Задача нечеткой кластеризации: Метод декомпозиционного дерева

Preliminaries

In [1]:

```
import numpy as np
from typing import Callable
```

Определим Расстояние Хемминга:

In [2]:

```
def Hamming(i:np.int, j:np.int, table: np.ndarray) -> np.ndarray:
    """
    Computes Hamming distance between A_i and A_j
    Where A_i \in [0,1]^n -- n-dim. vector that contains estimation that
    i-th element has P_k feature
    """
    A_i = table[:,i]
    A_j = table[:,j]
    return sum(abs(A_i - A_j))
```

Определим (max-min)-композицию:

In [3]:

```
def mu(ix: np.int, iy: np.int, arr1: np.ndarray, arr2=np.nan) -> np.float:
    """
    Returns (max-min)-composition for single pair
    """
    if arr2 is np.nan:
        arr2 = arr
    xz = arr1[ix,:]
    zy = arr2[:,iy]
    choose = np.array(list(map(lambda x: min(x), list(zip(xz,zy)))))
    val = max(choose)
    return val

def maxmin(arr1:np.ndarray, arr2=np.nan) -> np.ndarray:
    """
    Returns (max-min)-composition for given matrices
    """
    if arr2 is np.nan:
        arr2 = arr1
    x = arr1.shape[0]
    y = arr2.shape[1]
    pot = np.empty((x,y))
    for ix in range(0,x):
        for iy in range(0, y):
            pot[ix,iy] = mu(ix,iy,arr1,arr2)
    return pot
```

Определим (max-T)-композицию:

In [4]:

```
def T(x:np.float, y:np.float, a=-5, b=0.2) -> np.float:
    """
    Computes given T-norm
    """
    numer = x*y - (1-a)*(1-b)*(1-x)*(1-y)
    denom = 1 + a*b*(1-x)*(1-y)
    return max(0, numer/denom)

def muT(ix: np.int, iy: np.int, arr1: np.ndarray, arr2=np.nan) -> np.float:
    """
    Returns (max-T)-composition for single pair
    """
    if arr2 is np.nan:
        arr2 = arr
    xz = arr1[ix,:]
    zy = arr2[:,iy]
    choose = np.array(list(map(lambda x: T(*x), list(zip(xz,zy)))))
    val = max(choose)
    return val

def maxT(arr1:np.ndarray, arr2=np.nan, T=T) -> np.ndarray:
    """
    Computes (max-T)-composition for given matrices and given T-norm
    """
    if arr2 is np.nan:
        arr2 = arr1
    x = arr1.shape[0]
    y = arr2.shape[1]
    pot = np.empty((x,y))
    for ix in range(0,x):
        for iy in range(0, y):
            pot[ix,iy] = np.round(muT(ix,iy,arr1,arr2),2)
    return pot
```

Определим транзитивное замыкание:

In [5]:

```
def union(arr1:np.ndarray, arr2:np.ndarray) -> np.ndarray:
    """
    Computes union of 2 fuzzy relations
    """
    shape = arr1.shape
    pot = np.empty(shape)
    for ix in range(0,shape[0]):
        for iy in range(0,shape[1]):
            pot[ix,iy] = max(arr1[ix,iy],arr2[ix,iy])
    return pot

def trans_closure(arr:np.ndarray, comp = maxmin, show = False) -> np.ndarray:
    """
    Computes transitive closure of a fuzzy relation
    """
    pot = arr
    new_arr = comp(arr)
    count = 1
    if show:
        print(count)
        print(new_arr)
        print('----------')
    if (new_arr==arr).all():
        return pot
    else:
        pot = union(new_arr,arr)
        while True:
            prev_arr = new_arr
            new_arr = comp(new_arr,arr)
            count += 1
            if show:
                print(count)
                print(new_arr)
                print('----------')
            if (prev_arr==new_arr).all():
                return pot
            else:
                pot = union(new_arr,arr)
```

Определим процедуру декомпозиции отношения эквивалентности:

In [6]:

```
def alphas(arr:np.ndarray) -> np.ndarray:
    """
    Returns valid alpha for future alpha-decomposition for decomposing tree
    """
    mask = (arr > 0) & (arr < 1)
    alphs = np.unique(np.append(arr[mask], [1])) # add 1 separately in case if r>1 ∀r ∈ R
    return alphs

def decompose(arr:np.ndarray) -> np.ndarray:
    """
    Returns pairs (alpha, R_aplha) that are given during decomposition for given relation
    Such that for given R: R = max_alpha{alpha * R_aplha}, where alpha \in (0,1]
    """
    alphs = alphas(arr)
    cuts = []
    for a in alphs:
        mask = (arr >= a)
        cuts.append(mask.astype(np.int))
    return list(zip(alphs,cuts))
```

Определим процедуру нормализации матрицы:

In [7]:

```
def normalize(R:np.ndarray) -> np.ndarray:
    row_sums = R.sum(axis=1)
    return R / row_sums[:, np.newaxis]
```

Main Part

Имеется следующая таблица:

In [8]:

```
U = ('a', 'b', 'c', 'd', 'e')
given = "0.8 0.7 0.7 0.3 0.0 | 0.5 1 0.5 0.0 0.0 | 0.5 0.6 0.8 0.4 0.2 | \
         0.9 0.5 0.3 0.2 0.2 | 0.6 0.8 0.9 0.3 0.1 | 0.2 0.4 0.6 0.8 0.9"
lines = given.split("|")
elements = [line.split() for line in lines]
table = np.array([list(map(lambda x: float(x), line)) for line in elements])
table
```

Out[8]:

```
array([[0.8, 0.7, 0.7, 0.3, 0. ],
       [0.5, 1. , 0.5, 0. , 0. ],
       [0.5, 0.6, 0.8, 0.4, 0.2],
       [0.9, 0.5, 0.3, 0.2, 0.2],
       [0.6, 0.8, 0.9, 0.3, 0.1],
       [0.2, 0.4, 0.6, 0.8, 0.9]])
```

С помощью расстояния Хемминга построим матрицу отношения несходства R, а после перейдем к матрице сходства:

In [9]:

```
card = table.shape[1]
R = np.empty((card,card))
for i in range(0,card):
    for j in range(0,card):
        R[i,j] = Hamming(i,j,table)
R = normalize(R)
Rhat = 1 - R
print("R matrix:\n{}\n\nRhat matrix:\n{}".format(R,Rhat))
```

```
R matrix:
[[0.         0.15957447 0.18085106 0.28723404 0.37234043]
 [0.16483516 0.         0.13186813 0.30769231 0.3956044 ]
 [0.20987654 0.14814815 0.         0.27160494 0.37037037]
 [0.31764706 0.32941176 0.25882353 0.         0.09411765]
 [0.32110092 0.33027523 0.27522936 0.0733945  0.        ]]

Rhat matrix:
[[1.         0.84042553 0.81914894 0.71276596 0.62765957]
 [0.83516484 1.         0.86813187 0.69230769 0.6043956 ]
 [0.79012346 0.85185185 1.         0.72839506 0.62962963]
 [0.68235294 0.67058824 0.74117647 1.         0.90588235]
 [0.67889908 0.66972477 0.72477064 0.9266055  1.        ]]
```

Найдем транзитивное замыкание $\hat{R}$ с двумя разными композициями:

In [10]:

```
ClR = trans_closure(Rhat)
TClR = trans_closure(Rhat, comp=maxT)
print('Transitive closure with (max-min)-composition:\n{}\n \
Transitive closure with (max-T)-composition:\n{}'.format(ClR, TClR))
```

```
Transitive closure with (max-min)-composition:
[[1.         0.84042553 0.84042553 0.72839506 0.72839506]
 [0.83516484 1.         0.86813187 0.72839506 0.72839506]
 [0.83516484 0.85185185 1.         0.72839506 0.72839506]
 [0.74117647 0.74117647 0.74117647 1.         0.90588235]
 [0.74117647 0.74117647 0.74117647 0.9266055  1.        ]]
 Transitive closure with (max-T)-composition:
[[1.         0.84042553 0.82       0.71276596 0.63      ]
 [0.84       1.         0.87       0.69230769 0.6043956 ]
 [0.79012346 0.85185185 1.         0.73       0.63      ]
 [0.68235294 0.67058824 0.74117647 1.         0.91      ]
 [0.68       0.67       0.72477064 0.93       1.        ]]
```

Разложим полученное с помощью (max-min)-композиции отношение по теореме о декомпозиции через систему α-срезов:

In [11]:

```
decClR = decompose(ClR)
decClR
```

Out[11]:

```
[(0.7283950617283951, array([[1, 1, 1, 1, 1],
         [1, 1, 1, 1, 1],
         [1, 1, 1, 1, 1],
         [1, 1, 1, 1, 1],
         [1, 1, 1, 1, 1]])), (0.7411764705882353, array([[1, 1, 1, 0, 0],
         [1, 1, 1, 0, 0],
         [1, 1, 1, 0, 0],
         [1, 1, 1, 1, 1],
         [1, 1, 1, 1, 1]])), (0.8351648351648351, array([[1, 1, 1, 0, 0],
         [1, 1, 1, 0, 0],
         [1, 1, 1, 0, 0],
         [0, 0, 0, 1, 1],
         [0, 0, 0, 1, 1]])), (0.8404255319148937, array([[1, 1, 1, 0, 0],
         [0, 1, 1, 0, 0],
         [0, 1, 1, 0, 0],
         [0, 0, 0, 1, 1],
         [0, 0, 0, 1, 1]])), (0.8518518518518519, array([[1, 0, 0, 0, 0],
         [0, 1, 1, 0, 0],
         [0, 1, 1, 0, 0],
         [0, 0, 0, 1, 1],
         [0, 0, 0, 1, 1]])), (0.8681318681318682, array([[1, 0, 0, 0, 0],
         [0, 1, 1, 0, 0],
         [0, 0, 1, 0, 0],
         [0, 0, 0, 1, 1],
         [0, 0, 0, 1, 1]])), (0.9058823529411765, array([[1, 0, 0, 0, 0],
         [0, 1, 0, 0, 0],
         [0, 0, 1, 0, 0],
         [0, 0, 0, 1, 1],
         [0, 0, 0, 1, 1]])), (0.926605504587156, array([[1, 0, 0, 0, 0],
         [0, 1, 0, 0, 0],
         [0, 0, 1, 0, 0],
         [0, 0, 0, 1, 0],
         [0, 0, 0, 1, 1]])), (1.0, array([[1, 0, 0, 0, 0],
         [0, 1, 0, 0, 0],
         [0, 0, 1, 0, 0],
         [0, 0, 0, 1, 0],
         [0, 0, 0, 0, 1]]))]
```

In [ ]:

```
```