МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Институт компьютерных наук
Кафедра автоматизированных систем управления

Лабораторная работа №3

по дисциплине «Машинное обучение»

Студент М-РИТ-25-1 _____ Станиславчук С. М.

(подпись, дата)

Руководитель

Доцент, д.т.н. _____ Сараев П. В.

(подпись, дата)

Липецк 2025

1. Задание кафедры

1) Постройте кластеризацию данных для определния вида ирисов (база Iris) на основе четырех параметров (без использования информации о классе). Для кластеризации используйте метод k-средних и иерархический метод Уорда. Исследуйте влияние аргументов процедур, реализующих указанные выше методы, на результат кластеризации. Постройте графики.

2) Определите оптимальное количество кластеров на основе метода "локтя".

3) Сравните результаты кластеризации для случая 3-х кластеров с реальными классами, представленными в базе данных Iris.

4) Сделайте выводы о проделанной работе.

2. Цель работы

Цель работы — изучение методов кластеризации данных и определения оптимального количества кластеров.

3. Ход работы

В качестве языка программирования для выполнения лабораторной работы, я выбрал Python.

Влияние аргументов процедуры, реализующую метод K-средних

Kmeans()

n_clusters — значение числа кластеров.

random_state — сид случайной генерации.

n_init — число запуска алгоритма с разным центроидным сидом.

init:

(default) «k-means++»: выбирает начальные центроиды кластера, используя выборку, основанную на эмпирическом распределении вероятностей вклада точек в общую инерцию.

«random» - выбирает n_clusters наблюдений (строк) случайным образом из данных для начальных центроидов.

fcluster()

Z — матрица слияний

t — для критериев "inconsistent", "distance" или "monocrit" это пороговое значение, применяемое при формировании плоских кластеров. Для критериев "maxclust" или "maxclust_monocrit" это максимальное количество запрашиваемых кластеров.

Criterion:

«inconsistent» - если узел кластера и все его потомки имеют значение "inconsistent" (неконсистентности) меньше или равно `t`, то все листья под этим узлом попадают в один плоский кластер. Если ни один “нормальный” (не одноэлементный) кластер не удовлетворяет этому условию, тогда каждый элемент будет в своём собственном кластере.

«distance» - делит дерево так, чтобы расстояние между любыми двумя точками внутри одного кластера (т.н. *копенетическое расстояние*) не превышало `t`.

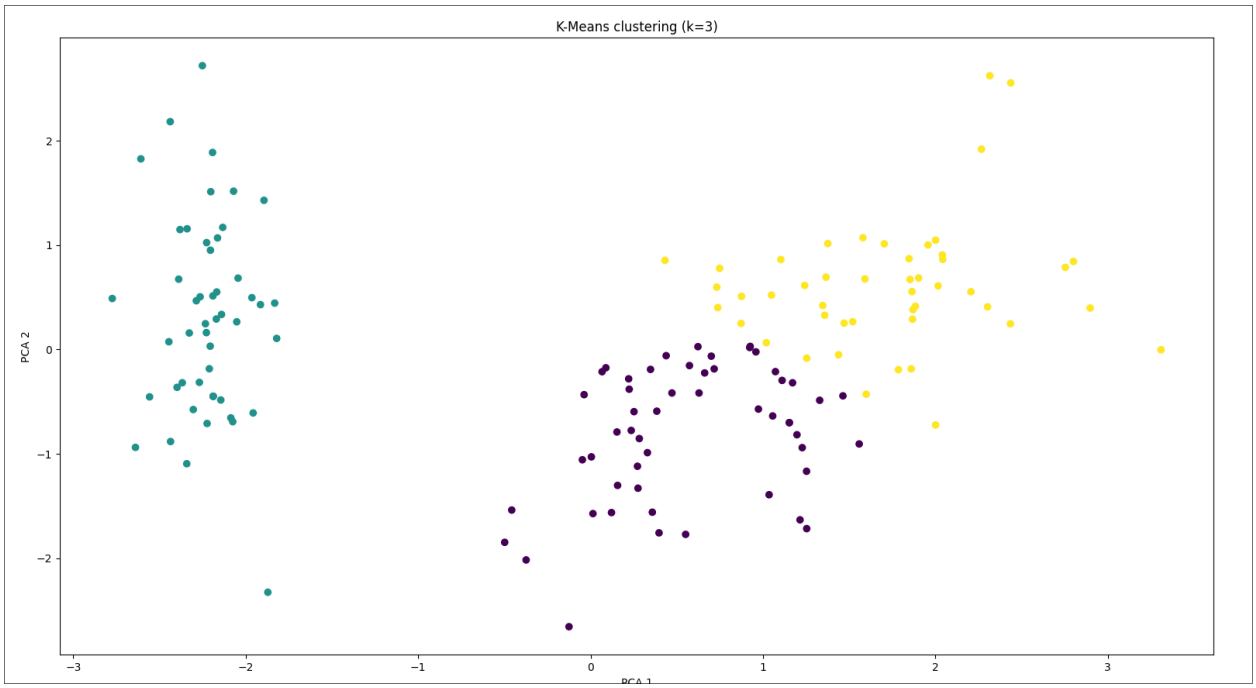
«maxclust» - Находит минимальный порог `r`, при котором расстояние между любыми точками внутри одного кластера не больше `r`, и при этом получается не больше `t` кластеров.

«monocrit» - Делает плоские кластеры на основе пользовательского “монотонного критерия”. Кластер формируется, если `monocrit[j] <= t`.

«maxclust_monocrit» - Формирует кластеры из всех узлов, где `monocrit[i] <= r`, причём подбирает минимальное `r`, чтобы итоговое количество кластеров было ≤ `t`.

Пример выполнения программы №1 (Kmeans):

функция - KMeans(n_clusters=3, random_state=42, n_init=5)



Образовалось 3 кластера, очень похожих на реальные классы ирисов.

Сравнение с реальными классами (setosa получилось отделить без ошибок, а вот virginica и versicolor «смешались» между собой).

```
Confusion Matrix (True labels vs Clusters):
[[ 0 50  0]
 [39  0 11]
 [14  0 36]]
```

3. Вывод

n_init влияет довольно сильно, лучше всего запустить алгоритм несколько раз, значение=5 было «лучшим» для моей задачи, значения выше не меняли результат.

algorithm — (default) «lloyd», «elkan» может ускорить получение результатов, используя больше вычислительной памяти. На результат кластеризации не влияют.

«random» зависит от random_state, а также может требовать больше n_init для такой же точности результата как «k-means++»

Приложение 1

```
from sklearn.metrics import confusion_matrix, adjusted_rand_score
from sklearn.preprocessing import LabelEncoder
import pandas as pd

def compare(y_true, y_pred, target_names=None):
    # Ensure y_true and y_pred are numeric
    if y_true.dtype == object:
        le = LabelEncoder()
        y_true = le.fit_transform(y_true)
        if target_names is None:
            target_names = le.classes_

    cm = confusion_matrix(y_true, y_pred)
    ari = adjusted_rand_score(y_true, y_pred)

    print("▫ Confusion Matrix (True labels vs Clusters):")
    print(cm)
    print(f"\n▫ Adjusted Rand Index (ARI): {ari:.3f}")

    return cm, ari

def plt_kmeans():
    plt.figure(figsize=(6,5))
    plt.scatter(X_pca[:,0], X_pca[:,1], c=labels_kmeans, cmap='viridis')
    plt.title(f"K-Means clustering (k={cluster_count})")
    plt.xlabel("PCA 1")
    plt.ylabel("PCA 2")
    plt.show()

def plt_ward_tree():
    plt.figure(figsize=(10, 6))
    dendrogram(Z)
    plt.title("Hierarchical clustering (Ward's method)")
    plt.xlabel("Samples")
    plt.ylabel("Distance")
    plt.show()

def plt_ward_clust():
    plt.figure(figsize=(6,5))
    plt.scatter(X_pca[:,0], X_pca[:,1], c=labels_ward, cmap='plasma')
    plt.title(f"Hierarchical clustering (Ward, { cluster_count } clusters)")
    plt.xlabel("PCA 1")
    plt.ylabel("PCA 2")
    plt.show()

def arg_influence():
    for k in [2, 3, 4, 5, 6]:
        km = KMeans(n_clusters=k, random_state=42)
        km.fit(X_scaled)
        plt.scatter(X_pca[:,0], X_pca[:,1], c=km.labels_, cmap='viridis')
        plt.title(f"KMeans (k={k})")
        plt.show()

def elbow():
    inertias = []
    k_values = range(1, 11)

    for k in k_values:
        kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
        kmeans.fit(X)
        inertias.append(kmeans.inertia_)

    plt.figure(figsize=(6,4))
    plt.plot(k_values, inertias, marker='o')
    plt.title("Elbow Method for K-Means")
    plt.xlabel("Number of clusters (k)")
    plt.ylabel("Inertia")
```

```
    plt.grid(True)
    plt.show()



# 1. Постройте кластеризацию данных для определния вида ирисов
# (база Iris) на основе четырех параметров (без использования информации
# о классе). Для кластеризации используйте метод k-средних и иерархический
# метод Уорда. Исследуйте влияние аргументов процедур, реализующих указанные
# выше методы, на результат кластеризации. Постройте графики.

import pandas as pd

iris = pd.read_csv('iris.csv')
print(iris)

from sklearn.preprocessing import StandardScaler

y_true = iris['target']

# unsupervised learning
X = iris.drop(columns=['target'], errors='ignore')

# 1. K-mean кластеризация
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

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

cluster_count = 3
kmeans = KMeans(n_clusters=cluster_count, random_state=42, n_init=5)
kmeans.fit(X_scaled)
labels_kmeans = kmeans.labels_

compare(y_true, labels_kmeans, target_names=iris['target'].unique())

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

plt_kmeans()

# 2. Иерархическая кластеризация (метод Уорда)
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

Z = linkage(X_scaled, method='ward')

# plt_ward_tree()

criterion1 = 'maxclust'
criterion2 = 'inconsistent'
criterion3 = 'distance'

depth = 5
# criterion4 = 'monocrit'
# criterion5 = 'maxclust_monocrit'
labels_ward = fcluster(Z, cluster_count, criterion=criterion1, depth=depth)

compare(y_true, labels_ward, target_names=iris['target'].unique())

# 3. Влияние аргументов
arg_influence()

# 2. Опрееделите оптимальное количество кластеров
# на основе метода "локтя".
elbow()

# 3. Сравните результаты кластеризации для случая 3-х кластеров
```

```
# с реальными классами, представленными в базе данных Iris

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import confusion_matrix

X = iris.drop(columns=['target'], errors='ignore')
y_true = iris['target'] if 'target' in iris.columns else None

# Нормализация данных (для KMeans)
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_scaled)

kmeans = KMeans(n_clusters=3, random_state=42, n_init=10)
kmeans.fit(X_scaled)
y_kmeans = kmeans.labels_

le = LabelEncoder()
y_true_encoded = le.fit_transform(y_true)

compare(y_true_encoded, y_kmeans)

print("\nКоды классов:")
for i, cls in enumerate(le.classes_):
    print(f"{i}: {cls}")
```