



Министерство науки и высшего образования Российской Федерации  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
**«Московский государственный технический университет имени  
Н.Э. Баумана**  
(национальный исследовательский университет)  
(МГТУ им. Н.Э. Баумана)

---

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

## **Лабораторная работа № 10 по дисциплине «Методы машинного обучения»**

**Тема** Кластерный анализ

**Студент** Сапожков А.М.

**Группа** ИУ7-23М

**Преподаватель** Солодовников В.И.

# Содержание

<b>ВВЕДЕНИЕ</b>	4
<b>1 Аналитическая часть</b>	5
1.1 Иерархическая кластеризация	5
1.2 Алгоритм HDBSCAN	6
<b>2 Технологическая часть</b>	7
2.1 Средства реализации	7
2.2 Реализация алгоритмов	7
<b>3 Исследовательская часть</b>	17
3.1 Агломеративная кластеризация	17
3.1.1 Кластеризация исходных данных	17
3.1.2 Кластеризация векторов, полученных после снижения размерности ис- ходных данных с помощью алгоритма UMAP	19
3.1.3 Подбор количества кластеров	22
3.2 Алгоритм HDBSCAN	25
3.2.1 Оптимизация гиперпараметров по метрике ARI	25
3.2.2 Оптимизация гиперпараметров по метрике DBI	27
<b>ЗАКЛЮЧЕНИЕ</b>	29

# ВВЕДЕНИЕ

Кластеризация является одной из важнейших задач в области анализа данных. Она используется для группировки объектов в такие подмножества (кластеры), в которых объекты внутри каждого кластера максимально схожи, а объекты из разных кластеров максимально различны. Задача кластеризации широко применяется в различных областях, включая обработку текстов, анализ изображений, биоинформатику и многие другие.

Целью данной лабораторной работы является изучение алгоритмов кластеризации на примере кластерного анализа результатов социологического исследования. Для достижения поставленной цели необходимо выполнить следующие задачи.

1. Провести кластерный анализ данных. Использовать не менее двух алгоритмов кластеризации (например: иерархический, К-средних, DBSCAN и др.). Варьировать различные значения гиперпараметров и тип расстояний.
2. Оценить работу алгоритмов с использованием внешних и внутренних мер оценки качества (в том числе, построить таблицу сопряжённости с учётом классов Шкалы Кантила). Определить оптимальное количество кластеров и их структуру.

# 1 Аналитическая часть

## 1.1 Иерархическая кластеризация

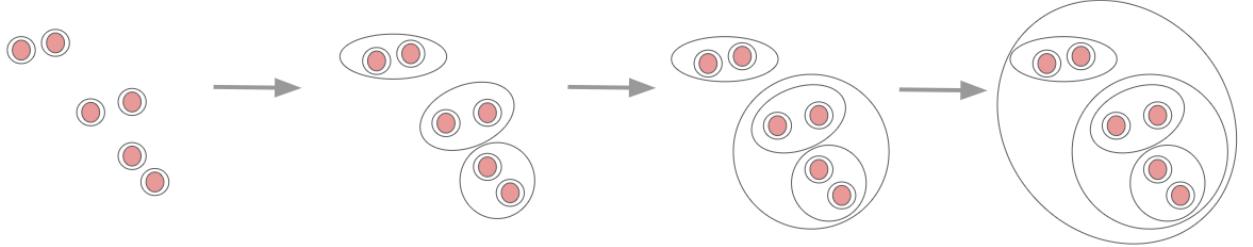
Среди алгоритмов иерархической кластеризации выделяются два основных типа.

1. **Нисходящая кластеризация (divisive)**: Работает по принципу «сверху-вниз»: в начале, все объекты принадлежат одному кластеру. В ходе итеративного процесса крупные кластеры разделяются на более мелкие. Такие задачи называются задачами таксономии. При этом, получается дерево кластеров (дендrogramма).

2. **Восходящая кластеризация (agglomerative)**: Изначально каждый элемент множества является отдельным кластером. Процесс образования новых кластеров заключается в объединение некоторых кластеров в один на основе заданного расстояния. В итоге итеративного объединения получаем дерево, которое сходится к одному кластеру.

На рисунке 1.1 представлена иллюстрация работы иерархической кластеризации.

## AGGLOMERATIVE CLUSTERING



## DIVISIVE CLUSTERING

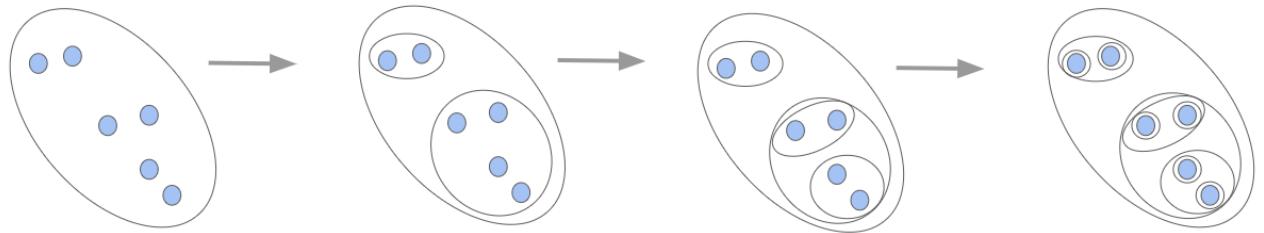


Рисунок 1.1 — Иерархическая кластеризация

## 1.2 Алгоритм HDBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) — основанная на плотности пространственная кластеризация для приложений с шумами.

Если дан набор точек в некотором пространстве, алгоритм группирует вместе точки, которые тесно расположены (точки со многими близкими соседями), помечая как выбросы точки, которые находятся одиноко в областях с малой плотностью (ближайшие соседи которых лежат далеко). DBSCAN развивает идею кластеризации с помощью выделения связных компонент.

Алгоритм HDBSCAN расширяет DBSCAN за счёт автоматического определения оптимальных параметров кластеризации (например, не требует ручного выбора радиуса  $\varepsilon$ ), что особенно полезно для данных с переменной плотностью кластеров. Кроме того, HDBSCAN строит иерархическую структуру кластеров, что позволяет выделять устойчивые группы даже при сложном распределении данных, и эффективно обрабатывает выбросы. Алгоритм также демонстрирует инвариантность к масштабу данных, что упрощает его применение без предварительной нормализации исходных данных.

## 2 Технологическая часть

### 2.1 Средства реализации

В качестве языка программирования для реализации алгоритмов был выбран язык программирования Python ввиду наличия библиотек для обучения регрессионных моделей, таких как sklearn и numpy.

### 2.2 Реализация алгоритмов

На листинге 2.1 представлена реализация алгоритма кластеризации респондентов, принимавших участие в социологическом исследовании.

Листинг 2.1 — Кластеризация респондентов социологического исследования

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import re

import warnings
warnings.filterwarnings('ignore')

from google.colab import drive
drive.mount('/content/drive')

dataset = pd.read_excel('/content/drive/MyDrive/Colab Notebooks/ml\_
_lab\_08/ММО_ЛР8_Исходные_данные.xlsx')
dataset = dataset.drop(['Респондент'], axis=1)
dataset['Сообщество'] = dataset['Сообщество'].apply(lambda it: int(re.
    findall(r'\b\d+\b', it)[0]))
class_names = list(set(dataset['Ощущаемое.счастье']))
class_names.remove('Неизвестно')

from operator import itemgetter
group = {
    0: 0,
    1: 0,
    2: 0,
    3: 0,
    4: 1,
    5: 1,
```

```

    6: 1,
    7: 2,
    8: 2,
    9: 2,
}

rank_mapping = {
    'Prospering': 0,
    'Thriving': 1,
    'Blooming': 2,
    'Doing well': 3,
    'Just ok': 4,
    'Coping': 5,
    'Struggling': 6,
    'Suffering': 7,
    'Depressed': 8,
    'Hopeless': 9,
}

labels = sorted(list(set(class_names)), key=lambda v: rank_mapping[v])
dataset_knowns = dataset[dataset['Ощущаемое.счастье'] != 'Неизвестно']
y_true = np.array([labels.index(xi) for xi in dataset_knowns['
    Ощущаемое.счастье'][:10000].to_numpy()])
X = dataset_knowns.drop('Ощущаемое.счастье', axis=1).values[:10000]
labels_true = np.array([labels[i] for i in y_true])

import umap
!pip install umap-learn[plot]
import umap.plot
from umap import UMAP
import time
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import adjusted_rand_score, silhouette_score,
    davies_bouldin_score, calinski_harabasz_score
import matplotlib.pyplot as plt

from sklearn.metrics.pairwise import pairwise_distances

def evaluate_clustering(X, y_true, distance_threshold, metric='
    euclidean', linkage='ward'):
    clustering = AgglomerativeClustering(n_clusters=None,

```

```

        distance_threshold=distance_threshold, metric=metric, linkage=
        linkage)
    start = time.process_time()
    y_pred = clustering.fit_predict(X)
    diff = time.process_time() - start
    n_clusters = len(np.unique(y_pred))
    return y_pred, n_clusters, {
        'Time': diff,
        'ARI': adjusted_rand_score(y_true, y_pred),
    }

def find_optimal_clustering(X, y_true, metric='euclidean', linkage='ward',
                            iterations=1000):
    optimal_threshold = None
    best_metrics = None
    y_optimal = None
    n_clusters_optimal = None
    dist = pairwise_distances(X, metric=metric)
    for threshold in np.linspace(np.min(dist), np.max(dist), iterations):
        y_pred, n_clusters, metrics = evaluate_clustering(dist, y_true,
                                                          threshold, 'precomputed', linkage)
        ari = metrics['ARI']
        if best_metrics is None or ari > best_metrics['ARI']:
            best_metrics = metrics
            y_optimal = y_pred
            optimal_threshold = threshold
            n_clusters_optimal = n_clusters

    return optimal_threshold, best_metrics, y_optimal,
           n_clusters_optimal

import seaborn as sns

def plot_clustering_heatmap(X, y_true):
    distances = ['euclidean', 'manhattan', 'cosine']
    linkages = ['complete', 'average'] # , 'single'

    time, ari, n_clusters = np.ndarray((len(distances), len(linkages)))
    , np.ndarray((len(distances), len(linkages))), np.ndarray((len(
    distances), len(linkages)))

```

```

for i, metric in enumerate(distances):
    for j, linkage in enumerate(linkages):
        try:
            _, metrics, _, n = find_optimal_clustering(X, y_true, metric=
                metric, linkage=linkage, iterations=10)
            time[i, j], ari[i, j], n_clusters[i, j] = metrics['Time'],
                metrics['ARI'], n
        except Exception as e:
            print(e)
            time[i, j], ari[i, j], n_clusters[i, j] = 0, 0, 0

fig, (ax1, ax2, ax3) = plt.subplots(ncols=3, figsize=(15, 4))
sns.heatmap(time, ax=ax1, annot=True, cmap='Reds', xticklabels=
    linkages, yticklabels=distances)
sns.heatmap(ari, ax=ax2, annot=True, cmap='Reds', xticklabels=
    linkages, yticklabels=distances)
sns.heatmap(n_clusters, ax=ax3, annot=True, cmap='Reds',
    xticklabels=linkages, yticklabels=distances)
ax1.set_title('Time')
ax2.set_title('ARI score')
ax3.set_title('Clusters count')
plt.show()

from sklearn.metrics import pairwise_distances
from collections import Counter
from mpl_toolkits import mplot3d

def class_purity(y_true, y_pred, cls):
    class_mask = (y_true == cls)
    class_predictions = y_pred[class_mask]
    cluster_counts = Counter(class_predictions).values()
    purity = max(cluster_counts) / len(class_predictions)
    return purity

def plot_clustering(title, X, y_true, y_pred, metric):
    fig, plots = plt.subplots(2, 2, figsize=(12,12))
    fig.suptitle(title)
    plt.prism()

    n_clusters = len(np.unique(y_true))

```

```

purities = []

ax = fig.add_subplot(2, 2, 1, projection='3d') if X.shape[1] == 3
else plots[0, 0]
for i in range(n_clusters):
    digit_indices = (y_true == i)
    purities.append(class_purity(y_true, y_pred, i))
    dims = [X[digit_indices, i] for i in range(X.shape[1])]
    ax.set_title('Original')
    ax.scatter(*dims, label=f"Class {i}")
    ax.legend()

purities.append(np.average(purities))

avg_dist = np.zeros((n_clusters, n_clusters))
for i in range(n_clusters):
    for j in range(n_clusters):
        avg_dist[i, j] = pairwise_distances(
            X[y_true == i], X[y_true == j], metric=metric
        ).mean()
avg_dist /= avg_dist.max()
sns.heatmap(avg_dist, ax=plots[1, 0], annot=True, cmap='Reds',
            xticklabels=np.arange(n_clusters), yticklabels=np.arange(
            n_clusters))

inner_distances = [avg_dist[i, i] for i in range(n_clusters)]
inner_distances.append(np.average(inner_distances))
sns.heatmap([inner_distances, purities], ax=plots[1, 1], annot=True,
            , cmap='Reds', xticklabels=[*np.arange(n_clusters), 'avg'],
            yticklabels=['inner distance', 'purity'])

n_clusters = len(np.unique(y_pred))

ax = fig.add_subplot(2, 2, 2, projection='3d') if X.shape[1] == 3
else plots[0, 1]
for i in range(n_clusters):
    digit_indices = (y_pred == i)
    dims = [X[digit_indices, i] for i in range(X.shape[1])]
    ax.set_title('Prediction')
    ax.scatter(*dims, label=f"Cluster {i}")
    ax.legend()

```

```

plt.tight_layout()
plt.legend(loc='best')
plt.show()

umap_3d_embeddings = UMAP(n_components=3, random_state=7).
    fit_transform(X)
umap_3d_embeddings.shape

plot_clustering_heatmap(X, y_true)
threshold, metrics, y_pred_best, n_clusters = find_optimal_clustering
    (X, y_true, metric='euclidean', linkage='complete', iterations
     =100)
print(metrics, n_clusters)
plot_clustering('Agglomerative clustering', umap_3d_embeddings,
    y_true, y_pred_best, 'manhattan')

plot_clustering_heatmap(umap_3d_embeddings, y_true)
threshold, metrics, y_pred_best, n_clusters = find_optimal_clustering
    (umap_3d_embeddings, y_true, metric='manhattan', linkage='average'
     , iterations=100)
print(metrics, n_clusters)
plot_clustering('3D UMAP embeddings agglomerative clustering',
    umap_3d_embeddings, y_true, y_pred_best, 'manhattan')
y_pred_grouped = np.array(itemgetter(*y_pred_best)(group))
print(y_pred_grouped.shape)
y_true_grouped = np.array(itemgetter(*y_true)(group))
print(y_true_grouped.shape)

plot_clustering('3D UMAP embeddings agglomerative clustering',
    umap_3d_embeddings, y_true_grouped, y_pred_grouped, 'manhattan')
plot_clustering('3D UMAP embeddings agglomerative clustering',
    umap_3d_embeddings, y_true_grouped, y_pred_best, 'manhattan')

print(f"ARI: {adjusted_rand_score(y_true_grouped, y_pred_best)}")
print(f"DBI: {davies_bouldin_score(umap_3d_embeddings, y_pred_best)}")
    (original: {davies_bouldin_score(umap_3d_embeddings,
     y_true_grouped)}")
print(f"Silhouette: {silhouette_score(umap_3d_embeddings, y_pred_best
     , random_state=7)} (original: {silhouette_score(umap_3d_embeddings
     , y_true_grouped, random_state=7)})")

```

```

print(f"Calinski Harabasz: {calinski_harabasz_score(
    umap_3d_embeddings, y_pred_best)} (original: {
    calinski_harabasz_score(umap_3d_embeddings, y_true_grouped)})")

dbi_orig = davies_bouldin_score(X, y_true_grouped)
silhouette_orig = silhouette_score(X, y_true_grouped, random_state=7)
calinski_harabasz_orig = calinski_harabasz_score(X, y_true_grouped)

dbi_umap = davies_bouldin_score(umap_3d_embeddings, y_true_grouped)
silhouette_umap = silhouette_score(umap_3d_embeddings, y_true_grouped
    , random_state=7)
calinski_harabasz_umap = calinski_harabasz_score(umap_3d_embeddings,
    y_true_grouped)

dbi_scores = []
ari_scores = []
silhouette_scores = []
calinski_harabasz_scores = []

for k in range(2, 20):
    y_pred = AgglomerativeClustering(n_clusters=k, metric='euclidean',
        linkage='average').fit_predict(umap_3d_embeddings)
    dbi_scores.append(davies_bouldin_score(umap_3d_embeddings, y_pred))
    ari_scores.append(adjusted_rand_score(y_true_grouped, y_pred))
    silhouette_scores.append(silhouette_score(umap_3d_embeddings, y_pred,
        random_state=7))
    calinski_harabasz_scores.append(calinski_harabasz_score(
        umap_3d_embeddings, y_pred))

print(f"Optimal clusters count (DBI):           {np.argmin(
    dbi_scores)+2}")
print(f"Optimal clusters count (ARI):            {np.argmax(
    ari_scores)+2}")
print(f"Optimal clusters count (Silhouette):      {np.argmax(
    silhouette_scores)+2}")
print(f"Optimal clusters count (Calinski Harabasz): {np.argmax(
    calinski_harabasz_scores)+2}")

fig, plots = plt.subplots(2, 2, figsize=(12,12))
k_range = np.arange(2, 20)

```

```

plots[0, 0].plot(k_range, dbi_scores)
plots[0, 1].plot(k_range, ari_scores)
plots[1, 0].plot(k_range, silhouette_scores)
plots[1, 1].plot(k_range, calinski_harabasz_scores)

plots[0, 0].set_title('DBI')
plots[0, 1].set_title('ARI')
plots[1, 0].set_title('Silhouette score')
plots[1, 1].set_title('Calinski Harabasz score')

right_answer = len(np.unique(y_true))
plots[0, 0].vlines(right_answer, 0, 1, color='g', label='Right Answer')
plots[0, 1].vlines(right_answer, 0, 1, color='g', label='Right Answer')
plots[1, 0].vlines(right_answer, 0, 1, color='g', label='Right Answer')
plots[1, 1].vlines(right_answer, np.min(calinski_harabasz_scores), np.max(calinski_harabasz_scores), color='g', label='Right Answer')

plots[0, 0].hlines(dbi_orig, 2, 19, color='m', label='Original value from source')
plots[1, 0].hlines(silhouette_orig, 2, 19, color='m', label='Original value from source')
plots[1, 1].hlines(calinski_harabasz_orig, 2, 19, color='m', label='Original value from source')

plots[0, 0].hlines(dbi_umap, 2, 19, color='c', label='Original value from UMAP')
plots[1, 0].hlines(silhouette_umap, 2, 19, color='c', label='Original value from UMAP')
plots[1, 1].hlines(calinski_harabasz_umap, 2, 19, color='c', label='Original value from UMAP')

plots[0, 0].set_xticks(k_range)
plots[0, 1].set_xticks(k_range)
plots[1, 0].set_xticks(k_range)
plots[1, 1].set_xticks(k_range)

plots[0, 0].legend(loc='best')
plots[0, 1].legend(loc='best')

```

```

plots[1, 0].legend(loc='best')
plots[1, 1].legend(loc='best')

plt.show()

from hdbscan import HDBSCAN
y_pred = HDBSCAN().fit_predict(X)
plot_clustering('3D UMAP embeddings HDBSCAN clustering',
    umap_3d_embeddings, y_true_grouped, y_pred, 'euclidean')

def custom_ari(estimator, X):
    labels = estimator.fit_predict(X)
    core_samples_mask = np.zeros_like(labels, dtype=bool)
    core_samples_mask[estimator.labels_ >= 0] = True
    if not np.any(core_samples_mask):
        return 0.0
    true_labels = y_true[core_samples_mask]
    labels = labels[core_samples_mask]
    return adjusted_rand_score(true_labels, labels)

def custom_dbis(estimator, X):
    labels = estimator.fit_predict(X)
    core_samples_mask = np.zeros_like(labels, dtype=bool)
    core_samples_mask[estimator.labels_ >= 0] = True
    if not np.any(core_samples_mask):
        return 0.0
    true_labels = y_true[core_samples_mask]
    labels = labels[core_samples_mask]
    return -davies_bouldin_score(X[core_samples_mask], labels)

def coverage(estimator, X):
    labels = estimator.fit_predict(X)
    core_samples_mask = np.zeros_like(labels, dtype=bool)
    core_samples_mask[estimator.labels_ >= 0] = True
    return np.mean(core_samples_mask)

from sklearn.model_selection import GridSearchCV

def estimate_hdbscan(refit):
    param_grid = {
        'min_cluster_size': np.arange(500, 1500, 100),

```

```

        'min_samples': np.arange(1, 30, 3),
        'metric': ['euclidean', 'manhattan', 'cosine']
    }

    hdb = HDBSCAN()
    grid_search = GridSearchCV(hdb, param_grid, scoring={'ARI':
        custom_ari, 'DBI': custom_db, 'coverage': coverage}, refit=
        refit)
    grid_search.fit(umap_3d_embeddings)

    best_params = grid_search.best_params_

    y_pred = HDBSCAN(min_cluster_size = grid_search.best_params_['
        min_cluster_size'],
        min_samples = grid_search.best_params_['min_samples'],
        metric = grid_search.best_params_['metric']).fit_predict(
            umap_3d_embeddings)

    return best_params, y_pred

def visualize_dbSCAN(best_params, y_pred):
    print(f"Best parameters: {best_params}")
    core_samples_mask = np.zeros_like(y_pred, dtype=bool)
    core_samples_mask[y_pred >= 0] = True
    cov = np.mean(core_samples_mask)

    print(f"ARI: {adjusted_rand_score(y_true_grouped, y_pred)}")
    print(f"DBI: {davies_bouldin_score(umap_3d_embeddings, y_pred)}")
    print(f"ARI (covered): {adjusted_rand_score(y_true_grouped[
        core_samples_mask], y_pred[core_samples_mask])}")
    print(f"DBI (covered): {davies_bouldin_score(umap_3d_embeddings[
        core_samples_mask], y_pred[core_samples_mask])}")
    print(f"Covariance: {cov}")
    print(f"N clusters: {len(np.unique(y_pred))}")

    plot_clustering('3D UMAP embeddings HDBSCAN clustering',
        umap_3d_embeddings, y_true_grouped, y_pred, 'euclidean')

    best_params, y_pred = estimate_dbSCAN('ARI')
    visualize_dbSCAN(best_params, y_pred)

```

### **3 Исследовательская часть**

Для тестирования разработанного алгоритма применялась облачная платформа Google Colab, не требующая установки ПО на локальный компьютер.

#### **3.1 Агломеративная кластеризация**

##### **3.1.1 Кластеризация исходных данных**



Рисунок 3.1 — Сравнение метрик при использовании различных расстояний и критериев связи в алгоритме агломеративной кластеризации исходных данных

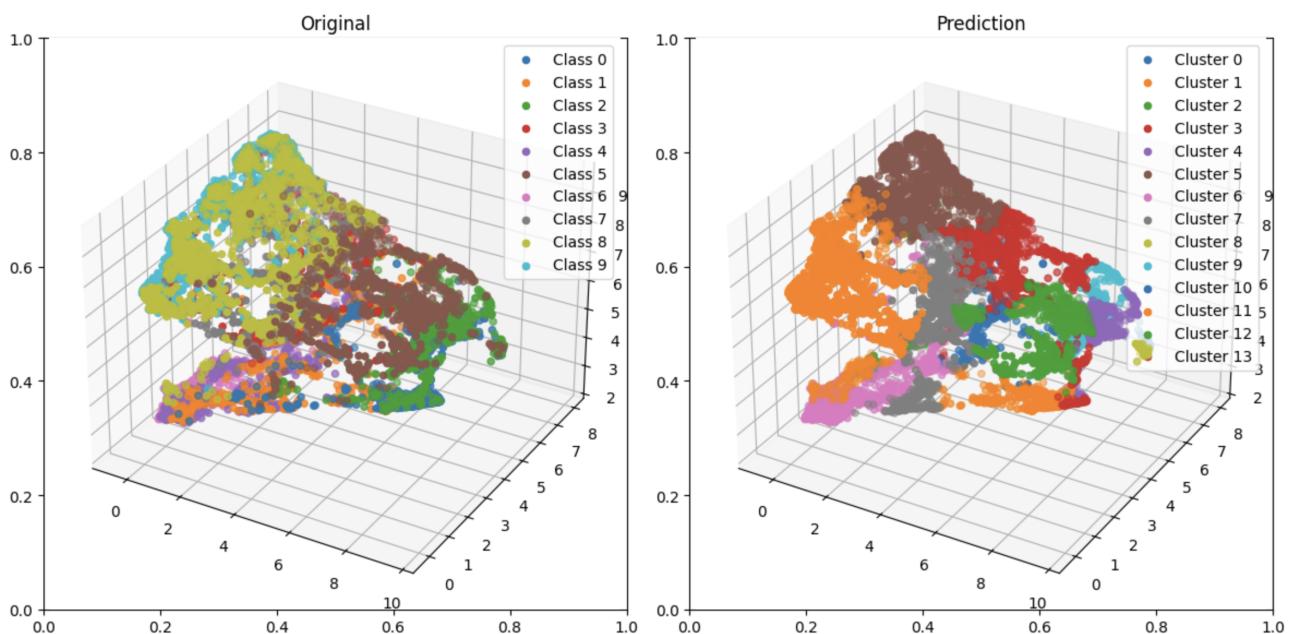


Рисунок 3.2 — Сравнение оригинального разбиения респондентов по шкале Кантрила с результатом кластеризации исходных данных (ARI: 0.18)

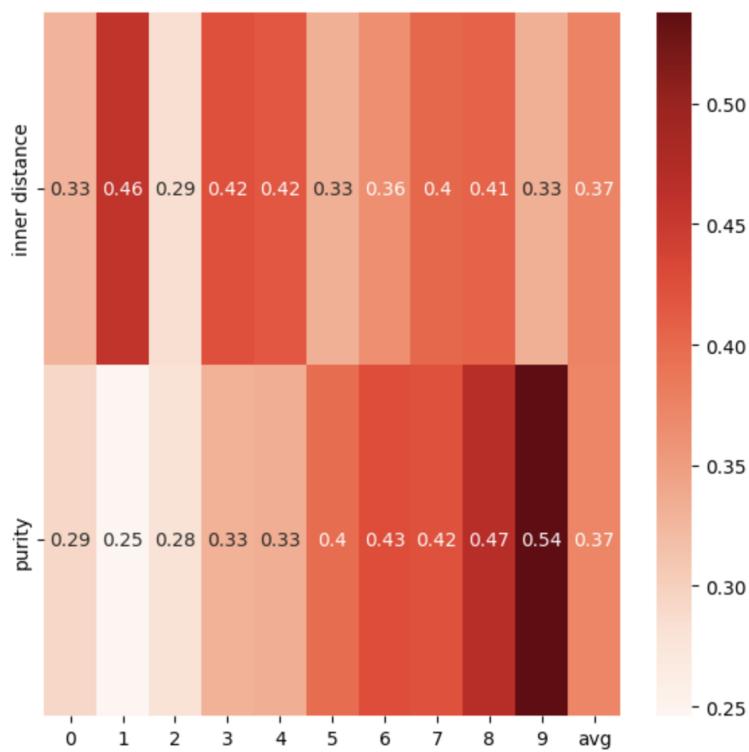


Рисунок 3.3 — Соотнесение внутрикластерных расстояний с чистотой исходных кластеров, отсортированных по шкале Кантрила)

### 3.1.2 Кластеризация векторов, полученных после снижения размерности исходных данных с помощью алгоритма UMAP

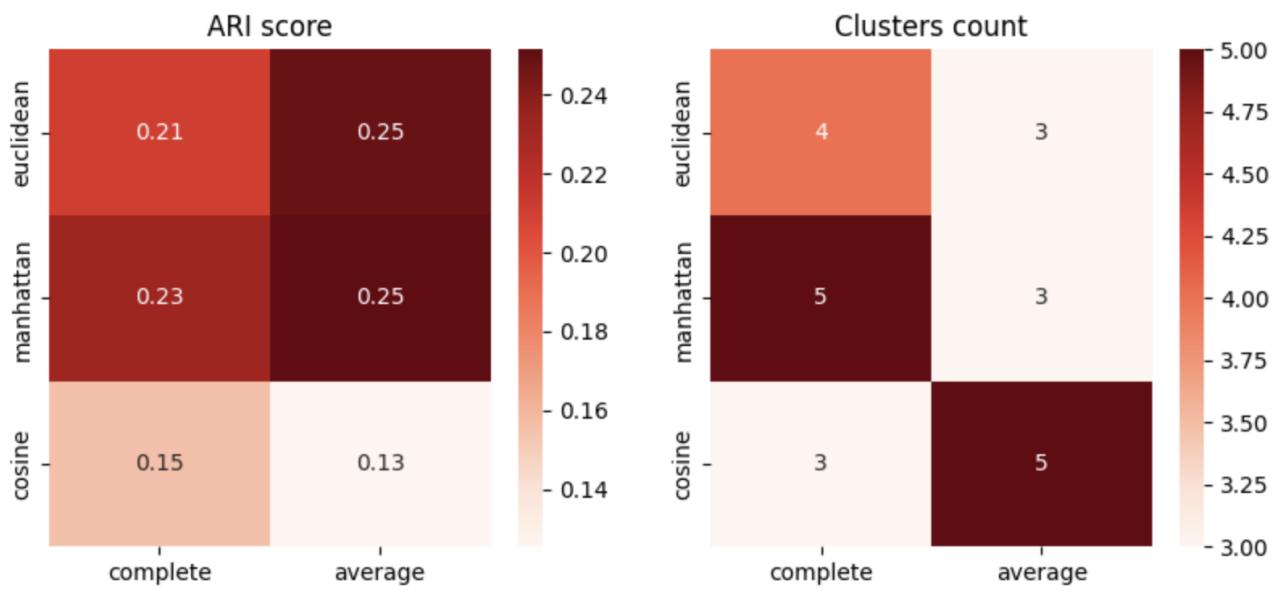


Рисунок 3.4 — Сравнение метрик при использовании различных расстояний и критериев связи в алгоритме агломеративной кластеризации векторов, полученных после снижения размерности исходных данных с помощью алгоритма UMAP

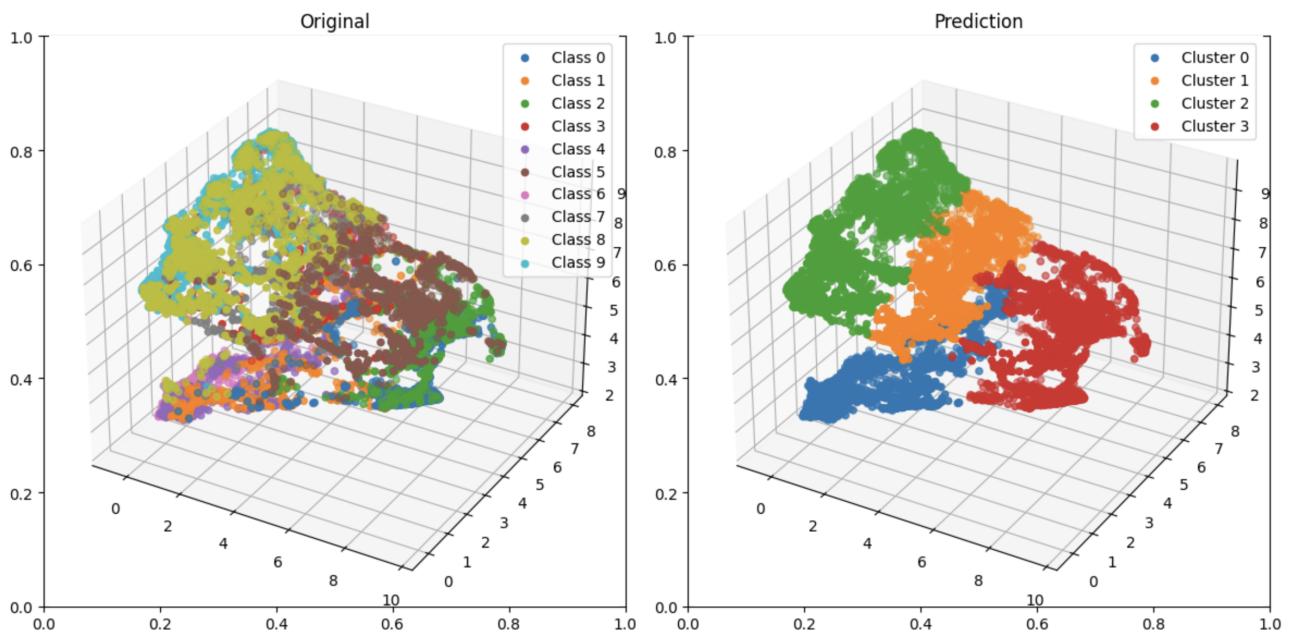


Рисунок 3.5 — Сравнение оригинального разбиения респондентов по шкале Кантрила с результатом кластеризации исходных данных (ARI: 0.27)

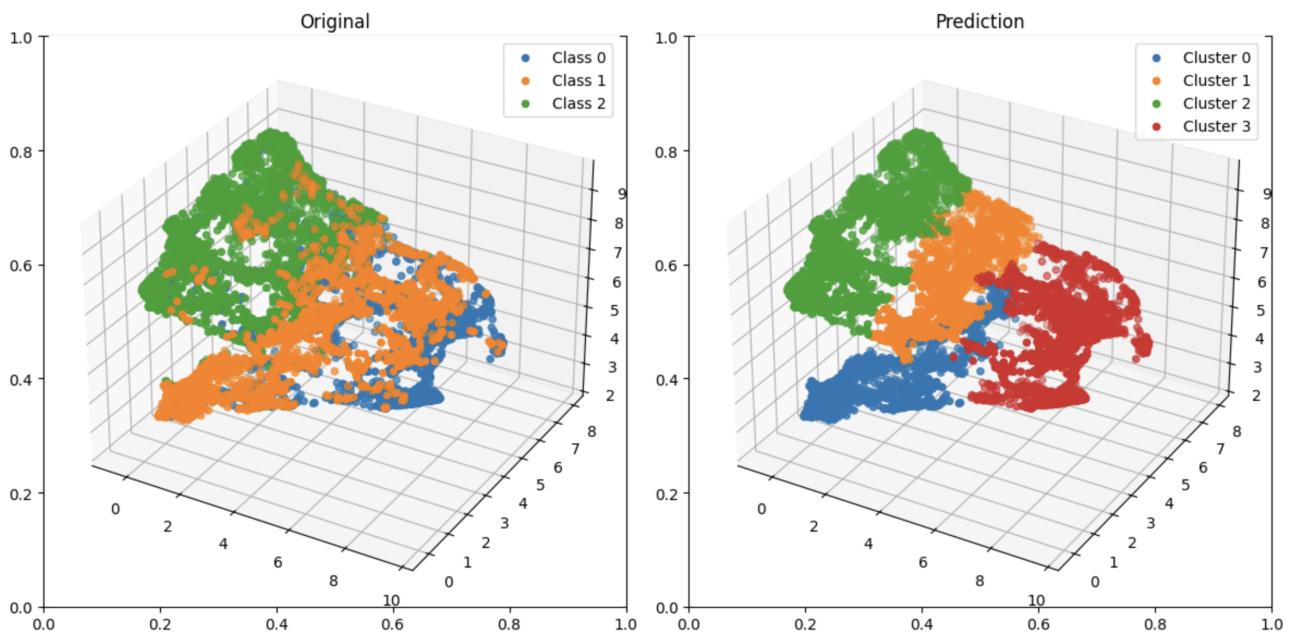


Рисунок 3.6 — Сравнение оригинального разбиения респондентов по укрупнённой шкале Кантрила с результатом кластеризации исходных данных (ARI: 0.37)

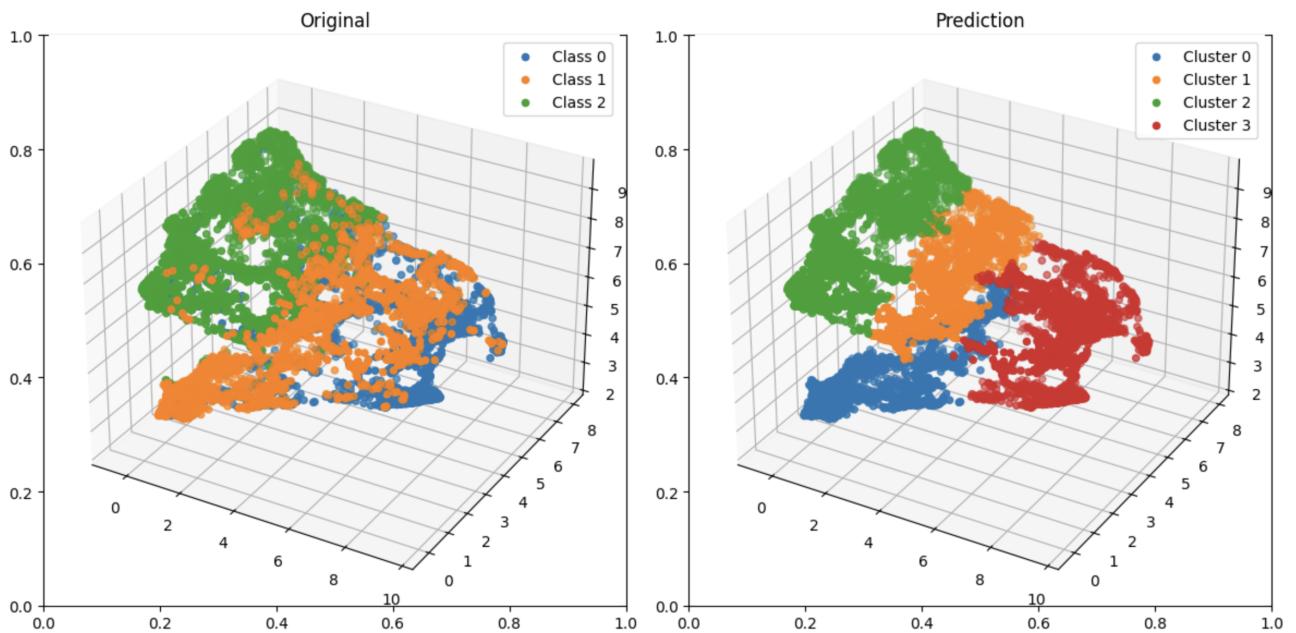


Рисунок 3.7 — Соотнесение внутрикластерных расстояний с чистотой исходных кластеров, отсортированных по укрупнённой шкале Кантрила)

### **3.1.3 Подбор количества кластеров**

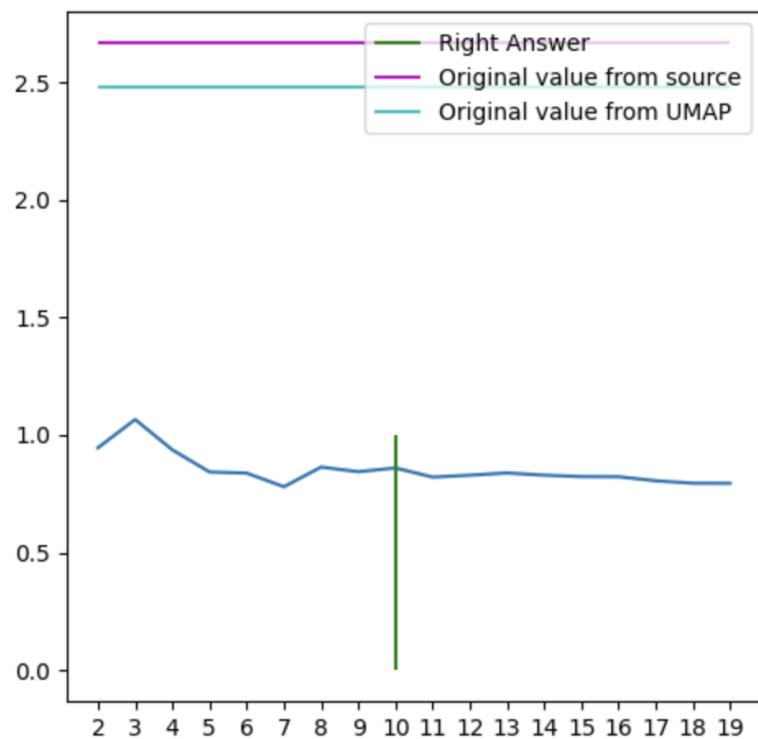


Рисунок 3.8 — Зависимость метрики DBI от заданного количества кластеров

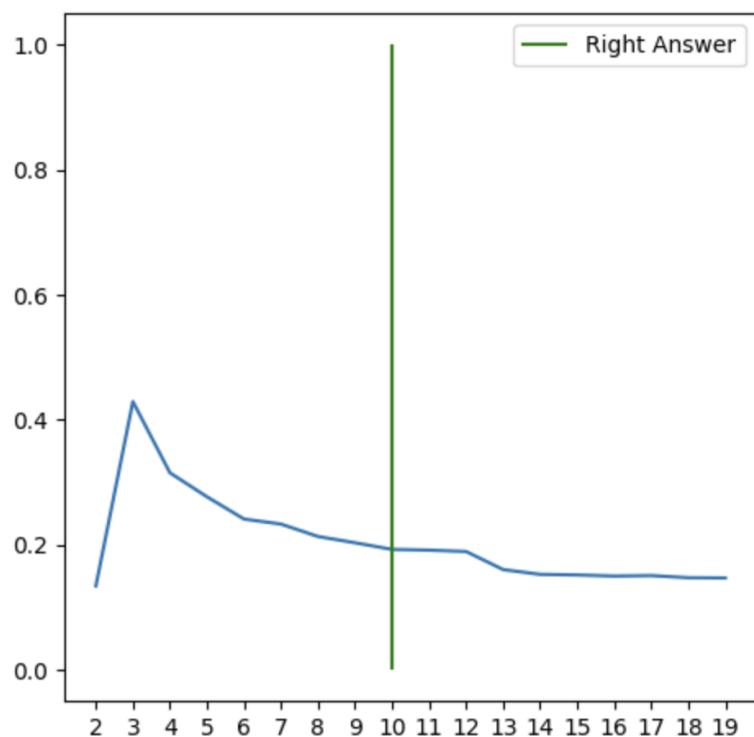


Рисунок 3.9 — Зависимость метрики ARI от заданного количества кластеров

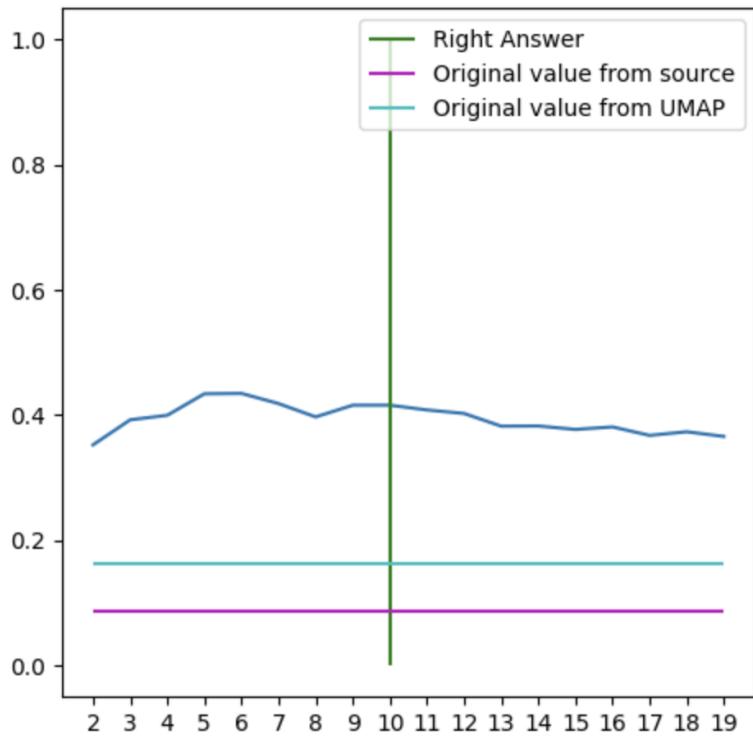


Рисунок 3.10 — Зависимость метрики Silhouette score от заданного количества кластеров

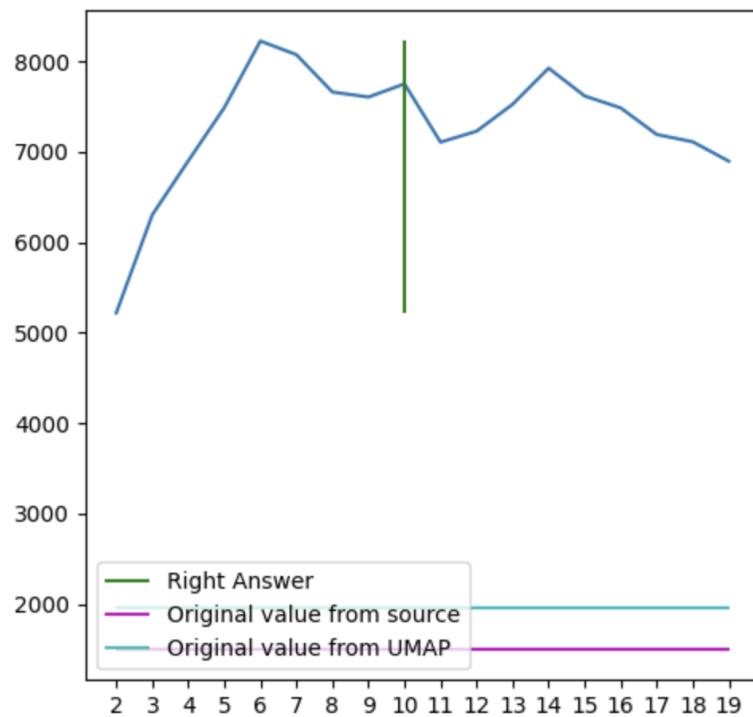


Рисунок 3.11 — Зависимость метрики Calinski Harabasz score от заданного количества кластеров

## **3.2 Алгоритм HDBSCAN**

### **3.2.1 Оптимизация гиперпараметров по метрике ARI**

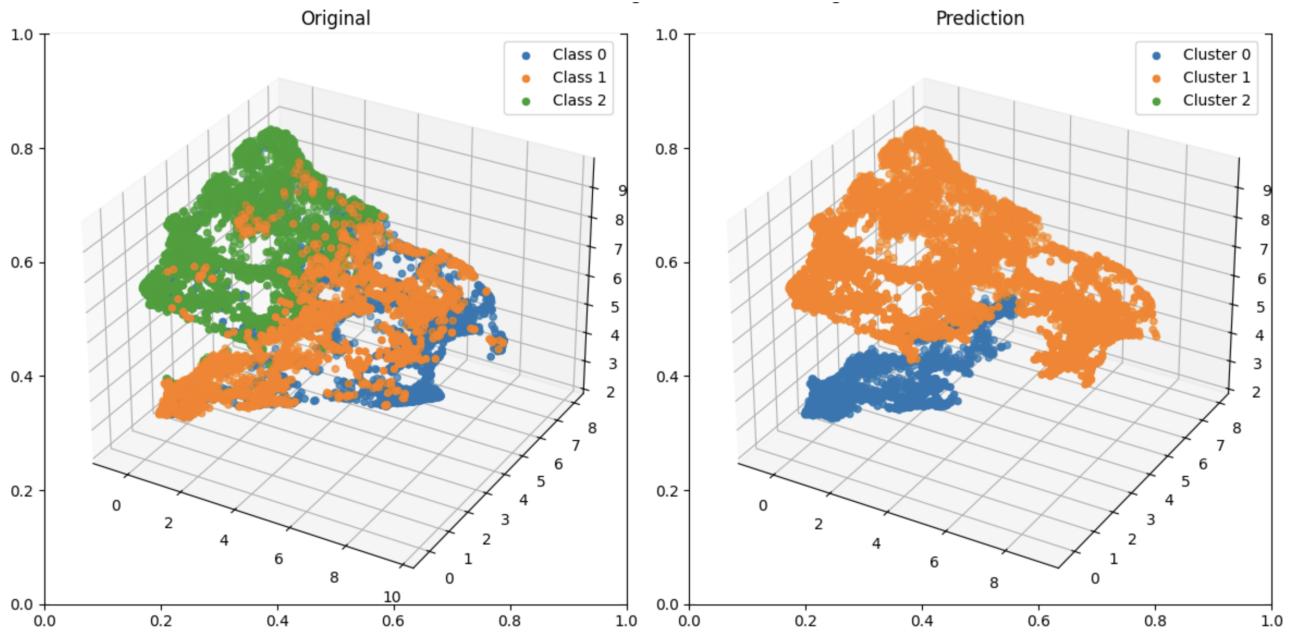


Рисунок 3.12 — Сравнение оригинального разбиения респондентов по укрупнённой шкале Кантрила с результатом кластеризации векторов, полученных после снижения размерности исходных данных с помощью алгоритма UMAP (ARI: 0.24)

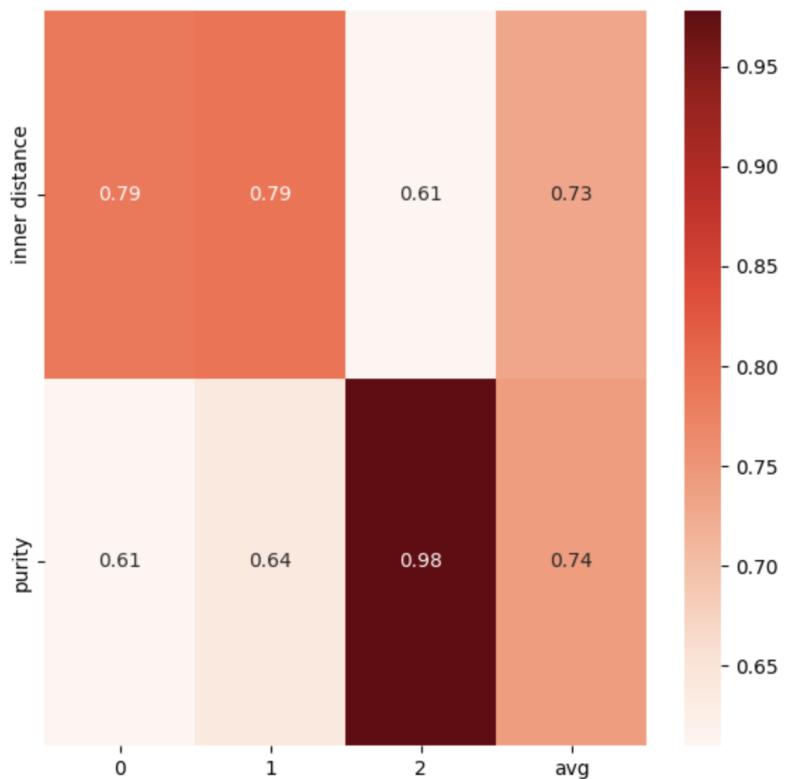


Рисунок 3.13 — Соотнесение внутрикластерных расстояний с чистотой исходных кластеров, отсортированных по укрупнённой шкале Кантрила)

### **3.2.2 Оптимизация гиперпараметров по метрике DBI**

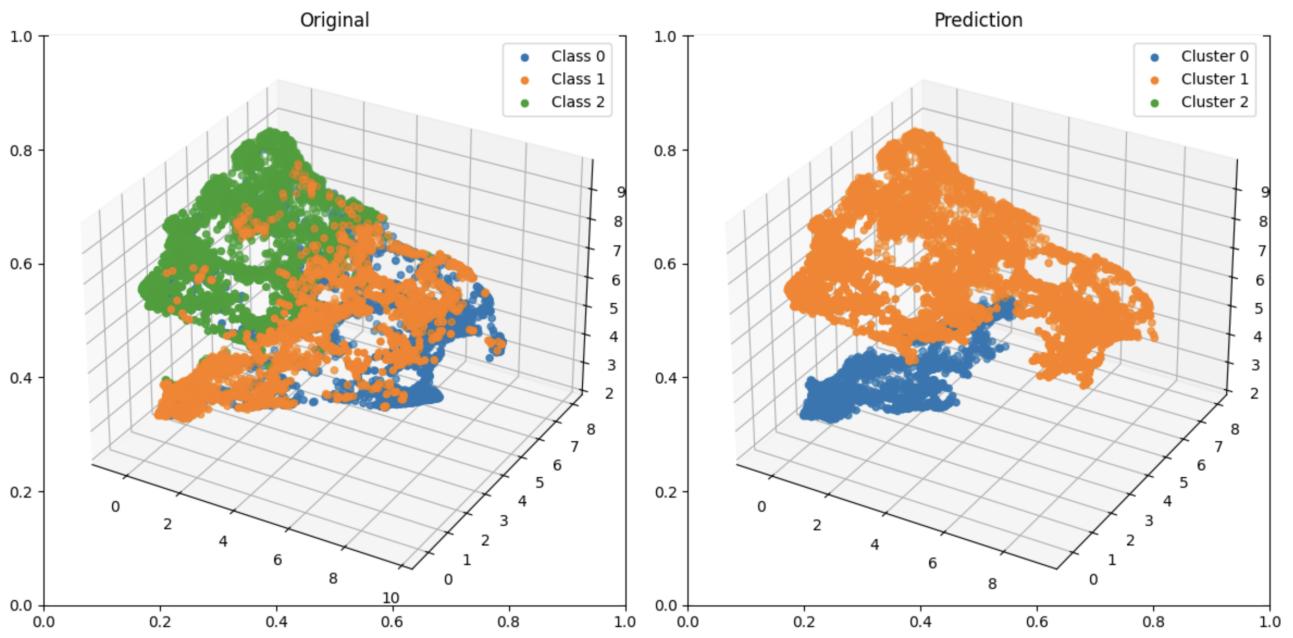


Рисунок 3.14 — Сравнение оригинального разбиения респондентов по укрупнённой шкале Кантрила с результатом кластеризации векторов, полученных после снижения размерности исходных данных с помощью алгоритма UMAP (ARI: 0.24)

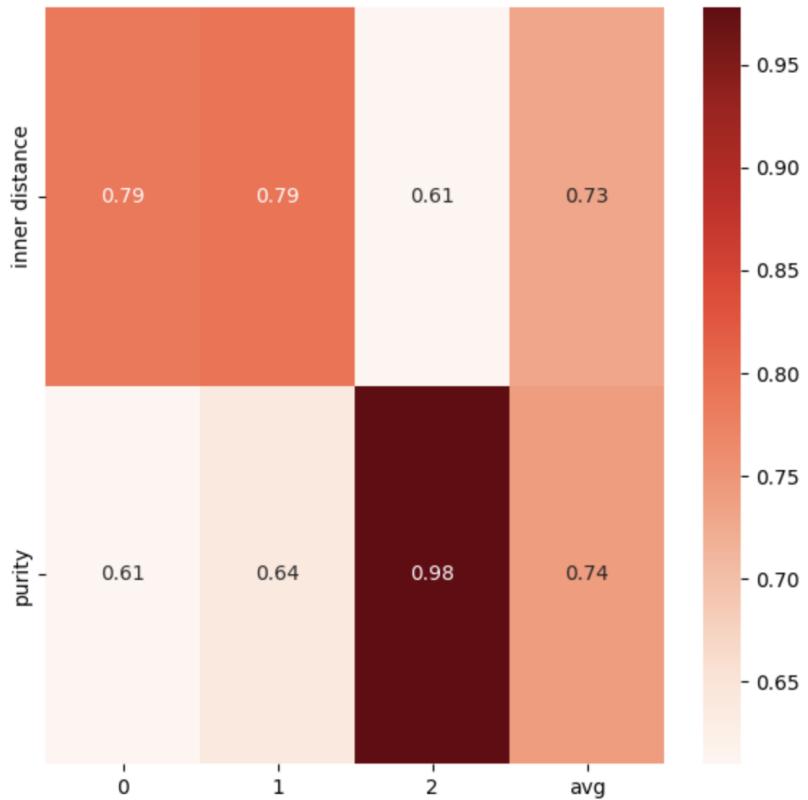


Рисунок 3.15 — Соотнесение внутрикластерных расстояний с чистотой исходных кластеров, отсортированных по укрупнённой шкале Кантрила)

# ЗАКЛЮЧЕНИЕ

В рамках лабораторной работы было проведено изучение эволюционных алгоритмов на примере решения задачи кластеризации.

1. Проведён кластерный анализ данных с использованием агломеративной кластеризации и алгоритма HDBSCAN с варьированием различных значений гиперпараметров и типов расстояний.
2. Проведена оценка работы алгоритмов с использованием внешних и внутренних мер оценки качества. Определено оптимальное количество кластеров и их структура.

Максимальное значение метрики ARI (0.37) было достигнуто при использовании агломеративной кластеризации векторов, полученных после снижения размерности исходных данных с помощью алгоритма UMAP. В алгоритме использовалось манхэттенское расстояние и попарное среднее расстояние между кластерами. В результате было получено 4 кластера, что ближе к числу классов в укрупнённой шкале Кантрила (3 класса), чем к стандартной (10 классов).