|  |  |  |  |
| --- | --- | --- | --- |
| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка  ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**  Дисципліна  **«Спеціалізоване програмування автоматизованих систем »**  **Екзаменаційна робота**  **Білет №** | | | |
| **Виконав:** | Гоша Давід Олександрович | **Перевірила**: | Ковтун Оксана Іванівна |
| Група | ІПЗ-33 | Дата перевірки |  |
| Форма навчання | денна | Бали |  |
| Спеціальність | 121 |
| 2023 | | | |

**Завдання:**

Проведіть кластерізацію даних за алгоритмом k-means, використавши відстань Мінковського, із застосуванням бібліотеки Scikit-Learn.

* Підготуйте набір даних з файлу «seeds.csv».
* Виведіть у консоль координати центроїдів кожного кластера.
* Виведіть у консоль загальну внутрішньо-кластерну суму квадратів відстаней від екземплярів до найближчого центроїда.
* Виведіть у консоль кількість ітерацій, які виконує алгоритм k-means.
* Побудуйте графіки результатів кластеризації для декількох параметрів попарно (не менше 3-х), додайте на графіки центроїди.
* Порахуйте кількість екземплярів у кожному кластері, порівняйте з відомим розподілом на класи.

**Код:**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

from sklearn.preprocessing import StandardScaler

from itertools import combinations

def load\_and\_scale\_data(filename):

*# Load data*

    df = pd.read\_csv(filename)

    X = df.drop(columns=['class']).values

*# Scale the data*

    sc = StandardScaler()

    X = sc.fit\_transform(X)

    return X, df['class'].values

def run\_kmeans(X, n\_clusters=3):

*# KMeans*

    kmeans = KMeans(n\_clusters=n\_clusters, random\_state=0)

    kmeans.fit(X)

    return kmeans

def print\_kmeans\_info(kmeans):

    print("Centroids: ", kmeans.cluster\_centers\_)

    print("Sum of squared distances: ", kmeans.inertia\_)

    print("Number of iterations: ", kmeans.n\_iter\_)

def plot\_clusters(X, kmeans):

    pairs = list(combinations(range(X.shape[1]), 2))[:3]

    for i, (feat1, feat2) in enumerate(pairs):

        plt.figure(i)

        plt.scatter(X[:, feat1], X[:, feat2], c=kmeans.labels\_)

        plt.scatter(kmeans.cluster\_centers\_[:, feat1], kmeans.cluster\_centers\_[:, feat2], s=300, c='red')

        plt.show()

def count\_cluster\_instances(kmeans):

    clusters, counts = np.unique(kmeans.labels\_, return\_counts=True)

    for cluster, count in zip(clusters, counts):

        print(f"Cluster {cluster}: {count} instances")

def compare\_to\_classes(kmeans, classes):

    class\_counts = np.unique(classes, return\_counts=True)

    for class\_, count in zip(\*class\_counts):

        print(f"Class {class\_}: {count} instances")

def main():

    X, classes = load\_and\_scale\_data('seeds.csv')

    kmeans = run\_kmeans(X)

    print\_kmeans\_info(kmeans)

    plot\_clusters(X, kmeans)

    count\_cluster\_instances(kmeans)

    compare\_to\_classes(kmeans, classes)

if \_\_name\_\_ == "\_\_main\_\_":

    main()