

Adam Bogusz

Nr indeksu: 97569

Grupa I2N 1.1/1

Personalized Clustering for Emotion Recognition Improvement

Opis metody personalizowanej klasteryzacji

Artykuł wprowadza koncepcję **personalizowanego klastrowania**, mającą na celu wypełnienie luki między niedokładnymi modelami ogólnymi a kosztownymi modelami indywidualnymi.

Metodologia opiera się na **grupowaniu użytkowników w klastry o zbliżonych profilach reakcji emocjonalnych** i trenowaniu dla nich dedykowanych modeli. Główną zaletą tego rozwiązania jest **możliwość dynamicznego przypisywania nowych użytkowników do istniejących grup bez konieczności etykietowania ich danych**, co redukuje zmienność wyników i zwiększa niezawodność systemu.

Etapy metodologii

1. Przygotowanie danych
2. Metodologia M1: Tworzenie typologii użytkowników
 - a. Trenowanie modeli pół-personalizowanych
3. Metodologia M2: Konfiguracja włączania użytkowników bez danych etykietowanych
 - a. Faktyczne włączanie użytkowników
4. Wnioskowanie i fuzja

Przygotowanie danych

Zanim nastąpi klastrowanie, surowe dane muszą zostać przetworzone, aby nadawały się do analizy przez algorytmy:

- **Ekstrakcja cech:** Z surowych sygnałów fizjologicznych pobieranych w 20-sekundowych oknach obliczane jest 57 cech statystycznych
- **Normalizacja:** Wszystkie cechy są poddawane standaryzacji Z-score (badanie odchylenia wartości od średniej) dla każdego ochotnika, aby wyeliminować różnice wynikające z osobniczych cech fizjologicznych.

Metodologia M1

Ten etap odbywa się na etapie treningu systemu, przy użyciu danych etykietowanych, w celu zdefiniowania bazowych profili (klastrow).

- **Transformacja macierzy:** Mapa cech jest przekształcana w macierz profilowania użytkownika, gdzie dla każdego użytkownika obliczana jest średnia i odchylenie standardowe cech dla każdej klasy emocji.
- **Wyszukiwanie klastrow typologii (TC):** Przekształcona macierz jest poddawana klastrowaniu hierarchicznemu w celu znalezienia optymalnej liczby grup użytkowników (typologii) o podobnych reakcjach.
- **Optymalizacja:** Poprzez wskaźnik Dunna oceniana jest jakość podziału i wymuszone zostaje, aby każdy klaster zawierał minimum 15% zbioru danych.

Metodologia M1, ciąg dalszy

Po zdefiniowaniu klastrów tworzone są dedykowane modele sztucznej inteligencji.

- **Przypisanie danych:** Wszyscy użytkownicy są przypisani do konkretnych klastrów typologii (TC).
- **Trenowanie:** Dla każdego klastra typologii trenowany jest oddzielny model klasyfikacyjny (KNN), który jest pół-personalizowany

Metodologia M2

Aby system mógł przyjmować nowych użytkowników, którzy nie dostarczają pełnych profili statystycznych (tj. nie mają etykietowanych danych), konieczne jest **stworzenie klastrów wewnętrznych (IC)**.

Wewnątrz każdego klastra typologii (TC) algorytm ponownie klastruje dane treningowe, tworząc klastry wewnętrzne.

Metodologia M2, ciąg dalszy

Dodawanie nowych użytkowników do aktywnego systemu jest dokonywane poprzez **obliczenie odległości ich obserwacji do najbliższych klastrów wewnętrznych (IC)** w ramach każdego klastra typologii (TC).

Użytkownicy zostają przypisani do tego klastra typologii (i odpowiadającego mu modelu AI), dla którego suma odległości była najmniejsza.

Wnioskowanie i fuzja

Po skutecznym przypisaniu użytkownika do klastra, system uruchamia dedykowany pół-personalizowany model AI (wytrenowany na danej typologii) w celu dokonania finalnej klasyfikacji emocji.

Aby zapewnić większą niezawodność, **wynik ten może być połączony z predykcją modelu ogólnego** przy użyciu techniki **późnej fuzji**.

Mechanizm ten pozwala na dynamiczne ważenie decyzji obu modeli, co koryguje ewentualne błędy w przypadku, gdy przypisanie do typologii jest niejednoznaczne lub klaster jest słabiej zdefiniowany.

Implementacja - M1, przetwarzanie danych

```
1 #data_processing.py
2 import pandas as pd
3
4 #Transformacja macierzy D (O x F) do D' (N x M)
5 #zgodnie z metodologia M1
6 #
7 #features_D = macierz D o wymiarach O x F (Obserwacje x Cechy)
8 #user_ids = identyfikatory uzytkownikow
9 #labels = etykiety emocji
10 def transform_matrix_M1(features_D, user_ids, labels):
11
12     print(f" [M1] Rozpoczecie transformacji D (O={len(features_D)}) do D'...")
13
14     #tworzenie nazw kolumn i DataFrame
15     feature_names = [f'feature_{i}' for i in range(features_D.shape[1])]
16     df = pd.DataFrame(features_D, columns=feature_names)
17     df['user_id'] = user_ids
18     df['label'] = labels
19
20     #liczenie sredniej i odchylenia standardowego wszystkich cech dla kazdej z klas docelowych
21     #grupowanie i agregacja danych
22     grouped = df.groupby(['user_id', 'label']).agg(['mean', 'std'])
23
24     #spłaszczenie do jednego wiersza na uzytkownika (N x M)
25     matrix_D_prime = grouped.unstack(level='label', fill_value=0)
26
27     #spłaszczenie nazw kolumn
28     matrix_D_prime.columns = ['_'.join(map(str, col)).strip('_') for col in matrix_D_prime.columns]
29
30     #zamiana wartosci NaN na 0 w przypadku braku danych
31     matrix_D_prime = matrix_D_prime.fillna(0)
32
33     print(f" [M1] Transformacja zakonczona. Kszalt D': {matrix_D_prime.shape}")
34     return matrix_D_prime #wyjciowa macierz D' o wymiarach N x M (Uzytkownicy x Metryki)
```

1. Wzięcie macierzy surowych obserwacji D ($O \times F$)
2. Obliczenie średniej i odchylenia standardowego dla każdej cechy, pogrupowane według ID użytkownika i klasy emocji
3. Przekształcenie danych tak, aby jeden wiersz reprezentował jednego użytkownika
4. Utworzenie finalnej macierzy D' ($N \times M$)

Implementacja - M2, wskaźnik Dunna

$$DI_m = \frac{\min_{1 \leq i < j \leq m} \delta(C_i, C_j)}{\max_{1 \leq k \leq m} \Delta_k}$$

```
1 #clustering_tc.py
2 import numpy as np
3 from sklearn.cluster import AgglomerativeClustering
4 from scipy.spatial.distance import pdist, squareform
5
6 #Obliczenie wskaźnika Dunna służącego do oceny klastrowania
7 #używany do wybrania najlepszego podziału na grupy spośród testowanych opcji
8 #
9 #Zpoints = zbiór punktów danych|analizowana macierz profili użytkowników (X_D_prime)
10 def dunn_index(points, labels):
11     unique = np.unique(labels)
12     if len(unique) < 2:
13         return -1
14
15     #macierz odległości między parami punktów
16     D = squareform(pdist(points))
17
18     #obliczanie mianownika (szukanie klastra z największą odległością między najdalszymi punktami)
19     max_intra = 0 #maksymalna średnica
20     for lab in unique:
21         pts = np.where(labels == lab)[0]
22         if len(pts) > 1:
23             intrad = D[np.ix_(pts, pts)] #wycinanie podmacierzy z odległościami między punktami
24             #wybranie największej odległości z górnego trójkąta macierzy i porównanie z max_intra
25             max_intra = max(max_intra, intrad[np.triu_indices(len(pts), k=1)].max())
26
27     #obliczanie licznika (szukanie minimalnej odległości międzyklastrowej)
28     min_inter = np.inf #minimalna
29     for i, lab_i in enumerate(unique):
30         pts_i = np.where(labels == lab_i)[0] #indeksy rozpatrywanego klastra
31         for j, lab_j in enumerate(unique): #wybranie klastra do porównania
32             if i >= j: continue
33             pts_j = np.where(labels == lab_j)[0]
34             if len(pts_i) == 0 or len(pts_j) == 0: continue
35             #wycięcie podmacierzy z odległościami między wszystkimi punktami obu klastrów
36             dist_ij = D[np.ix_(pts_i, pts_j)]
37             #wybranie najmniejszej odległości
38             min_inter = min(min_inter, dist_ij.min())
39
40     return min_inter / max_intra if max_intra > 0 else np.inf
```

1. Obliczenie macierzy odległości euklidesowych między każdą parą punktów w zbiorze danych D
2. Obliczenie mianownika (zwartość), tj. znalezienie maksymalnej średnicy klastra (najw. odl. między dwoma punktami wewnątrz tej samej grupy)
3. Obliczenie licznika (separacja), tj. znalezienie minimalnej odległości międzyklastrowej (czyli między dwoma najbliższymi punktami z różnych grup)
4. Zwrócenie ilorazu minimalnej separacji i maksymalnej średnicy (im wyższy wynik, tym lepsze klastrowanie)

Implementacja - M2, szukanie klastrów

```
44 #Szukanie najlepszego sposobu podziału użytkowników na typologie
45 #czyli wyszukiwanie optymalnych klastrów
46 #
47 #X_D_prime = przekształcona macierz profili użytkowników (D')
48 #min_percentage = minimalny prog wielkości klastra
49 def find_optimal_clusters(X_D_prime, min_percentage=0.15):
50     print(f" [TC] Rozpoczęcie wyszukiwania optymalnych Klastrow Typologii (TC)...")
51
52     #prog wielkości, tj. minimalna liczba użytkowników w klastrze
53     threshold = int(min_percentage * len(X_D_prime))
54     print(f" [TC] Minimalny rozmiar klastra (15%): {threshold} użytkowników")
55
56     best_dunn = -1
57     best_labels = None
58
59     #testowanie podziałów na 2,3...10 klastrow
60     for k in range(2, 11):
61         model = AgglomerativeClustering(n_clusters=k, linkage='ward')
62         labels = model.fit_predict(X_D_prime)
63
64         #identyfikowanie małych klastrów
65         unique, counts = np.unique(labels, return_counts=True)
66         labels_adj = labels.copy()
67         small_clusters = [(lab, count) for lab, count in zip(unique, counts) if count > 0 and count < threshold]
68         small_clusters.sort(key=lambda x: x[1])
69
70         #naprawianie* za małych klastrów
71         for lab, count in small_clusters:
72             #obliczanie środka za małego klastra
73             if np.sum(labels_adj == lab) == 0: continue
74             centroid = X_D_prime[labels_adj == lab].mean(axis=0)
75
76             #szukanie najbliższego dużego klastra
77             centers = []
78             remaining_unique = np.unique(labels_adj)
79             for lab2 in remaining_unique:
80                 if lab2 != lab and np.sum(labels_adj == lab2) >= threshold:
81                     centers.append((lab2, np.linalg.norm(centroid - X_D_prime[labels_adj == lab2].mean(axis=0))))
82
83             #polaczenie obu klastrów
84             if centers:
85                 nearest_tc_label = min(centers, key=lambda x: x[1])[0]
86                 labels_adj[labels_adj == lab] = nearest_tc_label
87
88             #porzadkowanie etykiet po laczeniu klastrów
89             unique_final = np.unique(labels_adj)
90             label_map = {old: new for new, old in enumerate(unique_final)}
91             labels_adj = np.array([label_map[l] for l in labels_adj])
92
93             #ocena jakości podziału na grupy
94             dunn = dunn_index(X_D_prime, labels_adj)
95
96             #porownanie biezacego podzialu z najlepszym uzyskanym
97             #i zapamiatanie ukladu etykiet, jesli nowy podzial jest lepszy
98             if dunn > best_dunn:
99                 best_dunn = dunn
100                 best_labels = labels_adj.copy()
101
102     print(f" [TC] Znalezione optymalna liczba Klastrow Typologii: {len(np.unique(best_labels))}")
103     return best_labels
```

1. Obliczenie minimalnej wymaganej liczby użytkowników w klastrze, aby uniknąć zbyt małych grup
2. Testowanie podziałów na od 2 do 10 grup przy użyciu klastrowania hierarchicznego
3. Identyfikowanie zbyt małych pod względem liczby użytkowników grup i przyłączenie ich do najbliższego dużego klastra
4. Obliczenie wskaźnika Dunna dla skorygowanego podziału w celu zmierzenia, jak dobrze odseparowane i zwarte są grupy
5. Zwrócenie podziału, który uzyskał najwyższy wskaźnik jakości (najwyższy wskaźnik Dunna)

Implementacja - M2, klastry wewnętrzne

```
1 #methodology_m2.py
2 import numpy as np
3 from sklearn.cluster import KMeans
4 from sklearn.metrics import pairwise_distances
5
6 #Tworzenie klastrów wewnętrznych
7 #
8 #X_train_features = macierz surowych obserwacji treningowych (macierz D)
9 #obs_tc_labels = etykiety do którego klastra typologii (TC) należą obserwacje
10 #k_internal = liczba klastrów wewnętrznych (IC) do utworzenia w klastrach typologii
11 def create_internal_clusters(X_train_features, obs_tc_labels, k_internal=5):
12     #identyfikacja utworzonych klastrów wewnętrznych na etapie M1
13     ic_centroids_map = {}
14     unique_tcs = np.unique(obs_tc_labels)
15
16     print(f" [M2] Tworzenie Klastrów Wewnętrznych (IC) dla {len(unique_tcs)} TC...")
17
18     #przetwarzanie klastrów typologii
19     for tc_label in unique_tcs:
20         #wyciąganie surowych obserwacji należące do aktualnego klastra
21         observations_for_tc = X_train_features[obs_tc_labels == tc_label]
22
23         #dzielenie dane jednego klastra na mniejsze podgrupy (klastry wewnętrzne)
24         kmeans = KMeans(n_clusters=k_internal, random_state=42, n_init=10)
25         kmeans.fit(observations_for_tc)
26
27         #obliczanie środków nowych podgrup służących do przyciągania danych
28         #nowych użytkowników w fazie włączania
29         ic_centroids_map[tc_label] = kmeans.cluster_centers_
30
31     print(" [M2] Konfiguracja M2 zakończona. Mapa centroidów IC gotowa.")
32
33     #słownik o kluczach w postaci numerów TC
34     #i wartościach w postaci tablic ze współrzędnymi centroidów
35     return ic_centroids_map
36
```

1. Identyfikacja unikalnych klastrów typologii (TC), które zostały utworzone w etapie M1
2. Wyodrębnienie ze zbioru treningowego surowych obserwacji (cech) należących wyłącznie do aktualnie przetwarzanego klastra
3. Podział danych wewnątrz każdego klastra na mniejsze podgrupy (klastry wewnętrzne [IC]), przy użyciu algorytmu k-średnich
4. Obliczenie i zapisanie centroidów klastrów wewnętrznych jako punktów odniesienia w przestrzeni cech, służących do dopasowywania nowych użytkowników w fazie włączania

Implementacja - M2, przypisywanie użytkownika

```
38 #Włączanie nowego użytkownika (bez etykietowanych danych)
39 #d najlepiej pasującego klastra typologii (TC)
40 #
41 #X_new_user = nieetykietowane obserwacje nowego użytkownika (macierz 0 x F)
42 #ic_centroids_map = centroidy klastrów wewnętrznych dla każdego klastra typologii
43 def assign_new_user_M2(X_new_user, ic_centroids_map):
44     total_distances = {}
45
46     for tc_label, ic_centroids in ic_centroids_map.items():
47         #stworzenie macierzy odległości poprzez porównanie
48         #każdej obserwacji nowego użytkownika z każdym
49         #centroidem klastra wewnętrznego
50         dists = pairwise_distances(X_new_user, ic_centroids)
51
52         #znalezienie najlepszego dopasowania dla każdej obserwacji
53         min_dists_per_obs = dists.min(axis=1)
54
55         #agregacja wyników dla całego klastra typologii
56         #reprezentująca całkowity błąd dopasowania użytkownika
57         #do danej typologii
58         tc_total_dist = min_dists_per_obs.sum()
59         total_distances[tc_label] = tc_total_dist
60
61     #wybranie klastra typologii z najniższą sumą (najmniejsza łączna odległość)
62     assigned_tc_label = min(total_distances, key=total_distances.get)
63
64     print(" [M2] Obliczone sumaryczne odległości do TC:")
65     for tc, dist in total_distances.items():
66         print(f"    -> TC {tc}: {dist:.2f}")
67
68     return assigned_tc_label
```

1. Obliczenie macierzy odległości pomiędzy każdą obserwacją nowego użytkownika a wszystkimi centroidami klastrów wewnętrznych (IC) w ramach analizowanego klastra typologii
2. Wyznaczenie minimalnej odległości dla każdej pojedynczej obserwacji do najbliższego centroidu
3. Agregacja wyników poprzez zsumowanie minimalnych odległości, co stanowi całkowitą miarę dopasowania użytkownika do danej typologii
4. Przypisanie użytkownika do tego klastra typologii (TC), dla którego obliczona suma odległości była najmniejsza

Symulacja - dane i profilowanie

```
1 #main.py
2 import numpy as np
3
4 from data_processing import transform_matrix_M1
5 from clustering_tc import find_optimal_clusters
6 from methodology_m2 import create_internal_clusters, assign_new_user_M2
7
8 if __name__ == "__main__":
9
10     np.random.seed(777)
11
12     print("--- ROZPOCZECIE SYMULACJI METODOLOGII Z ARTYKULU ---")
13
14     ###0. Konfiguracja i generowanie danych
15     F_features = 57 #liczba cech fizjologicznych
16     TC_classes = 2 #liczba klas emocji (0 - brak strachu, 1 - strach)
17
18     #symulowanie danych: 30 uzytkownikow po 40 obserwacji
19     N_train_users = 30
20     O_obs_per_user = 40
21     O_total_train_obs = N_train_users * O_obs_per_user
22
23     #macierz D (obserwacje) z wartosciami symulujacymi odczyty z czujnikow
24     X_D_matrix = np.random.rand(O_total_train_obs, F_features)
25
26     #etykiety emocji dla kazdej obserwacji w X_D_matrix
27     X_D_labels = np.random.randint(0, TC_classes, size=O_total_train_obs)
28
29     #identyfikatory uzytkownikow dla kazdej obserwacji
30     X_D_user_ids = np.repeat(np.arange(N_train_users), O_obs_per_user)
31
32     print(f"Konfiguracja: {N_train_users} ochotnikow, {O_obs_per_user} obs/ochotnika")
33     print("-" * 40)
34
35     ###
36
37     ###1. Profilowanie (data_processing.py)
38     #transformacja M1 macierzy D z obserwacjami na macierz D' z profilami uzytkownikow
39     matrix_D_prime = transform_matrix_M1(X_D_matrix, X_D_user_ids, X_D_labels)
40
41     X_for_clustering = matrix_D_prime.values #konwersja macierzy D' na tablice NumPy
42     user_id_map = matrix_D_prime.index.to_numpy() #tablica z ID uzytkownikow
43
44     print("-" * 40)
```

--- ROZPOCZECIE SYMULACJI METODOLOGII Z ARTYKULU ---
Konfiguracja: 30 ochotnikow, 40 obs/ochotnika

X_D_matrix [1200, 57]

	0	1	2	3	4	...
0	0.152664	0.302357	0.0620364	0.45986	0.835253	0.9...
1	0.922232	0.0912056	0.315122	0.528022	0.328062	0.4...
2	0.7281	0.398508	0.10583	0.398583	0.521964	0.1...
3	0.930602	0.0661946	0.124188	0.828426	0.368128	0.0...
4	0.175827	0.196484	0.841296	0.418207	0.313451	0.6...
5	0.758751	0.415086	0.772361	0.611176	0.17912	0.4...
6	0.808267	0.757753	0.781256	0.383026	0.530786	0.1...
7	0.558074	0.0366963	0.756786	0.32432	0.0143273	0.9...
8	0.0708492	0.174937	0.429039	0.528041	0.327027	0.4...

[M1] Rozpoczenie transformacji D (0=1200) do D'...
[M1] Transformacja zakonczona. Kszalt D': (30, 228)

Matrix_D_prime [30, 228]

user_id	feature_0_mean_0	feature_0_mean_1	feature_0_std_0	feature_0_std_1	feature_1_mean_0
0	0.535471	0.491923	0.283658	0.340261	0.498832
1	0.502797	0.480437	0.297952	0.307048	0.310451
2	0.522715	0.595698	0.321405	0.301599	0.51004
3	0.43151	0.491261	0.297704	0.281607	0.57793
4	0.423179	0.49297	0.28866	0.276471	0.44095
5	0.438183	0.458656	0.265098	0.294778	0.432778
6	0.558517	0.489529	0.239779	0.287724	0.484524
7	0.561947	0.488927	0.223483	0.316289	0.543572
8	0.471877	0.419391	0.324025	0.294772	0.437054
9	0.460678	0.469917	0.294267	0.327604	0.546804
10	0.612484	0.529534	0.235438	0.295281	0.463742
11	0.400322	0.529584	0.321077	0.286947	0.557545
12	0.608286	0.640202	0.256968	0.284681	0.501871

Symulacja - klastry typologii i wewnętrzne

```
46 ###2. Klastrowanie TC (clustering_tc.py)
47 #szukanie najlepszego sposobu podziału użytkowników na typologie
48 min_threshold_users = int(0.15 * N_train_users) #min. liczba użytkowników do utworzenia klastra
49 best_tc_labels = find_optimal_clusters(X_for_clustering, min_threshold_users) #optymalne klastry
50
51 print(f"Wynik klastrowania TC: {len(np.unique(best_tc_labels))} klastrow.")
52 print("-" * 40)
53 ###
54
55 ###3. Konfiguracja M2 (methodology_m2.py) ---
56 #mapa ID użytkowników i odpowiadających im klastrow typologii
57 tc_label_map = {user_id: tc_label for user_id, tc_label in zip(user_id_map, best_tc_labels)}
58
59 #wektor z obserwacjami i przypisanymi do nich klastrami
60 obs_tc_labels = np.array([tc_label_map[user_id] for user_id in X_D_user_ids])
61
62 #tworzenie klastrow wewnętrznych
63 ic_map = create_internal_clusters(X_D_matrix, obs_tc_labels, k_internal=5)
64 print("-" * 40)
65 ###
```

```
[TC] Rozpoczęcie wyszukiwania optymalnych Klastrow Typologii (TC)...
[TC] Minimalny rozmiar klastra (15%): 120 użytkowników
[TC] Znaleziono optymalna liczba Klastrow Typologii: 9
Wynik klastrowania TC: 9 klastrow.
```

best_tc_labels [30]

	0
0	6
1	5
2	5
3	0
4	0
5	0
6	4
7	4

```
[M2] Tworzenie Klastrow Wewnętrznych (IC) dla 9 TC...
[M2] Konfiguracja M2 zakończona. Mapa centroidów IC gotowa.
```

ic_map [9, [5,57]]

	0	1	2	3	4	5	6
0	0.490609	0.534784	0.557131	0.560885	0.528591	0.527518	0.365416
1	0.607098	0.435525	0.626799	0.436929	0.636504	0.5198	0.593826
2	0.365255	0.656467	0.394648	0.551589	0.481945	0.654583	0.52328
3	0.301064	0.480433	0.541709	0.437969	0.473311	0.441273	0.511524
4	0.422806	0.421479	0.265039	0.43463	0.428614	0.393213	0.437063

Symulacja - dodanie nowego użytkownika

```
66
67 ###4: Włączanie nowego użytkownika (methodology_m2.py)
68 print("Symulacja: Pojawia się nowy użytkownik z 20 nieetykietowanymi obserwacjami...")
69 O_new_user_obs = 20 #liczba obserwacji
70 X_new_user = np.random.rand(O_new_user_obs, F_features) #cechy bez informacji o emocjach
71
72 #przypisanie do klastra
73 final_cluster = assign_new_user_M2(X_new_user, ic_map)
74
75 print("\n--- ZAKONCZENIE SYMULACJI ---")
76 print(f"Wynik: Nowy użytkownik został przypisany do Klastra Typologii (TC) nr: {final_cluster}")
77 ###
```

```
-----
Symulacja: Pojawia się nowy użytkownik z 20 nieetykietowanymi obserwacjami...
[M2] Obliczone sumaryczne odległości do TC:
-> TC 0: 42.67
-> TC 1: 42.97
-> TC 2: 42.64
-> TC 3: 43.26
-> TC 4: 42.94
-> TC 5: 43.58
-> TC 6: 43.47
-> TC 7: 44.67
-> TC 8: 43.10
```

Name	Type	Size	Value
final_cluster	int32	1	2

```
--- ZAKONCZENIE SYMULACJI ---
Wynik: Nowy użytkownik został przypisany do Klastra Typologii (TC) nr: 2
```

Dziękuję za uwagę!