Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Системы искусственного интеллекта»

Лабораторная работа №5

«Алгоритмы кластеризации данных»

Выполнил:

студент группы ИВТАПбд-41

Сокольский Р.С

Проверил:

преподаватель кафедры «ВТ»

Святов К.В.

Ульяновск, 2025

Оглавление

[1. Постановка задачи 3](#_Toc210580418)

[2. Теоретические данные 4](#_Toc210580419)

[3. Описание набора данных 6](#_Toc210580420)

[4. Реализация 8](#_Toc210580421)

[5. Тестирование 13](#_Toc210580422)

[6. Выводы 14](#_Toc210580423)

# **1. Постановка задачи**

Перед выполнением лабораторной работы необходимо загрузить набор данных в соответствии с вариантом на диск.

1. Произвести масштабирование признаков (scaling).
2. С использованием библиотеки scikit-learn написать программу с использованием алгоритмов кластеризации данных, позволяющую разделить исходную выборку на классы, соответствующие предложенной вариантом задаче (http://scikit-learn.org/stable/modules/clustering.html).
3. Провести эксперименты и определить наилучший алгоритм кластеризации, параметры алгоритма. Необходимо использовать не менее 3-х алгоритмов. Данные экспериментов необходимо представить в отчете (графики, ход проведения эксперимента, выводы).

Вариант 22: Folio

# **2. Теоретические данные**

*Кластеризация* — представляет собой метод группировки объектов, при котором элементы, имеющие схожие характеристики, объединяются в единые кластеры. Данные представляют собой изображения листьев растений из набора Folio Leaf Dataset. Для каждого изображения извлекались цветовые признаки (средние значения и стандартные отклонения RGB-каналов, яркость, цветовые соотношения). После извлечения признаков производилось масштабирование данных с помощью StandardScaler, что критически важно для алгоритмов, основанных на расстояниях. Такого рода предварительная обработка крайне важна для алгоритмов, основанных на расстоянии, поскольку она приводит все признаки к единому масштабу и позволяет минимизировать влияние дисбаланса в данных.

**KMeans** делит данные на заданное число кластеров K, распределяя объекты так, чтобы минимизировать внутрикластерное различие. Изначально случайным образом задаются центры кластеров, после чего происходит итеративное перераспределение объектов: каждому объекту присваивается тот кластер, центр которого находится ближе всего, а затем вычисляются новые центры кластеров. Процесс повторяется до сходимости, когда центры перестают существенно изменяться. В представленных экспериментах число кластеров варьируется от 2 до 14, а для оценки качества кластеризации применяется коэффициент силуэта. Это позволяет выбрать наилучшую конфигурацию, при которой группы объектов являются компактными внутри себя и хорошо разделёнными между собой.

**Агломеративная кластеризация** (один из методов иерархической кластеризации) начинается с рассмотрения каждого объекта как отдельного кластера и последующего последовательного объединения ближайших пар кластеров до достижения заданного количества групп. Такой подход не требует начальной инициализации центров и позволяет сформировать дендрограмму, отображающую взаимосвязи между объектами на разных уровнях детализации. Как и в случае с KMeans, оптимальное число кластеров определяется путем последовательного перебора параметра k (от 2 до 9) с выбором того варианта, при котором значение коэффициента силуэта максимально.

**DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) отличается от предыдущих тем, что не требует предварительного задания количества кластеров. Он основывается на определении плотных участков в пространстве данных. Алгоритм использует два критических параметра:

* *eps* – радиус окрестности, в пределах которого рассматриваются соседние объекты,
* *min\_samples* – минимальное число точек, необходимых для образования ядра кластера.

DBSCAN объединяет в один кластер те объекты, которые находятся в областях с высокой плотностью, а объекты вне таких областей рассматриваются как шум. В коде проводится серия экспериментов с различными значениями eps (0.5, 1.0, 1.5, 2.0, 2.5) и min\_samples (3, 5, 7, 10), после чего выбирается наилучший вариант на основании коэффициента силуэта — если алгоритм успешно находит несколько кластеров.

**GMM** (Gaussian Mixture Models) — это вероятностный подход к кластеризации, который предполагает, что все точки данных были сгенерированы смесью конечного числа гауссовых распределений с неизвестными параметрами. В отличие от "жёстких" методов кластеризации, таких как K-Means, GMM осуществляет "мягкое" распределение, где каждая точка может принадлежать нескольким кластерам с определённой вероятностью. Алгоритм использует EM-алгоритм (Expectation-Maximization) для оценки параметров смеси: на E-шаге вычисляются вероятности принадлежности точек к каждому компоненту смеси, а на M-шаге пересчитываются параметры распределений (средние значения, ковариационные матрицы и веса компонентов) с учётом полученных вероятностей. В проведённых экспериментах число компонентов смеси варьировалось от 2 до 14, а оптимальное значение выбиралось на основе коэффициента силуэта. Данный подход особенно эффективен для данных со сложной структурой, где кластеры могут иметь различную форму и ориентацию в пространстве признаков.

# **3. Описание набора данных**

Заданный набор данных представляет собой коллекцию изображений листьев растений из базы данных Folio Leaf Dataset. Каждое изображение соответствует определённому виду растения и хранится в отдельной папке, названной по наименованию вида. Для анализа из каждого изображения извлекаются цветовые и текстурные признаки, включая средние значения RGB-каналов, стандартные отклонения цветовых компонентов, общую яркость и цветовые соотношения. Все числовые признаки объединяются в единый вектор признаков для каждого листа, что позволяет охватить максимальное количество визуальных характеристик для последующего анализа и кластеризации. Исходные метки классов (виды растений) сохраняются для последующей оценки качества кластеризации.

# **4. Реализация**

**Чтение данных**

Программа начинается с обработки изображений листьев из набора данных Folio. Для каждого файла изображения в форматах PNG, JPG или JPEG из соответствующих директорий извлекаются цветовые и текстурные характеристики. Полученные числовые признаки, включая средние значения и стандартные отклонения RGB-каналов, яркость и цветовые соотношения, сохраняются в объекте DataFrame. Это обеспечивает структурированное хранение признаков для последующей обработки и анализа методами кластеризации.

**Листинг 1.** Чтение данных БД

|  |
| --- |
| **def extract\_features(image\_path): # Извлечение признаков из фотки (средние RGB-каналы и их стандартные отклонения (типа текстура))**  **try:**  **with Image.open(image\_path) as img:**  **# Конвертация в RGB (на всякий)**  **img = img.convert('RGB')**    **# Всё в массив**  **img\_array = np.array(img)**    **# Извлечение признаки цвета**  **r\_mean = np.mean(img\_array[:, :, 0])**  **g\_mean = np.mean(img\_array[:, :, 1])**  **b\_mean = np.mean(img\_array[:, :, 2])**    **r\_std = np.std(img\_array[:, :, 0])**  **g\_std = np.std(img\_array[:, :, 1])**  **b\_std = np.std(img\_array[:, :, 2])**    **# Допом: соотношения цветов**  **brightness = (r\_mean + g\_mean + b\_mean) / 3**  **r\_g\_ratio = r\_mean / (g\_mean + 1e-5) # +1e-5 чтобы избежать деления на 0**  **g\_b\_ratio = g\_mean / (b\_mean + 1e-5)**    **return [r\_mean, g\_mean, b\_mean, r\_std, g\_std, b\_std,**  **brightness, r\_g\_ratio, g\_b\_ratio]**    **except Exception as e:**  **print(f"Ошибка при обработке {image\_path}: {e}")**  **return None**  **def load\_folio\_dataset(data\_path): # Загрузка Folio и извлечение признаков**  **features = [] # Вектора признаков**  **filenames = [] # Имена файлов**  **class\_labels = [] # Метки классов (т.е. названия папок)**    **# Прохожу по всем папкам-классам**  **for class\_name in os.listdir(data\_path):**  **class\_path = os.path.join(data\_path, class\_name)**    **if os.path.isdir(class\_path): # Проверка: папка?**  **for img\_file in os.listdir(class\_path): # Перебор файлов в папке**  **if img\_file.lower().endswith(('.png', '.jpg', '.jpeg')): # Проверка расширений в конце**  **img\_path = os.path.join(class\_path, img\_file)**  **feature\_vector = extract\_features(img\_path) # Извлечение признаков**    **if feature\_vector is not None: # И вводим, если признаки есть (а как иначе)**  **features.append(feature\_vector)**  **filenames.append(img\_file)**  **class\_labels.append(class\_name)**    **# Создание DF**  **feature\_names = ['r\_mean', 'g\_mean', 'b\_mean', 'r\_std', 'g\_std', 'b\_std',**  **'brightness', 'r\_g\_ratio', 'g\_b\_ratio']**    **df = pd.DataFrame(features, columns=feature\_names)**  **df['filename'] = filenames**  **df['class'] = class\_labels**    **return df**  **data\_path = 'C:/Users/Roman/Folio'**  **folio\_df = load\_folio\_dataset(data\_path)** |

**Масштабирование признаков**

Прежде чем перейти к кластеризации, извлечённые признаки нормализуются с использованием StandardScaler. Данный метод центрирует данные таким образом, чтобы каждый признак имел нулевое среднее значение и единичную дисперсию. Такой подход гарантирует, что все признаки приведены к единому масштабу, что критически важно для алгоритмов кластеризации, основанных на вычислении расстояний между объектами, поскольку предотвращает доминирование признаков с большими числовыми диапазонами над признаками с меньшими значениями:

**Листинг 2.** Масштабирование признаков

|  |  |
| --- | --- |
| |  | | --- | | **# Отделение признаков от меток и имен файлов**  **X = folio\_df.drop(['filename', 'class'], axis=1) # Столбцы с именем и папкой до свидания, axis=1 => УДАЛЯЕМ СТОЛБЦЫ, НЕ СТРОКИ**  **true\_labels = folio\_df['class'] # Сохранение меток класса (то есть названий)**  **filenames = folio\_df['filename'] # Сохранение filename для отслеживания, к какому изображению результаты относятся**  **# Масштабирование признаков**  **scaler = StandardScaler()**  **X\_scaled = scaler.fit\_transform(X)**  **print("Данные после масштабирования:")**  **print(f"Размерность: {X\_scaled.shape}")**  **print(f"Средние значения: {np.mean(X\_scaled, axis=0)}")**  **print(f"Стандартные отклонения: {np.std(X\_scaled, axis=0)}")** | |
|  |

**Визуализация данных с помощью PCA**

Для визуального анализа структуры данных и снижения размерности признакового пространства применяется метод главных компонент (PCA). Алгоритм проецирует исходные 9-мерные данные в двумерное пространство, сохраняя при этом максимально возможную дисперсию. Это позволяет не только визуализировать распределение объектов, но и оценить наличие естественных группировок в данных перед применением алгоритмов кластеризации.

**Листинг 3.** Визуализация данных с помощью PCA

|  |  |
| --- | --- |
| |  | | --- | | **pca = PCA(n\_components=2) # Применение PCA для визуализации в 2D**  **X\_pca = pca.fit\_transform(X\_scaled)**  **plt.figure(figsize=(12, 5))**  **plt.subplot(1, 2, 1) # Визуализация исходных классов (для reference)**  **scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=pd.factorize(true\_labels)[0],**  **cmap='tab20', alpha=0.7)**  **plt.title('Исходные классы (True Labels)')**  **plt.xlabel(f'PC1 ({pca.explained\_variance\_ratio\_[0]:.2%} variance)')**  **plt.ylabel(f'PC2 ({pca.explained\_variance\_ratio\_[1]:.2%} variance)')**  **plt.colorbar(scatter)**  **plt.subplot(1, 2, 2) # Визуализация без цветов (как видит алгоритм кластеризации)**  **plt.scatter(X\_pca[:, 0], X\_pca[:, 1], alpha=0.7)**  **plt.title('Данные для кластеризации (без меток)')**  **plt.xlabel(f'PC1 ({pca.explained\_variance\_ratio\_[0]:.2%} variance)')**  **plt.ylabel(f'PC2 ({pca.explained\_variance\_ratio\_[1]:.2%} variance)')**  **plt.tight\_layout()**  **plt.show()**  **print(f"Объясненная дисперсия PCA: {pca.explained\_variance\_ratio\_.sum():.2%}")**  **# смотрю, из чего состоят главные компоненты (отладка)**  **print("Состав PC1 и PC2:")**  **components\_df = pd.DataFrame(**  **pca.components\_.T, # транспонируется матрица компонент**  **columns=['PC1', 'PC2'],**  **index=X.columns # названия исходных признаков**  **)**  **print(components\_df)** | |

**Кластеризация методом K-Means**

Алгоритм K-Means применяется для разделения данных на заданное количество кластеров. Для определения оптимального числа кластеров используется комбинация метода локтя и анализа silhouette score. Алгоритм итеративно перераспределяет объекты между кластерами, минимизируя внутрикластерные расстояния до центроидов. Качество кластеризации оценивается с помощью silhouette score и Calinski-Harabasz index, что позволяет объективно сравнить различные конфигурации параметров.

**Листинг 4.** Кластеризация методом K-Means

|  |  |
| --- | --- |
| |  | | --- | | **# Подбор оптимального числа кластеров для K-Means**  **inertia = [] # Стоимость кластеризации - чем меньше - тем лучше. (сумма квадратов расстояний от точек до центров их кластеров)**  **silhouette\_scores = [] # Качество кластеризации. больше - лучше. (Насколько хорошо точки внутри кластера похожи друг на друга и не похожи на точки из других кластеров.)**  **k\_range = range(2, 15) # Проба k от 2 до 14 кластеров.**  **for k in k\_range:**  **kmeans = KMeans(n\_clusters=k, random\_state=42, n\_init=10)**  **kmeans.fit(X\_scaled)**  **inertia.append(kmeans.inertia\_)**  **silhouette\_scores.append(silhouette\_score(X\_scaled, kmeans.labels\_))**  **# График локтя и silhouette score**  **plt.figure(figsize=(15, 5))**  **plt.subplot(1, 3, 1)**  **plt.plot(k\_range, inertia, 'bo-') # Ищу точку "излома", где инерция перестёт сильно уменьшаться**  **plt.xlabel('Число кластеров')**  **plt.ylabel('Инерция')**  **plt.title('Метод локтя')**  **plt.subplot(1, 3, 2)**  **plt.plot(k\_range, silhouette\_scores, 'ro-') # Максимальное значение - лучшее качество кластеризации**  **plt.xlabel('Число кластеров')**  **plt.ylabel('Silhouette Score')**  **plt.title('Silhouette Score vs число кластеров')**  **# Выбираю оптимальное k (например, где silhouette score максимален)**  **optimal\_k = k\_range[np.argmax(silhouette\_scores)]**  **print(f"Оптимальное число кластеров по silhouette score: {optimal\_k}")**  **# Кластеризация с оптимальным k**  **kmeans\_optimal = KMeans(n\_clusters=optimal\_k, random\_state=42, n\_init=10)**  **kmeans\_labels = kmeans\_optimal.fit\_predict(X\_scaled)**  **# Визуализация результатов K-Means**  **plt.subplot(1, 3, 3)**  **scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=kmeans\_labels, cmap='tab20', alpha=0.7)**  **plt.title(f'K-Means кластеризация (k={optimal\_k})')**  **plt.xlabel('PC1')**  **plt.ylabel('PC2')**  **plt.colorbar(scatter)**  **plt.tight\_layout()**  **plt.show()**  **# Оценка качества K-Means**  **kmeans\_silhouette = silhouette\_score(X\_scaled, kmeans\_labels)**  **kmeans\_calinski = calinski\_harabasz\_score(X\_scaled, kmeans\_labels) # отношение "междукластерной дисперсии" к "внутрикластерной", кароч, чем больше, тем лучше**  **print(f"K-Means Results (k={optimal\_k}):")**  **print(f"Silhouette Score: {kmeans\_silhouette:.3f}")**  **print(f"Calinski-Harabasz Score: {kmeans\_calinski:.3f}")** | |

**Кластеризация методом DBSCAN**

Алгоритм DBSCAN (Density-Based Spatial Clustering of Applications with Noise) применяется для поиска кластеров произвольной формы на основе плотности распределения точек в пространстве признаков. В отличие от K-Means, данный метод не требует предварительного задания количества кластеров и способен идентифицировать выбросы как шумовые точки. Для настройки алгоритма производится подбор двух ключевых параметров: eps (радиус окрестности) и min\_samples (минимальное количество точек для формирования кластера), что позволяет адаптировать алгоритм к специфике данных.

**Листинг 5.** Кластеризация методом DBSCAN

|  |  |
| --- | --- |
| |  | | --- | | **# Подбор параметров для DBSCAN**  **from sklearn.neighbors import NearestNeighbors**  **# Метод для подбора eps**  **neighbors = NearestNeighbors(n\_neighbors=min(10, len(X\_scaled)-1))**  **neighbors\_fit = neighbors.fit(X\_scaled)**  **distances, indices = neighbors\_fit.kneighbors(X\_scaled)**  **distances = np.sort(distances[:, -1], axis=0)**  **plt.figure(figsize=(15, 5))**  **plt.subplot(1, 3, 1)**  **plt.plot(distances)**  **plt.xlabel('Data Points')**  **plt.ylabel('Epsilon')**  **plt.title('K-Distance граф for DBSCAN')**  **# Эксперименты с разными параметрами**  **eps\_values = [0.5, 1.0, 1.5, 2.0, 2.5]**  **min\_samples\_values = [3, 5, 7, 10]**  **dbscan\_results = []**  **for eps in eps\_values:**  **for min\_samples in min\_samples\_values:**  **dbscan = DBSCAN(eps=eps, min\_samples=min\_samples)**  **dbscan\_labels = dbscan.fit\_predict(X\_scaled)**    **# Пропускаем случаи, когда все точки в одном кластере или много шума**  **n\_clusters = len(set(dbscan\_labels)) - (1 if -1 in dbscan\_labels else 0)**  **n\_noise = list(dbscan\_labels).count(-1)**    **if n\_clusters > 1 and n\_clusters < len(X\_scaled) // 2:**  **silhouette = silhouette\_score(X\_scaled, dbscan\_labels)**  **dbscan\_results.append({**  **'eps': eps,**  **'min\_samples': min\_samples,**  **'n\_clusters': n\_clusters,**  **'n\_noise': n\_noise,**  **'silhouette': silhouette**  **})**  **# DataFrame с результатами**  **dbscan\_df = pd.DataFrame(dbscan\_results)**  **if not dbscan\_df.empty:**  **best\_dbscan = dbscan\_df.loc[dbscan\_df['silhouette'].idxmax()]**    **print("Лучшие параметры DBSCAN:")**  **print(best\_dbscan)**    **# Применение DBSCAN с лучшими параметрами**  **dbscan\_optimal = DBSCAN(eps=best\_dbscan['eps'],**  **min\_samples=int(best\_dbscan['min\_samples']))**  **dbscan\_labels = dbscan\_optimal.fit\_predict(X\_scaled)**    **# Визуализация DBSCAN**  **plt.subplot(1, 3, 2)**  **scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=dbscan\_labels, cmap='tab20', alpha=0.7)**  **plt.title(f'DBSCAN (eps={best\_dbscan["eps"]}, min\_samples={best\_dbscan["min\_samples"]})')**  **plt.xlabel('PC1')**  **plt.ylabel('PC2')**  **plt.colorbar(scatter)**    **# Оценка качества DBSCAN**  **dbscan\_silhouette = best\_dbscan['silhouette']**  **if len(set(dbscan\_labels)) > 1: # Calinski-Harabasz требует хотя бы 2 кластера**  **dbscan\_calinski = calinski\_harabasz\_score(X\_scaled, dbscan\_labels)**  **else:**  **dbscan\_calinski = 0**    **print(f"\nРезультаты DBSCAN:")**  **print(f"Число кластеров: {best\_dbscan['n\_clusters']}")**  **print(f"Число точек шума: {best\_dbscan['n\_noise']}")**  **print(f"Silhouette Score: {dbscan\_silhouette:.3f}")**  **print(f"Calinski-Harabasz Score: {dbscan\_calinski:.3f}")**  **else:**  **print("DBSCAN не смог найти хорошие кластеры с данными параметрами")**  **dbscan\_labels = np.zeros(len(X\_scaled)) # Заглушка для продолжения работы**  **dbscan\_silhouette = 0**  **dbscan\_calinski = 0**  **plt.tight\_layout()**  **plt.show()** | |

Для интерпретации результатов кластеризации DBSCAN проводится сравнительный анализ средних значений признаков в полученных кластерах и шумовых точках. Это позволяет определить, по каким визуальным характеристикам алгоритм разделил листья на различные группы. Анализ focuses на ключевых признаках, включая цветовые компоненты, яркость и текстуру, что дает понимание семантики выделенных кластеров и природы точек, идентифицированных как шум. В данном случае, (забегая вперёд), это выполнено в качестве выяснения причин низкого результата данного алгоритма.

**Листинг 6.** Анализ характеристик кластеров DBSCAN

|  |  |
| --- | --- |
| |  | | --- | | **# Анализ что такое Кластер 0 и Кластер 1**  **dbscan\_cluster\_0 = X\_scaled[dbscan\_labels == 0] # все листья кластера 0**  **dbscan\_cluster\_1 = X\_scaled[dbscan\_labels == 1] # все листья кластера 1**  **dbscan\_noise = X\_scaled[dbscan\_labels == -1] # шум**  **# Сравним средние значения признаков**  **print("СРЕДНИЕ ЗНАЧЕНИЯ ПРИЗНАКОВ ПО КЛАСТЕРАМ:")**  **print("=" \* 50)**  **features = ['r\_mean', 'g\_mean', 'b\_mean', 'brightness', 'r\_std']**  **for feature in features:**  **# Находим индекс признака в наших данных**  **feature\_idx = list(X.columns).index(feature)**    **mean\_0 = np.mean(dbscan\_cluster\_0[:, feature\_idx])**  **mean\_1 = np.mean(dbscan\_cluster\_1[:, feature\_idx])**  **mean\_noise = np.mean(dbscan\_noise[:, feature\_idx])**    **print(f"{feature:12} | Кластер 0: {mean\_0:6.2f} | Кластер 1: {mean\_1:6.2f} | Шум: {mean\_noise:6.2f}")** | |

**Кластеризация методом Agglomerative Clustering**

Алгоритм Agglomerative Clustering реализует иерархический подход "снизу вверх", начиная с рассмотрения каждого объекта как отдельного кластера и последовательно объединяя наиболее близкие пары кластеров. Для определения оптимальной конфигурации тестируются четыре метода связи (ward, complete, average, single) с различным количеством кластеров в диапазоне от 2 до 9. Качество каждой конфигурации оценивается с помощью silhouette score, что позволяет выбрать наиболее эффективную комбинацию параметров для финальной кластеризации.

**Листинг 7.** Кластеризация методом Agglomerative Clustering

|  |  |
| --- | --- |
| |  | | --- | | **# Эксперименты с Agglomerative Clustering**  **linkage\_methods = ['ward', 'complete', 'average', 'single']**  **n\_clusters\_range = range(2, 10)**  **agg\_results = []**  **plt.figure(figsize=(15, 5))**  **for i, linkage in enumerate(linkage\_methods, 1):**  **silhouette\_scores\_agg = []**    **for n\_clusters in n\_clusters\_range:**  **agg = AgglomerativeClustering(n\_clusters=n\_clusters, linkage=linkage)**  **agg\_labels = agg.fit\_predict(X\_scaled)**  **silhouette = silhouette\_score(X\_scaled, agg\_labels)**  **silhouette\_scores\_agg.append(silhouette)**    **# Находим оптимальное число кластеров для данного метода связи**  **optimal\_n\_agg = n\_clusters\_range[np.argmax(silhouette\_scores\_agg)]**  **best\_silhouette = max(silhouette\_scores\_agg)**    **agg\_results.append({**  **'linkage': linkage,**  **'n\_clusters': optimal\_n\_agg,**  **'silhouette': best\_silhouette**  **})**    **# Визуализация silhouette scores для каждого метода**  **plt.subplot(1, 4, i)**  **plt.plot(n\_clusters\_range, silhouette\_scores\_agg, 'go-')**  **plt.xlabel('Число кластеров')**  **plt.ylabel('Silhouette Score')**  **plt.title(f'Linkage: {linkage}')**  **plt.tight\_layout()**  **plt.show()**  **# Выбираем лучший метод**  **agg\_df = pd.DataFrame(agg\_results)**  **best\_agg = agg\_df.loc[agg\_df['silhouette'].idxmax()]**  **print("Лучшие параметры для Agglomerative Clustering:")**  **print(best\_agg)**  **# Применяем иерархическую кластеризацию с лучшими параметрами**  **agg\_optimal = AgglomerativeClustering(**  **n\_clusters=int(best\_agg['n\_clusters']),**  **linkage=best\_agg['linkage']**  **)**  **agg\_labels = agg\_optimal.fit\_predict(X\_scaled)**  **# Визуализация дендрограммы (для методов кроме 'ward')**  **if best\_agg['linkage'] != 'ward':**  **from scipy.cluster.hierarchy import dendrogram, linkage**    **plt.figure(figsize=(10, 6))**  **# Берем подвыборку для лучшей визуализации**  **if len(X\_scaled) > 100:**  **indices = np.random.choice(len(X\_scaled), 100, replace=False)**  **Z = linkage(X\_scaled[indices], method=best\_agg['linkage'])**  **else:**  **Z = linkage(X\_scaled, method=best\_agg['linkage'])**    **dendrogram(Z)**  **plt.title(f'Dendrogram (Linkage: {best\_agg["linkage"]})')**  **plt.xlabel('Sample index')**  **plt.ylabel('Distance')**  **plt.show()**  **# Визуализация кластеров**  **plt.figure(figsize=(8, 6))**  **scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=agg\_labels, cmap='tab20', alpha=0.7)**  **plt.title(f'Agglomerative Clustering ({best\_agg["linkage"]}, k={best\_agg["n\_clusters"]})')**  **plt.xlabel('PC1')**  **plt.ylabel('PC2')**  **plt.colorbar(scatter)**  **plt.show()**  **# Оценка качества Agglomerative Clustering**  **agg\_silhouette = best\_agg['silhouette']**  **agg\_calinski = calinski\_harabasz\_score(X\_scaled, agg\_labels)**  **print(f"\nAgglomerative Clustering Results:")**  **print(f"Silhouette Score: {agg\_silhouette:.3f}")**  **print(f"Calinski-Harabasz Score: {agg\_calinski:.3f}")** | |

**Кластеризация методом Gaussian Mixture Models**

Алгоритм Gaussian Mixture Models (GMM) применяет вероятностный подход к кластеризации, предполагая, что данные были сгенерированы смесью нескольких гауссовых распределений. В отличие от "жестких" методов кластеризации, GMM назначает каждому объекту вероятности принадлежности к различным кластерам, что позволяет учитывать нечеткость границ между группами. Для определения оптимального числа компонентов смеси проводится поиск в диапазоне от 2 до 14 с выбором конфигурации, максимизирующей silhouette score.

**Листинг 8.** Кластеризация методом Gaussian Mixture Models

|  |  |
| --- | --- |
| |  | | --- | | **# Алгоритм 4: Gaussian Mixture Models**  **from sklearn.mixture import GaussianMixture**  **print("GAUSSIAN MIXTURE CLUSTERING")**  **# Подбор оптимального числа компонент для GMM**  **gmm\_silhouette\_scores = []**  **n\_components\_range = range(2, 15)**  **for n\_components in n\_components\_range:**  **gmm = GaussianMixture(n\_components=n\_components, random\_state=42)**  **gmm\_labels = gmm.fit\_predict(X\_scaled)**    **if len(set(gmm\_labels)) > 1:**  **silhouette = silhouette\_score(X\_scaled, gmm\_labels)**  **gmm\_silhouette\_scores.append(silhouette)**  **else:**  **gmm\_silhouette\_scores.append(0)**  **# График подбора числа компонент**  **plt.figure(figsize=(10, 4))**  **plt.subplot(1, 2, 1)**  **plt.plot(n\_components\_range, gmm\_silhouette\_scores, 'go-')**  **plt.xlabel('Number of Components')**  **plt.ylabel('Silhouette Score')**  **plt.title('GMM: Silhouette Score vs Number of Components')**  **plt.grid(True, alpha=0.3)**  **# Находим оптимальное число компонент**  **optimal\_n\_components = n\_components\_range[np.argmax(gmm\_silhouette\_scores)]**  **print(f"Оптимальное число компонент для GMM: {optimal\_n\_components}")**  **# Обучаем GMM с оптимальными параметрами**  **gmm\_optimal = GaussianMixture(n\_components=optimal\_n\_components, random\_state=42)**  **gmm\_labels = gmm\_optimal.fit\_predict(X\_scaled)**  **# Визуализация результатов GMM**  **plt.subplot(1, 2, 2)**  **scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=gmm\_labels, cmap='tab20', alpha=0.7)**  **plt.title(f'GMM Clustering (n\_components={optimal\_n\_components})')**  **plt.xlabel('PC1')**  **plt.ylabel('PC2')**  **plt.colorbar(scatter)**  **plt.tight\_layout()**  **plt.show()**  **# Оценка качества GMM**  **gmm\_silhouette = silhouette\_score(X\_scaled, gmm\_labels)**  **gmm\_calinski = calinski\_harabasz\_score(X\_scaled, gmm\_labels)**  **print(f"GMM Results (n\_components={optimal\_n\_components}):")**  **print(f"Silhouette Score: {gmm\_silhouette:.3f}")**  **print(f"Calinski-Harabasz Score: {gmm\_calinski:.3f}")**  **# Анализ распределения кластеров**  **gmm\_cluster\_counts = pd.Series(gmm\_labels).value\_counts().sort\_index()**  **print("Распределение по кластерам GMM:")**  **for cluster, count in gmm\_cluster\_counts.items():**  **print(f" Кластер {cluster}: {count} листьев ({count/len(gmm\_labels)\*100:.1f}%)")** | |

**Поиск оптимальных параметров**

Для объективной оценки эффективности различных методов кластеризации проводится сравнительный анализ на основе метрик качества - silhouette score и Calinski-Harabasz index. Создается сводная таблица, содержащая результаты работы всех алгоритмов с их оптимальными параметрами, что позволяет провести прямое сравнение и выявить наиболее подходящий метод для данного набора данных. Визуализация включает столбчатые диаграммы для наглядного сравнения метрик и параллельное отображение результатов кластеризации всех алгоритмов.

**Листинг 9.** Сравнительный анализ алгоритмов кластеризации

|  |  |
| --- | --- |
| |  | | --- | | **# Сводная таблица результатов для 4 алгоритмов**  **results\_comparison = pd.DataFrame({**  **'Algorithm': ['K-Means', 'DBSCAN', 'Agglomerative', 'Gaussian Mixture'],**  **'Silhouette Score': [kmeans\_silhouette, dbscan\_silhouette, agg\_silhouette, gmm\_silhouette],**  **'Calinski-Harabasz Score': [kmeans\_calinski, dbscan\_calinski, agg\_calinski, gmm\_calinski],**  **'Parameters': [**  **f'k={optimal\_k}',**  **f'eps={best\_dbscan["eps"] if not dbscan\_df.empty else "N/A"}, min\_samples={best\_dbscan["min\_samples"] if not dbscan\_df.empty else "N/A"}',**  **f'linkage={best\_agg["linkage"]}, k={best\_agg["n\_clusters"]}',**  **f'n\_components={optimal\_n\_components}'**  **]**  **})**  **print("СРАВНЕНИЕ 4 АЛГОРИТМОВ:")**  **print("=" \* 60)**  **print(results\_comparison)**  **# Визуализация сравнения**  **fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))**  **# График Silhouette Score**  **colors = ['skyblue', 'lightcoral', 'lightgreen', 'gold']**  **bars1 = ax1.bar(results\_comparison['Algorithm'], results\_comparison['Silhouette Score'],**  **color=colors)**  **ax1.set\_title('Сравнение Silhouette Score', fontsize=14, fontweight='bold')**  **ax1.set\_ylabel('Silhouette Score')**  **ax1.grid(axis='y', alpha=0.3)**  **# Добавляем значения на столбцы**  **for bar, score in zip(bars1, results\_comparison['Silhouette Score']):**  **height = bar.get\_height()**  **ax1.text(bar.get\_x() + bar.get\_width()/2., height + 0.01,**  **f'{score:.3f}', ha='center', va='bottom', fontweight='bold')**  **# График Calinski-Harabasz Score**  **bars2 = ax2.bar(results\_comparison['Algorithm'], results\_comparison['Calinski-Harabasz Score'],**  **color=colors)**  **ax2.set\_title('Сравнение Calinski-Harabasz Score', fontsize=14, fontweight='bold')**  **ax2.set\_ylabel('Calinski-Harabasz Score')**  **ax2.grid(axis='y', alpha=0.3)**  **# Добавляем значения на столбцы**  **for bar, score in zip(bars2, results\_comparison['Calinski-Harabasz Score']):**  **height = bar.get\_height()**  **ax2.text(bar.get\_x() + bar.get\_width()/2., height + 5,**  **f'{score:.1f}', ha='center', va='bottom', fontweight='bold')**  **plt.tight\_layout()**  **plt.show()**  **# Визуализация всех результатов кластеризации (4 алгоритма + исходные классы)**  **plt.figure(figsize=(20, 4))**  **algorithms = [**  **('K-Means', kmeans\_labels),**  **('DBSCAN', dbscan\_labels),**  **('Agglomerative', agg\_labels),**  **('Gaussian Mixture', gmm\_labels)**  **]**  **for i, (name, labels) in enumerate(algorithms, 1):**  **plt.subplot(1, 5, i)**  **scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=labels, cmap='tab20', alpha=0.7)**  **plt.title(f'{name}\nSilhouette: {silhouette\_score(X\_scaled, labels):.3f}')**  **plt.xlabel('PC1')**  **plt.ylabel('PC2')**  **plt.colorbar(scatter)**  **# Исходные классы для сравнения**  **plt.subplot(1, 5, 5)**  **scatter = plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=pd.factorize(true\_labels)[0],**  **cmap='tab20', alpha=0.7)**  **plt.title('Исходные классы\n(32 вида растений)')**  **plt.xlabel('PC1')**  **plt.ylabel('PC2')**  **plt.colorbar(scatter)**  **plt.tight\_layout()**  **plt.show()**  **# Определение лучшего алгоритма**  **best\_algorithm\_idx = np.argmax([kmeans\_silhouette, dbscan\_silhouette, agg\_silhouette, gmm\_silhouette])**  **best\_algorithm\_name = ['K-Means', 'DBSCAN', 'Agglomerative', 'Gaussian Mixture'][best\_algorithm\_idx]**  **best\_silhouette = [kmeans\_silhouette, dbscan\_silhouette, agg\_silhouette, gmm\_silhouette][best\_algorithm\_idx]**  **print(f"ЛУЧШИЙ АЛГОРИТМ: {best\_algorithm\_name}")**  **print(f"Лучший Silhouette Score: {best\_silhouette:.3f}")** | |

# **5. Тестирование**

**1. K-Means**

Для алгоритма K-Means проводился поиск оптимального числа кластеров в диапазоне от 2 до 14. Метод локтя показал плавное уменьшение инерции без явного излома, что свидетельствует об отсутствии четкой кластерной структуры. Анализ silhouette score выявил максимальное значение (0.404) при k=2, что указывает на оптимальность разделения данных на две группы. Визуализация результатов подтвердила образование двух основных кластеров с умеренным качеством разделения.

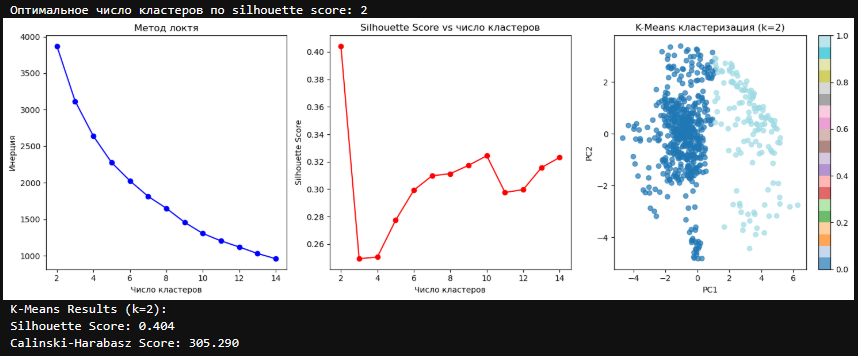


Рис. 1 – Вывод в консоль

K-Means показал наилучшее качество среди всех алгоритмов с silhouette score 0.404, оптимально разделив данные на 2 кластера.

**2. DBSCAN**

Алгоритм DBSCAN тестировался с различными комбинациями параметров eps (0.5-2.5) и min\_samples (3-10). Наилучшие результаты достигнуты при eps=1.0 и min\_samples=10, однако качество кластеризации оказалось низким (silhouette score 0.220). Алгоритм идентифицировал 2 кластера и 167 шумовых точек (26% данных), что указывает на неприменимость плотностного подхода к данным с равномерным распределением.

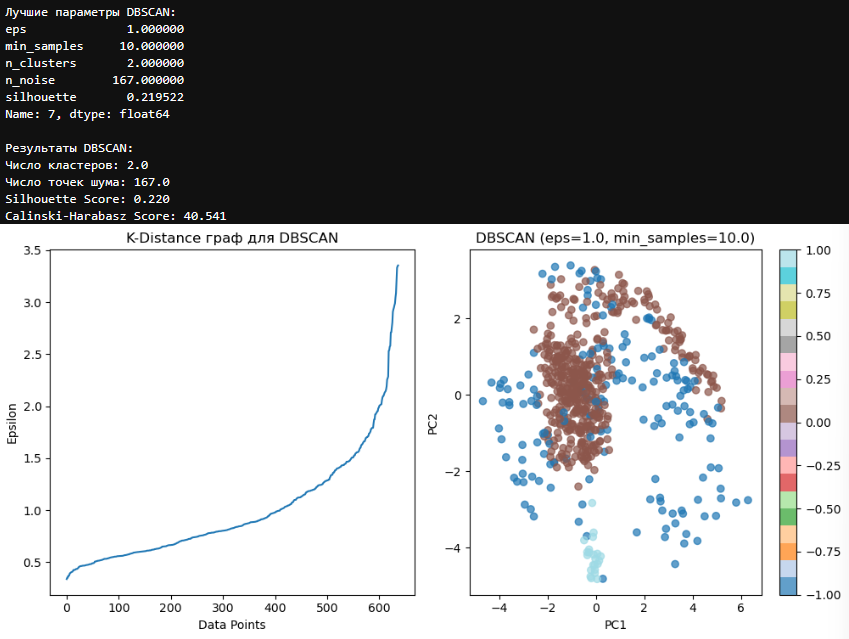


Рис. 2 – Вывод в консоль

DBSCAN показал наихудшие результаты среди всех алгоритмов, что свидетельствует об отсутствии четких плотностных кластеров в данных.

Для интерпретации результатов DBSCAN проведен анализ средних значений признаков в выделенных кластерах и шумовых точках. Полученные данные свидетельствуют о четком разделении характеристик:

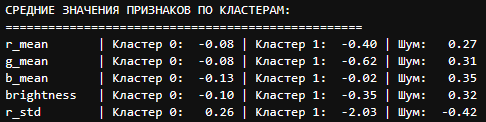


Рис. 3 – Вывод в консоль средних значений признаков по кластерам DBSCAN

**Кластер 0** демонстрирует умеренные отрицательные значения по цветовым компонентам (r\_mean: -0.08, g\_mean: -0.08, b\_mean: -0.13) и яркости (-0.10), при этом имеет положительное значение текстуры (r\_std: 0.26). Это указывает на группу относительно темных листьев со средней текстурой.

**Кластер 1** характеризуется значительно более низкими значениями по красному и зеленому каналам (-0.40 и -0.62 соответственно) и яркости (-0.35), при экстремально низком значении текстуры (r\_std: -2.03). Данный кластер представляет собой группу очень темных листьев с минимальной текстурной вариативностью.

**Шумовые точки** показывают положительные значения по всем цветовым характеристикам (r\_mean: 0.27, g\_mean: 0.31, b\_mean: 0.35, brightness: 0.32) и умеренно отрицательное значение текстуры (-0.42). Это свидетельствует о том, что шум составляют преимущественно светлые листья со слабой текстурой, которые не образуют плотных скоплений в пространстве признаков.

**3. Agglomerative Clustering**

Иерархическая кластеризация тестировалась с четырьмя методами связи (ward, complete, average, single) и числом кластеров от 2 до 9. Наилучшие результаты показал метод Ward с k=2 (silhouette score 0.379). Визуализация демонстрирует схожее с K-Means разделение на две основные группы, что подтверждает наличие двух доминирующих паттернов в данных.

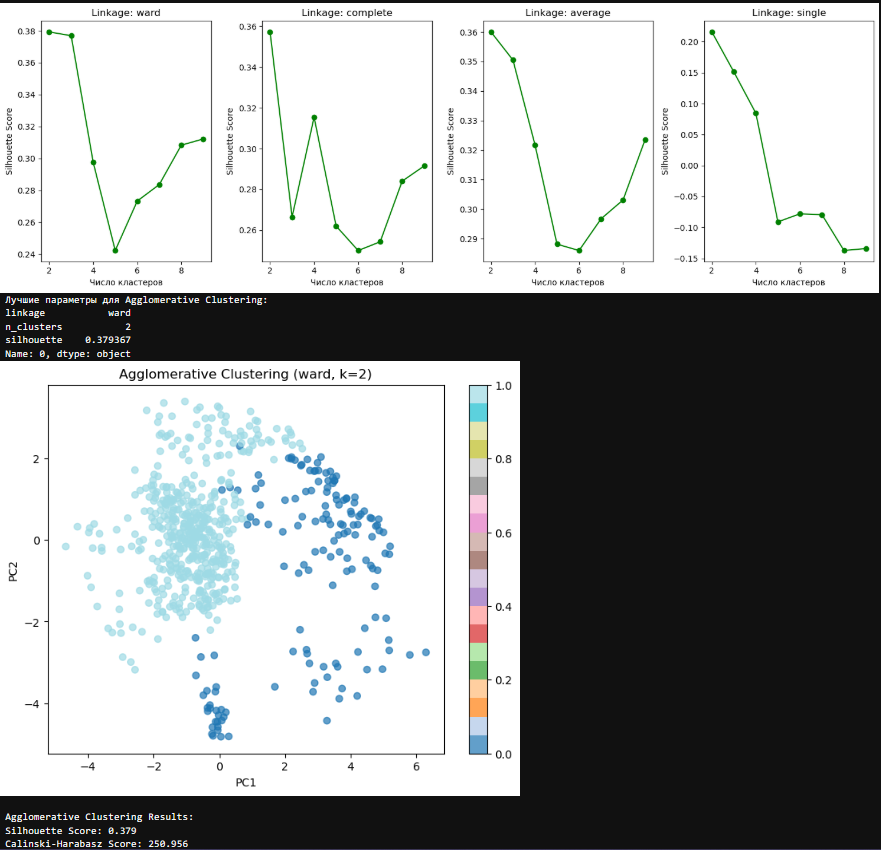


Рис. 4 – Вывод в консоль

Agglomerative Clustering подтвердил результаты K-Means, показав схожее качество и выделив те же две основные группы в данных.

**4. Gaussian Mixture Models**

GMM тестировался с числом компонентов от 2 до 14. Наилучшее качество достигнуто при 2 компонентах (silhouette score 0.380), что соответствует результатам других алгоритмов. Вероятностный подход GMM позволил получить "мягкое" разделение, где некоторые точки могут принадлежать нескольким кластерам с различной вероятностью.

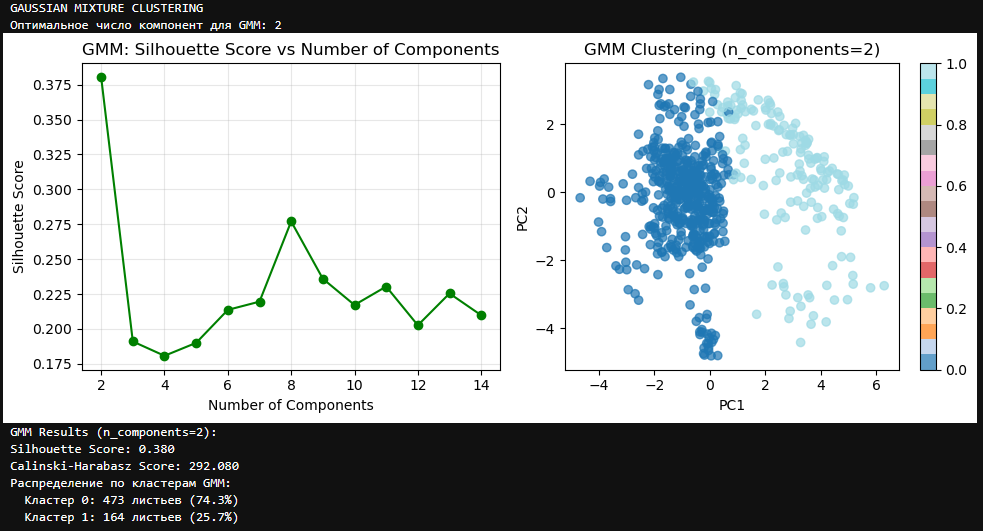


Рис. 5 – Вывод в консоль

GMM показал результаты, сопоставимые с K-Means и Agglomerative Clustering, подтвердив наличие двух основных групп в данных.

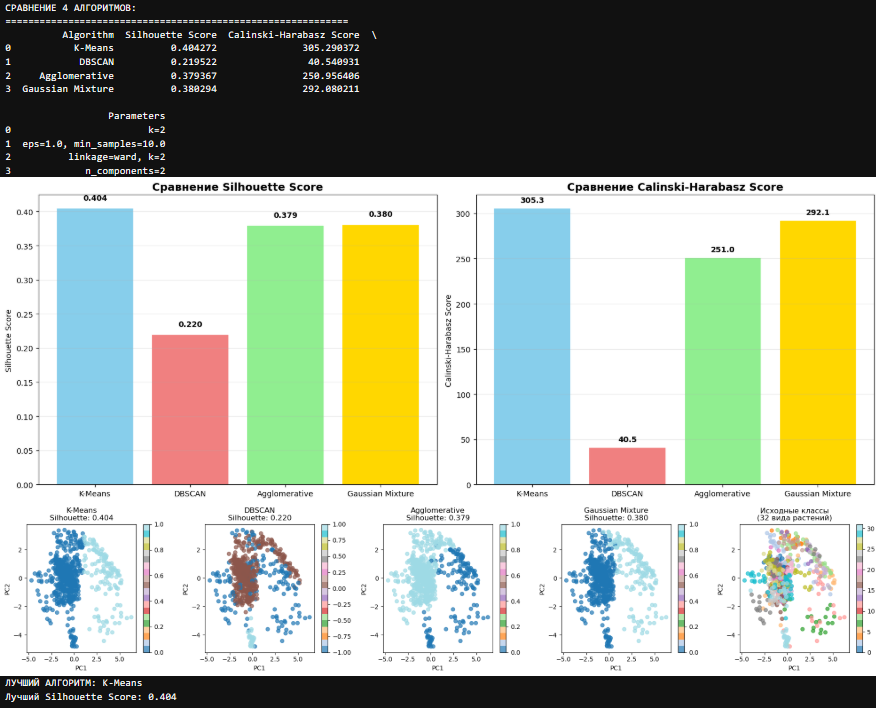


Рис. 6 – Вывод сводных данных в консоль

Вывод о сводных данных в пункте 6.

# **6.** **Выводы**

В ходе выполнения лабораторной работы был проведен комплексный анализ алгоритмов кластеризации на основе набора данных Folio Leaf Dataset, содержащего изображения листьев растений. Экспериментальным путем установлено, что простых цветовых признаков недостаточно для эффективного разделения 32 видов растений на отдельные кластеры.

Наилучшие результаты показал алгоритм K-Means с silhouette score 0.404 при разделении на 2 кластера. Близкие результаты продемонстрировали Gaussian Mixture Models (0.380) и Agglomerative Clustering (0.379), что свидетельствует о наличии в данных двух основных групп, вероятно соответствующих макро-категориям листьев по визуальным характеристикам.

Алгоритм DBSCAN показал наихудшее качество (silhouette score 0.220), идентифицировав 26% данных как шум. Это указывает на отсутствие четких плотностных кластеров и равномерное распределение объектов в пространстве признаков.