ЛАБОРАТОРНАЯ РАБОТА 3

```
# -*- coding: utf-8 -*-
"""lab3.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1aU33DV3eisSMDzG8ZaZAc4UAXMOn5n4q

# Лабораторная работа №3
"""

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import (silhouette_samples, silhouette_score,
                             adjusted_rand_score, v_measure_score,
                             calinski_harabasz_score,
davies_bouldin_score)
from scipy.spatial.distance import cdist
from scipy.cluster.hierarchy import dendrogram, linkage
from sklearn.decomposition import PCA

"""Берем уже известный нам датасет с винами"""

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-quality/winequality-red.csv'
df = pd.read_csv(url, sep=';')
df.head()

X = df.drop('quality', axis=1)
y = df['quality']
real_k = len(y.unique())

plt.figure(figsize=(8, 5))
ax = sns.countplot(x=y, palette='viridis', hue=y, legend=False)
plt.title('Распределение классов')
plt.xlabel('Оценка качества')
plt.ylabel('Количество экземпляров')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center', xytext=(0, 5),
textcoords='offset points')
plt.show()
```

```
"""## Коррелограмма признаков"""

plt.figure(figsize=(10, 8))
sns.heatmap(X.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Корреляционная матрица признаков')
plt.show()

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

"""## Основные функции"""

def plot_silhouette_analysis(data, n_clusters):
    fig, ax1 = plt.subplots(1, 1, figsize=(10, 6))

    clusterer = KMeans(n_clusters=n_clusters, random_state=42)
    cluster_labels = clusterer.fit_predict(data)

    silhouette_avg = silhouette_score(data, cluster_labels)
    sample_silhouette_values = silhouette_samples(data, cluster_labels)

    y_lower = 10
    for i in range(n_clusters):
        ith_cluster_silhouette_values =
sample_silhouette_values[cluster_labels == i]
        ith_cluster_silhouette_values.sort()

        size_cluster_i = ith_cluster_silhouette_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters)
        ax1.fill_betweenx(np.arange(y_lower, y_upper),
                          0, ith_cluster_silhouette_values,
                          facecolor=color, edgecolor=color, alpha=0.7,
                          label=f'Кластер {i}')

        ax1.text(-0.05, y_lower + 0.5 * size_cluster_i, str(i))
        y_lower = y_upper + 10

    ax1.set_title(f"Анализ силуэта для {n_clusters} кластеров")
    ax1.set_xlabel("Значение коэффициента силуэта")
    ax1.set_ylabel("Номер кластера")

    ax1.axvline(x=silhouette_avg, color="red", linestyle="--",
label=f'Средний: {silhouette_avg:.2f}')

    ax1.set_yticks([])
    ax1.set_xticks([-0.2, 0, 0.2, 0.4, 0.6, 0.8, 1])
    ax1.legend(loc='upper right')
    plt.show()
```

```
def evaluate_clusters(data, max_k=10):
    inertias, distortions, silhouettes = [], [], []
    K = range(2, max_k + 1)

    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42).fit(data)
        inertias.append(kmeans.inertia_)
        distortions.append(sum(np.min(cdist(data,
kmeans.cluster_centers_, 'euclidean'), axis=1)) / data.shape[0])
        silhouettes.append(silhouette_score(data, kmeans.labels_))

    fig, axes = plt.subplots(1, 3, figsize=(18, 5))
    axes[0].plot(K, inertias, 'bo-')
    axes[0].set_title('Метод локтя (Инерция)')
    axes[0].set_xlabel('Число кластеров')

    axes[1].plot(K, distortions, 'go-')
    axes[1].set_title('Метод локтя (Искажение)')
    axes[1].set_xlabel('Число кластеров')

    axes[2].plot(K, silhouettes, 'ro-')
    axes[2].set_title('Метрика силуэта')
    axes[2].set_xlabel('Число кластеров')
    plt.tight_layout()
    plt.show()

def plot_projections(data, original_labels, kmeans_labels, ward_labels,
title_prefix=""):
    fig, axes = plt.subplots(1, 3, figsize=(18, 5))

    scatter_orig = axes[0].scatter(data[:, 0], data[:, 1],
c=original_labels, cmap='viridis', alpha=0.6)
    axes[0].set_title(f'{title_prefix} Реальные классы')
    axes[0].legend(*scatter_orig.legend_elements(), title="Классы")

    scatter_kmeans = axes[1].scatter(data[:, 0], data[:, 1],
c=kmeans_labels, cmap='viridis', alpha=0.6)
    axes[1].set_title(f'{title_prefix} Предсказания K-means')
    axes[1].legend(*scatter_kmeans.legend_elements(), title="Кластеры")

    scatter_ward = axes[2].scatter(data[:, 0], data[:, 1],
c=ward_labels, cmap='viridis', alpha=0.6)
    axes[2].set_title(f'{title_prefix} Предсказания Ward')
    axes[2].legend(*scatter_ward.legend_elements(), title="Кластеры")

    plt.show()

def calculate_advanced_metrics(data, true_labels, predicted_labels,
model_name):
    ari = adjusted_rand_score(true_labels, predicted_labels)
```

```
    v_measure = v_measure_score(true_labels, predicted_labels)
    ch_score = calinski_harabasz_score(data, predicted_labels)
    db_score = davies_bouldin_score(data, predicted_labels)

    print(f"--- Дополнительные метрики для {model_name} ---")
    print(f"Adjusted Rand Index (ARI): {ari:.4f}")
    print(f"V-Measure: {v_measure:.4f}")
    print(f"Calinski-Harabasz Score: {ch_score:.4f}")
    print(f"Davies-Bouldin Score: {db_score:.4f}")

"""## K-means на оригинальных данных"""

print(f"Реальное число классов: {real_k}")
evaluate_clusters(X_scaled)

plot_silhouette_analysis(X_scaled, n_clusters=real_k)

kmeans = KMeans(n_clusters=real_k, random_state=42)
kmeans_labels = kmeans.fit_predict(X_scaled)
crosstab_kmeans = pd.crosstab(y, kmeans_labels, rownames=['Original'],
colnames=['Predicted K-means'])
crosstab_kmeans

calculate_advanced_metrics(X_scaled, y, kmeans_labels, "K-means")

"""## Ward на оригинальных данных"""

linked = linkage(X_scaled, method='ward')
plt.figure(figsize=(10, 5))
dendrogram(linked, truncate_mode='level', p=5)
plt.title('Дендрограмма (Метод Уорда)')
plt.show()

ward = AgglomerativeClustering(n_clusters=real_k, linkage='ward')
ward_labels = ward.fit_predict(X_scaled)

calculate_advanced_metrics(X_scaled, y, ward_labels, "Ward")

"""## Диаграмма рассеяния на оригинальных данных"""

plot_projections(X_scaled, y, kmeans_labels, ward_labels, "Оригинальные
признаки:")

"""## PCA"""

pca_full = PCA()
pca_full.fit(X_scaled)
eigenvalues = pca_full.explained_variance_
n_components = np.sum(eigenvalues > 1)
print(f"Количество компонент с собственными числами > 1:
{n_components}")
```

```
pca = PCA(n_components=n_components)
X_pca = pca.fit_transform(X_scaled)

"""## K-means на PCA"""

evaluate_clusters(X_pca)

plot_silhouette_analysis(X_pca, n_clusters=real_k)

kmeans_pca = KMeans(n_clusters=real_k, random_state=42)
kmeans_labels_pca = kmeans_pca.fit_predict(X_pca)
crosstab_kmeans_pca = pd.crosstab(y, kmeans_labels_pca,
rownames=['Original'], colnames=['Predicted K-means (PCA)'])
crosstab_kmeans_pca

calculate_advanced_metrics(X_scaled, y, kmeans_labels_pca, "K-means
PCA")

"""## Ward на PCA"""

linked_pca = linkage(X_pca, method='ward')
plt.figure(figsize=(10, 5))
dendrogram(linked_pca, truncate_mode='level', p=5)
plt.title('Дендрограмма (PCA, Метод Уорда)')
plt.show()

ward_pca = AgglomerativeClustering(n_clusters=real_k, linkage='ward')
ward_labels_pca = ward_pca.fit_predict(X_pca)

calculate_advanced_metrics(X_scaled, y, ward_labels_pca, "Ward PCA")

"""## Диаграмма рассеяния на PCA"""

plot_projections(X_pca, y, kmeans_labels_pca, ward_labels_pca, "После
PCA:")

"""Выводы по ЛР3:

**1. До понижения размерности**
- По графикам метода локтя на исходных признаках нет чётко выраженного
оптимума: инерция и искажение убывают плавно, поэтому число кластеров по
этим графикам определяется неуверенно.
- По силуэту максимум наблюдается при `k = 2` (около `0.21`), но это
говорит скорее о слабой естественной разделимости данных, чем о хорошем
разбиении: классы заметно перекрываются.
- По внешним метрикам качество кластеризации низкое:
  `K-means` — `ARI = 0.0768`, `V-measure = 0.1048`, `CH = 275.3569`, `DB
= 1.4589`;
  `Ward` — `ARI = 0.0610`, `V-measure = 0.0779`, `CH = 225.7710`, `DB =
1.5263`.
```

```
  Это означает, что оба метода слабо совпадают с исходными классами, а
`K-means` на исходных признаках немного лучше `Ward`.

**2. После применения МГК (PCA)**
- Было выделено 4 главные компоненты с собственными числами больше 1, то
есть существенная часть дисперсии данных сохраняется уже в меньшем числе
признаков.
- Внутренние характеристики кластеризации стали лучше: максимум силуэта
вырос примерно до `0.29` при `k = 5–6`, а на графиках кластеры визуально
стали компактнее.
- При этом совпадение с исходными классами почти не улучшилось:
  `K-means PCA` — `ARI = 0.0726`, `V-measure = 0.1002`, `CH = 272.4725`,
`DB = 1.4762`;
  `Ward PCA` — `ARI = 0.0507`, `V-measure = 0.0651`, `CH = 228.8654`,
`DB = 1.5821`.
  Значит, PCA улучшила геометрию кластеров, но не устранила основную
проблему — сильное перекрытие и дисбаланс классов.

**3. Общий итог**
- Для этого датасета качество кластеризации ограничено самим
распределением классов: крупные классы перемешиваются с другими, а
небольшие плохо отделяются.
- Оценивать результат здесь лучше по совокупности метрик: силуэт и
графики показывают внутреннюю структуру, а `ARI`, `V-measure`,
`Calinski-Harabasz` и `Davies-Bouldin` — насколько кластеризация
действительно удачна.
- Наиболее стабильным из рассмотренных вариантов выглядит `K-means`, но
в целом все методы дают скорее слабое, чем хорошее разделение объектов.
"""
```

ЛАБОРАТОРНАЯ РАБОТА 4

```
# -*- coding: utf-8 -*-
"""lab4.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1BR-
i3u76VRkl7rZ94T_YArEPP5nxqfED

# Лабораторная работа №4
"""

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

from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import AffinityPropagation, KMeans, DBSCAN,
SpectralClustering
from sklearn.metrics import (silhouette_samples, silhouette_score,
                             adjusted_rand_score, v_measure_score,
                             calinski_harabasz_score,
davies_bouldin_score)

"""Берем уже известный нам датасет с винами"""

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/wine-
quality/winequality-red.csv'
df = pd.read_csv(url, sep=';')
df.head()

X = df.drop('quality', axis=1)
y = df['quality']
real_k = len(y.unique())

plt.figure(figsize=(8, 5))
ax = sns.countplot(x=y, palette='viridis', hue=y, legend=False)
plt.title('Распределение классов')
plt.xlabel('Оценка качества')
plt.ylabel('Количество экземпляров')
for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}', (p.get_x() + p.get_width() /
2., p.get_height()),
```

```
                ha='center', va='center', xytext=(0, 5),
textcoords='offset points')
plt.show()

plt.figure(figsize=(10, 8))
sns.heatmap(X.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Корреляционная матрица признаков')
plt.show()

"""## Стандартизация датафрейма"""

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

"""## Расчет главных компонент и отбор по критерию Кайзера"""

pca_full = PCA().fit(X_scaled)
var = pca_full.explained_variance_
n = len(np.where(var > 1)[0])
print(f"Число главных компонент с собственным значением > 1: {n}")

pca = PCA(n_components=n).fit(X_scaled)
X_pca = pca.transform(X_scaled)

"""## Affinity Propagation"""

ap = AffinityPropagation(random_state=42)
ap_labels = ap.fit_predict(X_pca)
ap_n_clusters = len(ap.cluster_centers_indices_)
print(f"Число кластеров Affinity Propagation: {ap_n_clusters} (Реальное:
{real_k})")

plt.figure(figsize=(10, 7))

sns.scatterplot(
    x=X_pca[:, 0],
    y=X_pca[:, 1],
    hue=ap_labels,
    palette="turbo",
    s=15,
    alpha=0.5,
    edgecolor=None,
    legend=False
)

plt.scatter(
    ap.cluster_centers_[:, 0],
    ap.cluster_centers_[:, 1],
    color='black',
    marker='+',
    s=60,
```

```
    linewidths=1,
    label='Cluster Centers',
    zorder=10
)

plt.title("Affinity Propagation", fontsize=16, fontweight='light')
plt.xlabel("PCA 1", fontsize=10, color='grey')
plt.ylabel("PCA 2", fontsize=10, color='grey')

plt.tick_params(axis='both', which='both', bottom=False, left=False,
labelbottom=False, labelleft=False)
plt.legend(frameon=False, loc='upper right', fontsize=10)

plt.tight_layout()
plt.show()

"""## Метод силуэта"""

def plot_silhouette_analysis(data, clusterer, n_clusters):
    cluster_labels = clusterer.fit_predict(data)

    mask = cluster_labels != -1
    clean_data = data[mask]
    clean_labels = cluster_labels[mask]
    unique_labels = np.unique(clean_labels)
    n_clusters_real = len(unique_labels)

    fig, ax = plt.subplots(1, 1, figsize=(10, 6))
    silhouette_avg = silhouette_score(clean_data, clean_labels)
    sample_silhouette_values = silhouette_samples(clean_data,
clean_labels)

    y_lower = 10
    for i, cluster_idx in enumerate(unique_labels):
        ith_cluster_values = sample_silhouette_values[clean_labels ==
cluster_idx]
        ith_cluster_values.sort()

        size_cluster_i = ith_cluster_values.shape[0]
        y_upper = y_lower + size_cluster_i

        color = cm.nipy_spectral(float(i) / n_clusters_real)
        ax.fill_betweenx(np.arange(y_lower, y_upper), 0,
ith_cluster_values,
                         facecolor=color, edgecolor=color, alpha=0.7,
label=f'Кластер {i}')

        ax.text(-0.05, y_lower + 0.5 * size_cluster_i, str(cluster_idx))
        y_lower = y_upper + 10

    ax.set_title(f"Анализ силуэта для {n_clusters} кластеров")
```

```
    ax.set_xlabel("Значение коэффициента силуэта")
    ax.set_ylabel("Номер кластера")

    ax.axvline(x=silhouette_avg, color="red", linestyle="--",
label=f'Средний: {silhouette_avg:.2f}')
    ax.set_yticks([])

    ax.legend(loc='upper left', bbox_to_anchor=(1.05, 1),
borderaxespad=0.)
    plt.show()

def calculate_advanced_metrics(data, true_labels, predicted_labels,
model_name):
    ari = adjusted_rand_score(true_labels, predicted_labels)
    v_measure = v_measure_score(true_labels, predicted_labels)
    ch_score = calinski_harabasz_score(data, predicted_labels)
    db_score = davies_bouldin_score(data, predicted_labels)

    print(f"--- Дополнительные метрики для {model_name} ---")
    print(f"Adjusted Rand Index (ARI): {ari:.4f}")
    print(f"V-Measure: {v_measure:.4f}")
    print(f"Calinski-Harabasz Score: {ch_score:.4f}")
    print(f"Davies-Bouldin Score: {db_score:.4f}")

"""## Spectral Clustering"""

best_spectral_k = 2
best_spectral_score = -1

for k in range(2, 10):
    labels = SpectralClustering(n_clusters=k, random_state=42,
assign_labels='kmeans').fit_predict(X_pca)
    score = silhouette_score(X_pca, labels)
    if score > best_spectral_score:
        best_spectral_score = score
        best_spectral_k = k

print(f"Оптимальное число кластеров Spectral Clustering по силуэту:
{best_spectral_k}")

sc = SpectralClustering(n_clusters=best_spectral_k, random_state=42,
assign_labels='kmeans')
plot_silhouette_analysis(X_pca, sc, best_spectral_k)

preds = sc.fit_predict(X_pca)
calculate_advanced_metrics(X_pca, y, preds, "Spectral Clustering")

"""## DBSCAN"""

scores = []
n_clusters_list = []
```

```
eps_range = np.arange(0.3, 2.0, 0.1)

for eps in eps_range:
    model = DBSCAN(eps=eps)
    labels = model.fit_predict(X_pca)

    n_clusters = len(set(labels)) - (1 if -1 in labels else 0)
    if n_clusters > 1:
        score = silhouette_score(X_pca, labels)
        scores.append(score)
        n_clusters_list.append(n_clusters)
    else:
        scores.append(0)
        n_clusters_list.append(n_clusters)

fig, ax1 = plt.subplots(figsize=(8, 4))
ax1.plot(eps_range, scores, 'r-')
ax1.set_xlabel('Epsilon (eps)')
ax1.set_ylabel('Silhouette Score', color='r')

ax2 = ax1.twinx()
ax2.plot(eps_range, n_clusters_list, 'b--')
ax2.set_ylabel('Число кластеров', color='b')

plt.title('DBSCAN: Silhouette vs Epsilon')
plt.show()

best_idx = np.argmax(scores)
best_dbscan_eps = eps_range[best_idx]
best_dbscan_clusters = n_clusters_list[best_idx]

db = DBSCAN(eps=best_dbscan_eps)
plot_silhouette_analysis(X_pca, db, best_dbscan_clusters)

preds = db.fit_predict(X_pca)
calculate_advanced_metrics(X_pca, y, preds, "DBSCAN")

"""## Финальное сравнение"""

kmeans_labels = KMeans(n_clusters=real_k,
random_state=42).fit_predict(X_pca)
dbscan_labels = DBSCAN(eps=best_dbscan_eps).fit_predict(X_pca)
spectral_labels = SpectralClustering(n_clusters=best_spectral_k,
random_state=42, assign_labels='kmeans').fit_predict(X_pca)

print(f"Реальное число кластеров: {real_k}")
print(f"K-Means: {len(set(kmeans_labels))}")
print(f"DBSCAN: {len(set(dbscan_labels)) - (1 if -1 in dbscan_labels
else 0)} (исключая шум)")
print(f"Spectral Clustering: {len(set(spectral_labels))}")
```

```
calculate_advanced_metrics(X_pca, y, kmeans_labels, "KMeans")

fig, axes = plt.subplots(2, 2, figsize=(14, 10))
axes = axes.flatten()

titles = ['Реальные классы', 'Предсказания K-Means', 'Предсказания
DBSCAN', 'Предсказания Spectral']
label_sets = [y, kmeans_labels, dbscan_labels, spectral_labels]

for i in range(4):
    scatter = axes[i].scatter(X_pca[:, 0], X_pca[:, 1], c=label_sets[i],
cmap='tab10', edgecolor='k', s=50)
    axes[i].set_title(titles[i])
    axes[i].set_xlabel("PCA 1")
    axes[i].set_ylabel("PCA 2")

    legend = axes[i].legend(*scatter.legend_elements(),
title="Кластеры", loc="best")
    axes[i].add_artist(legend)

plt.tight_layout()
plt.show()
```