculate_td(A, weights_csm)

print("TD EV:", TD_EV)
print("TD LLSM:", TD_LLSM)
print("TD CSM:", TD_CSM)

td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}
min_method = min(td_values, key=td_values.get)
print(f"最小的 TD 值为 {td_values[min_method]}，选 {min_method} 计算的权重向量为可信属性的权重向量。")
```

Homework 6

将关键组件的矩阵 A 输入到代码中，输出结果为：

```
D:\Python\python.exe F:\Project\Python\SoftwareQuality\SoftwareQuality\Main9.py
判断矩阵:
 [[1.         2.         0.5        2.         0.25      ]
 [0.5        1.         2.         3.         0.5       ]
 [2.         0.5        1.         1.         0.5       ]
 [0.5        0.33333333 1.         1.         0.5       ]
 [4.         2.         2.         2.         1.        ]]
权重向量 EV: [0.172769   0.20021897 0.16090765 0.10824147 0.35786291]
权重向量 LLSM: [0.16020622 0.19957385 0.16020622 0.11195645 0.36805725]
CSM 算法在迭代次数为 52 次时收敛
CSM 权重向量: [0.15088289 0.19804454 0.16081124 0.10539108 0.38487026]
TD EV: 11.41112202170799
TD LLSM: 11.376383332147004
TD CSM: 11.47657996163459
最小的 TD 值为 11.376383332147004，选 LLSM 计算的权重向量为可信属性的权重向量。

Process finished with exit code 0
```

图 1: 权重向量

整理如下：

解答 一 .1: 关键组件

正互反判断矩阵：

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

方法	权重向量
EV	(0.1728, 0.2002, 0.1609, 0.1082, 0.3579)
LLSM	(0.1602, 0.1996, 0.1602, 0.1120, 0.3681)
CSM	(0.1509, 0.1980, 0.1608, 0.1054, 0.3849)

三个权重向量的值分别为 $TD^{EV} = 11.4111, TD^{LLSM} = 11.3764, TD^{CSM} = 11.4766$。

最小的 TD 值为 11.3764，选 LLSM 计算的权重向量为可信属性的权重向量。

表 1: 关键组件正互反判断矩阵及权重

组件名	CP1	CP2	CP3	CP4	CP5	属性权重
CP1	1	2	1/2	2	1/4	0.1602
CP2	1/2	1	2	3	1/2	0.1996
CP3	2	1/2	1	1	1/2	0.1602
CP4	1/2	1/3	1	1	1/2	0.1120
CP5	4	2	2	2	1	0.3681

同理，非关键组件代入矩阵 A，得到的结果如下：

```
权重向量 EV: [0.18422453 0.1492196  0.10347843 0.18386596 0.09311632 0.14820475
 0.06894196 0.06894844]
权重向量 LLSM: [0.18373096 0.1500157  0.10384807 0.18373096 0.09727315 0.14471694
 0.07130095 0.06538326]
CSM 算法在迭代次数为 69 次时收敛
CSM 权重向量: [0.19639888 0.14670573 0.09493649 0.19089291 0.09686292 0.15019035
 0.06370886 0.06030386]
TD EV: 23.23883310674466
TD LLSM: 22.908111977844907
TD CSM: 21.965839212803633
最小的 TD 值为 21.965839212803633，选 CSM 计算的权重向量为可信属性的权重向量。
```

图 2: 权重向量

解答 一 .2: 非关键组件

正互反判断矩阵：

$$A=\begin{bmatrix} 1 & 3 & 2 & \frac{1}{2} & 2 & 1 & 3 & 3 \\ \frac{1}{3} & 1 & 2 & 1 & 2 & 2 & 2 & 2 \\ \frac{1}{2} & \frac{1}{2} & 1 & \frac{1}{2} & 1 & \frac{1}{2} & 3 & 3 \\ 2 & 1 & 2 & 1 & 3 & \frac{1}{2} & 3 & 3 \\ \frac{1}{2} & \frac{1}{2} & 1 & \frac{1}{3} & 1 & 1 & 2 & 2 \\ 1 & \frac{1}{2} & 2 & 2 & 1 & 1 & 2 & 2 \\ \frac{1}{3} & \frac{1}{2} & 1 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & 1 & 2 \\ \frac{1}{3} & \frac{1}{2} & 2 & \frac{1}{3} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 1 \end{bmatrix}$$

方法	权重向量
EV	(0.1842, 0.1492, 0.1035, 0.1839, 0.0931, 0.1482, 0.0689, 0.0689)
LLSM	(0.1837, 0.1500, 0.1038, 0.1837, 0.0973, 0.1447, 0.0713, 0.0654)
CSM	(0.1964, 0.1467, 0.0949, 0.1909, 0.0969, 0.1502, 0.0637, 0.0603)

三个权重向量的值分别为 $TD^{EV}=23.2388, TD^{LLSM}=22.9081, TD^{CSM}=21.9658$。
最小的 TD 值为 21.9658，选 CSM 计算的权重向量为可信属性的权重向量。

表 2: 非关键组件正互反判断矩阵及权重

组件名	CP6	CP7	CP8	CP9	CP10	CP11	CP12	CP13	属性权重
CP6	1	3	2	1/2	2	1	3	3	0.1964
CP7	1/3	1	2	1	2	2	2	2	0.1467
CP8	1/2	1/2	1	1/2	1	1/2	3	3	0.0949
CP9	2	1	2	1	3	1/2	3	3	0.1909
CP10	1/2	1/2	1	1/3	1	1	2	2	0.0969
CP11	1	1/2	2	2	1	1	2	2	0.1502
CP12	1/3	1/2	1	1/3	1/2	1/2	1	2	0.0637
CP13	1/3	1/2	2	1/3	1/2	1/2	1/2	1	0.0603

2 计算 T_s 的值

先写出整体最重要的函数部分，如下所示：

```
    def calculate_system_trustworthiness(
        Critical_Confidence_value, Non_Critical_Confidence_value,
        alpha, beta, FC, NFC
):
    """
```

```
    计算系统可信度值 T_S

    参数:
    - Critical_Confidence_value: 关键组件的可信值数组 (numpy array)
    - Non_Critical_Confidence_value: 非关键组件的可信值数组 (numpy array)
    - alpha: 关键组件权重系数
    - beta: 非关键组件权重系数
    - FC: 关键组件权重向量 (numpy array)
    - NFC: 非关键组件权重向量 (numpy array)

    返回:
    - 系统可信度值 T_S
    """
    critical_product = np.prod(
        np.power(Critical_Confidence_value, FC)
    )
    print(f"关键属性乘积: {critical_product}")

    non_critical_product = np.prod(
        np.power(Non_Critical_Confidence_value, NFC)
    )
    print(f"非关键属性乘积: {non_critical_product}")

    T_S = alpha * critical_product + beta * non_critical_product

    return T_S
```

Ts

接下来，我们可以计算关键组件的可信值数组和非关键组件的可信值数组，并输入到函数中：

```
alpha = 0.7
beta = 0.3
print(f"此时的 Alpha 值为 {alpha}, Beta 值为 {beta}")

T_S = calculate_system_trustworthiness(
    Critical_Confidence_value, Non_Critical_Confidence_value,
    alpha, beta, final_critical_weights, final_non_critical_weights
)

print(f"Ts的值为: {T_S}")
```

Ts

调整 α 和 β 的值，可以得到不同的 T_s 值。

示例输出如下：

```
最小的 TD 值为 21.965839212803633，选 CSM 计算的权重向量为可信属性的权重向量。
此时的 Alpha 值为 0.7, Beta 值为 0.3
关键属性乘积: 8.02909524877413
非关键属性乘积: 7.804852693136423
Ts的值为：7.961822482082818
```

```
最小的 TD 值为 21.965839212803633，选 CSM 计算的权重向量为可信属性的权重向量。
此时的 Alpha 值为 0.6, Beta 值为 0.4
关键属性乘积: 8.02909524877413
非关键属性乘积: 7.804852693136423
Ts的值为：7.939398226519048
```

```
最小的 TD 值为 21.965839212803633，选 CSM 计算的权重向量为可信属性的权重向量。
此时的 Alpha 值为 0.55, Beta 值为 0.45
关键属性乘积: 8.02909524877413
非关键属性乘积: 7.804852693136423
Ts的值为：7.928186098737163
```

图 3: 不同 Ts 值的对比

最后，我们可以制作为 Ts 的表格，如下所示：

Alpha	**Beta**	T_s **值**
0.70	0.30	7.9618
0.60	0.40	7.9394
0.55	0.45	7.9282

3 可信等级

根据表格中的评判标准，我们可以轻易看出

$7.0 <= T < 8.5$ 或者$T > 8.5$且不能评为Ⅳ级别及以上者	1. 低于7.0分的关键组件个数不超过$n - \lceil n \times 2/3 \rceil$ 2. 没有低于4.5分的可信属性	Ⅲ

图 4: 可信等级

显然所有的关键属性都没有低于 7.0 分的，而 $T_s \geq 7$ 在不同的 α 和 β 之下都成立。
故可信软件等级为 **III**，当然，我们还可以编写代码来实现：

```python
def classify_trust_level(T, scores):
"""
分类可信等级
:param: T: 系统可信度值
:param: scores: 可信属性值数组
:return: 可信等级
"""
n = len(scores)
threshold_2_3 = n - int(np.ceil(2 * n / 3))  # 计算 n -  n × 2/3

# 统计低于某些阈值的属性个数
low_9_5 = np.sum(scores < 9.5)
low_8_5 = np.sum(scores < 8.5)
low_7_0 = np.sum(scores < 7.0)
low_4_5 = np.sum(scores < 4.5)

if T >= 9.5:
```

```
        if low_9_5 <= threshold_2_3 and low_8_5 == 0:
            return "V"

    if 8.5 <= T < 9.5 or (T > 9.5):
        if low_8_5 <= threshold_2_3 and low_7_0 == 0:
            return "IV"

    if 7.0 <= T < 8.5 or (T > 8.5):
        if low_7_0 <= threshold_2_3 and low_4_5 == 0:
            return "III"

    if 4.5 <= T < 7.0 or (T > 7.0):
        if low_4_5 <= threshold_2_3:
            return "II"

    if T < 4.5 or (T > 4.5):
        return "I"
```

可信等级代码

二 附录

1 完整可执行代码

```
from pprint import pprint

import numpy as np

# 作业
Critical_Confidence_value = np.array([8.430, 8.530, 6.042, 9.094, 8.289])

Non_Critical_Confidence_value = np.array([6.192, 8.020, 7.984, 8.713, 9.211, 7.777, 7.897, 8.075])

Critical_A = np.array([
    [1, 2, 1/2, 2, 1/4],
    [1/2, 1, 2, 3, 1/2],
    [2, 1/2, 1, 1, 1/2],
    [1/2, 1/3, 1, 1, 1/2],
    [4, 2, 2, 2, 1]
])

Non_Critical_A = np.array([
    [1, 3, 2, 1/2, 2, 1, 3, 3],
    [1/3, 1, 2, 1, 2, 2, 2, 2],
    [1/2, 1/2, 1, 1/2, 1, 1/2, 3, 3],
    [2, 1, 2, 1, 3, 1/2, 3, 3],
    [1/2, 1/2, 1, 1/3, 1, 1, 2, 2],
    [1, 1/2, 2, 2, 1, 1, 2, 2],
    [1/3, 1/2, 1, 1/3, 1/2, 1/2, 1, 2],
    [1/3, 1/2, 2, 1/3, 1/2, 1/2, 1/2, 1]
])

# Test Data
#
# Critical_Confidence_value = np.array([9.536, 7.531, 9.167, 8.423])
#
# Non_Critical_Confidence_value = np.array([7.556, 7.858, 8.979, 7.708, 9.265])
#
# Critical_A = np.array([
#     [1, 2, 1/2, 1/3],
#     [1/2, 1, 1/2, 1/3],
#     [2, 2, 1, 1/2],
```

```
#     [3, 3, 2, 1]
# ])
#
# Non_Critical_A = np.array([
#     [1, 1/2, 1, 1/3, 1/2],
#     [2, 1  , 2, 1/2, 1],
#     [1, 1/2, 1, 1/2, 1],
#     [3, 2  , 2, 1  , 2],
#     [2, 1  , 1, 1/2, 1]
# ])


def classify_trust_level(T, scores):
    """
    分类可信等级
    :param: T: 系统可信度值
    :param: scores: 可信属性值数组
    :return: 可信等级
    """
    n = len(scores)
    threshold_2_3 = n - int(np.ceil(2 * n / 3))  # 计算 n - n × 2/3

    # 统计低于某些阈值的属性个数
    low_9_5 = np.sum(scores < 9.5)
    low_8_5 = np.sum(scores < 8.5)
    low_7_0 = np.sum(scores < 7.0)
    low_4_5 = np.sum(scores < 4.5)

    if T >= 9.5:
        if low_9_5 <= threshold_2_3 and low_8_5 == 0:
            return "V"

    if 8.5 <= T < 9.5 or (T > 9.5):
        if low_8_5 <= threshold_2_3 and low_7_0 == 0:
            return "IV"

    if 7.0 <= T < 8.5 or (T > 8.5):
        if low_7_0 <= threshold_2_3 and low_4_5 == 0:
            return "III"

    if 4.5 <= T < 7.0 or (T > 7.0):
        if low_4_5 <= threshold_2_3:
            return "II"

    if T < 4.5 or (T > 4.5):
        return "I"


def calculate_system_trustworthiness(
        Critical_Confidence_value, Non_Critical_Confidence_value,
        alpha, beta, FC, NFC
):
    """
    计算系统可信度值 T_S

    参数:
    - Critical_Confidence_value: 关键组件的可信值数组 (numpy array)
    - Non_Critical_Confidence_value: 非关键组件的可信值数组 (numpy array)
    - alpha: 关键组件权重系数
    - beta: 非关键组件权重系数
    - FC: 关键组件权重向量 (numpy array)
    - NFC: 非关键组件权重向量 (numpy array)

    返回:
```

```
        - 系统可信度值 T_S
        """
        critical_product = np.prod(
            np.power(Critical_Confidence_value, FC)
        )
        print(f"关键属性乘积: {critical_product}")

        non_critical_product = np.prod(
            np.power(Non_Critical_Confidence_value, NFC)
        )
        print(f"非关键属性乘积: {non_critical_product}")

        T_S = alpha * critical_product + beta * non_critical_product

        return T_S

    def calculate_ev_weights(matrix):
        eigenvalues, eigenvectors = np.linalg.eig(matrix)  # 返回值为元组，第一个元素为特征值，第二个元素为特征向量
        max_index = np.argmax(eigenvalues)  # 取最大特征值对应的特征向量
        weights = eigenvectors[:, max_index].real
        # 归一化
        return weights / np.sum(weights)

    def calculate_llsm_weights(matrix):
        # 分子 = 每行的乘积 × 开维度数次方
        numerator = np.prod(matrix, axis=1) ** (1 / matrix.shape[0])
        # 分母 = 每行的分子相加
        denominator =  np.sum(numerator)
        # 归一化
        return numerator / denominator

    def calculate_csm_weights(matrix: np.array, epsilon: float, max_iterations: int) -> np.ndarray:
        '''计算 CSM，传入：matrix, precision, maximum iterations'''
        n = matrix.shape[0]
        # 初始化初始解
        W = np.ones(n) / n

        for k in range(max_iterations):
            e = np.zeros(n)
            for i in range(n):
                e[i] = np.sum([(1 + matrix[j, i] ** 2) * (W[i] / W[j]) - (1 + matrix[i, j] ** 2) * (W[j] / W[i])
                               for j in range(n) if j != i])

            max_e = np.max(np.abs(e))
            if max_e <= epsilon: # 若精度已到达，那么停止迭代
                print(f"CSM 算法在迭代次数为 {k} 次时收敛")
                break

            m = np.argmax(np.abs(e)) # 查找最大无差的索引

            # 计算 T(k)
            up = np.sum([(1 + matrix[m, j] ** 2) * (W[j] / W[m]) for j in range(n) if j != m])
            bottom = np.sum([(1 + matrix[j, m] ** 2) * (W[m] / W[j]) for j in range(n) if j != m])
            T = np.sqrt(up / bottom)

            # 更新矩阵向量，归一化
            X = W.copy()
            X[m] *= T
            W = X / np.sum(X)

        return W

    def calculate_td(matrix: np.array, weight_vector: np.ndarray) -> float:
```

```
        n = len(weight_vector)
        td = 0.0
        for i in range(n):
            for j in range(n):
                td += abs(matrix[i, j] - (weight_vector[i] / weight_vector[j]))
        return td


print("————————————————————————————")
print("以下是关键组件的判断矩阵：")


print("判断矩阵:")
pprint(Critical_A)
print("————————————————————————————")

weights_ev = calculate_ev_weights(Critical_A)
print("权重向量 EV:", weights_ev)

weights_llsm = calculate_llsm_weights(Critical_A)
print("权重向量 LLSM:", weights_llsm)

weights_csm = calculate_csm_weights(Critical_A, 1e-10, 1000)
print("CSM 权重向量:", weights_csm)

TD_EV = calculate_td(Critical_A, weights_ev)
TD_LLSM = calculate_td(Critical_A, weights_llsm)
TD_CSM = calculate_td(Critical_A, weights_csm)

print("TD EV:", TD_EV)
print("TD LLSM:", TD_LLSM)
print("TD CSM:", TD_CSM)

td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}
min_method = min(td_values, key=td_values.get)
print(f"最小的 TD 值为 {td_values[min_method]}，选 {min_method} 计算的权重向量为可信属性的权重向量。")
final_critical_weights = weights_ev if min_method == "EV" else weights_llsm if min_method == "LLSM" else
weights_csm

print("————————————————————————————")
print("以下是非关键组件的判断矩阵：")

print("判断矩阵:")
pprint(Non_Critical_A)
print("————————————————————————————")

weights_ev = calculate_ev_weights(Non_Critical_A)
print("权重向量 EV:", weights_ev)

weights_llsm = calculate_llsm_weights(Non_Critical_A)
print("权重向量 LLSM:", weights_llsm)

weights_csm = calculate_csm_weights(Non_Critical_A, 1e-10, 1000)
print("CSM 权重向量:", weights_csm)

TD_EV = calculate_td(Non_Critical_A, weights_ev)
TD_LLSM = calculate_td(Non_Critical_A, weights_llsm)
TD_CSM = calculate_td(Non_Critical_A, weights_csm)

print("TD EV:", TD_EV)
print("TD LLSM:", TD_LLSM)
print("TD CSM:", TD_CSM)

td_values = {"EV": TD_EV, "LLSM": TD_LLSM, "CSM": TD_CSM}
```

```
    min_method = min(td_values, key=td_values.get)
    # min_method = "LLSM"
    print(f"最小的 TD 值为 {td_values[min_method]}，选 {min_method} 计算的权重向量为可信属性的权重向量。")
    final_non_critical_weights = weights_ev if min_method == "EV" else weights_llsm if min_method == "LLSM" else
     weights_csm

    alpha = 0.6
    beta = 0.4
    print(f"此时的 Alpha 值为 {alpha}, Beta 值为 {beta}")

    T_S = calculate_system_trustworthiness(
        Critical_Confidence_value, Non_Critical_Confidence_value,
        alpha, beta, final_critical_weights, final_non_critical_weights
    )

    print(f"Ts的值为：{T_S}")
    Level = classify_trust_level(T_S, Critical_Confidence_value)
    print("该软件的可信等级为：" + Level)
```

完整代码