from collections import defaultdict

from typing import Dict, List

import numpy as np

# ======== Шаг 0: Симулируем входные данные =========

# Это то, что у тебя должно быть из сырых текстов, где

# ключи - слова (word=w), значения - словарь контекстов (context=c) с частотами слов в этих контекстах

# Пусть у нас есть тексты, где слова w1, w2 встречаются в контекстах c1, c2 с какими-то частотами

raw\_data = {

'w1': {'c1': 5, 'c2': 2, 'c3': 1}, # w1 встретилось 5 раз в c1, 2 раза в c2, 1 раз в c3

'w2': {'c1': 3, 'c2': 4, 'c4': 6}, # w2 встретилось 3 раза в c1, 4 раза в c2, 6 раз в c4

}

# ======== Шаг 1: proximity и frequency ========

def compute\_proximity\_frequency(data: Dict[str, Dict[str, int]], word: str, context: str) -> float:

"""

proximity(w1, ci) = frequency(w1, ci) / max\_frequency(w1)

"""

word\_contexts = data.get(word, {})

frequency\_w\_c = word\_contexts.get(context, 0) # frequency\_c\_i, w

# max frequency слова среди всех контекстов

max\_frequency\_w = max(word\_contexts.values(), default=1) # w1\_max\_frequency (на случай нуля ставим 1 чтобы не делить на 0)

if max\_frequency\_w == 0: # вообще не встретилось нигде

return 0

proximity\_w\_c = frequency\_w\_c / max\_frequency\_w

return proximity\_w\_c

# ======== Шаг 2: similarity между словом и контекстом ========

def compute\_similarity(data: Dict[str, Dict[str, int]], word: str, context: str) -> float:

"""

similarity(word, context) = proximity(word, context)

тут для простоты я считаю proximity сразу как similarity (м.б. у тебя тут другая формула)

"""

return compute\_proximity\_frequency(data, word, context)

# ======== Шаг 3: R\_i - ранг similarity слова с контекстами ========

def compute\_rank\_similarities(data: Dict[str, Dict[str, int]], word: str) -> Dict[str, int]:

"""

Получить ранги контекстов по similarity с данным словом.

Возвращает {context: rank} (чем больше similarity, тем меньше ранг)

"""

similarities = {}

for context in set().union(\*[contexts.keys() for contexts in data.values()]): # все уникальные контексты

sim = compute\_similarity(data, word, context)

similarities[context] = sim

# сортируем по убыванию similarity и присваиваем ранги (чем выше sim, тем меньше ранг)

sorted\_sims = sorted(similarities.items(), key=lambda x: x[1], reverse=True)

ranks = {context: rank+1 for rank, (context, \_) in enumerate(sorted\_sims)} # +1 потому что в питоне с 0 индексы

return ranks

# ======== Шаг 4: Финальная семантическая близость между w1 и w2 ========

def semantic\_proximity(data: Dict[str, Dict[str, int]], w1: str, w2: str, k: float = 1.0, s: float = 1.0) -> float:

"""

semantic proximity(w1, w2) = 1/k \* sum[ (Ri1 + Ri2 + s1 + s2) ^ -1 ]

где Ri1, Ri2 - ранги контекста i для w1 и w2 соответственно

s1, s2 - какие-то сглаживания (у тебя в формуле просто s, я сделал s1+s2 на всякий)

"""

ranks\_w1 = compute\_rank\_similarities(data, w1)

ranks\_w2 = compute\_rank\_similarities(data, w2)

prox = 0.0

all\_contexts = set(ranks\_w1.keys()).union(ranks\_w2.keys())

for context in all\_contexts:

ri1 = ranks\_w1.get(context, len(all\_contexts) + 1) # если контекст только у w2 есть, то худший ранг для w1

ri2 = ranks\_w2.get(context, len(all\_contexts) + 1)

sim\_w1\_c = compute\_similarity(data, w1, context)

sim\_w2\_c = compute\_similarity(data, w2, context)

prox += 1 / (ri1 + ri2 + sim\_w1\_c + sim\_w2\_c) # тут s1+s2 сделал как сумму similarity (м.б. ты хотел другое)

return (1 / k) \* prox

# ======== Шаг 5: Пример использования ========

if \_\_name\_\_ == '\_\_main\_\_':

w1 = 'w1'

w2 = 'w2'

print(f"Сырые данные:\n {raw\_data}")

prox\_w1\_c1 = compute\_proximity\_frequency(raw\_data, w1, 'c1')

print(f"\nProximity(w1, c1) = {prox\_w1\_c1:.2f}")

sim\_w1\_c1 = compute\_similarity(raw\_data, w1, 'c1')

print(f"Similarity(w1, c1) = {sim\_w1\_c1:.2f} (тут просто == proximity)")

ranks\_w1 = compute\_rank\_similarities(raw\_data, w1)

print(f"\nРанги контекстов для w1: {ranks\_w1}")

final\_prox = semantic\_proximity(raw\_data, w1, w2, k=1.0, s=1.0)

print(f"\nФинальная семантическая близость между '{w1}' и '{w2}': {final\_prox:.4f}")