Київський національний університет імені Тараса Шевченка
Факультет інформаційних технологій

Кафедра програмних систем і технологій

ЗВІТ

з практичної роботи № 4

Тема: «Алгоритми кластерізації»

Дисципліна «Спеціалізоване програмування автоматизованих систем»

Підготував:
студент гр. ІПЗ-33(1)
Мішак Максим

Перевірила:
Ніколаєнко Анастасія Юріївна

Завдання :

Проведіть кластеризацію даних за алгоритмом k-means, використавши відстань за варіантом. Дані можна взяти з файлу iris.csv. Порахуйте кількість екземплярів у кожному кластері, порівняйте з відомим розподілом на класи.

Варіант 7 :

5. Відстань **Мінковського** – параметрична метрика в евклідовому просторі, яку можна розглядати як узагальнення евклідової відстані та відстані міських кварталів.

$$\rho(x,x') = \sqrt[p]{\sum_i^n (x_i - x_i')^p},$$

Хід роботи :

Лабараторна робота схожа на роботу номер 2 . Різниця полягає в тому , що ЛР№2 ми робили алгоритм класифікаї , в той час як в теперішній розроблявся алгоритм кластеризації типу k-means . Суть алгоритму кластеризації k-means в тому , що :

- Ті данні , які ми передаємо в программу аналізуються та на основі аналізу по площині данних псевдо випадковим чином розміщуються центроїди . Центроїда - це середнє значення координтан точок , що передаються .
- Після первичного розташування цетроїд(їх кількість залежить від кількості кластерів , яку визначає програміст) алгоритм рахує відстань по заданій програмістом формулі до кожної з точок , які первично належать до одного з кластерів .
- Після підрахунку відстані до точок перераховується середнє значення координат точок кластеру , та центроїда переміщується на обраховані координати .
- Крок 2 та 3 повторюється , поки :
    - Центроїда не стане стабільною
    - Алгоритм пройде зазначену кількість ітерацій

Почати варто з того , що зчитаємо датасет з файлу iris.csv

```
# Завантаження даних та міток.

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

data = iris_data.drop(columns=['variety'])

labels = iris_data['variety']
```

Визначаємо параметр кластеризації(кількість кластерів)

```
k = 3
```

Викликаємо функцію кластеризації . Функція приймає 2 параметри : датасет та кількість кластерів

```
clusters = k_means(data, k)
```

Код функції :

```
def k_means(data, k, max_iterations=100):

    centroids = data.sample(k).values  # Ініціалізація центроїдів випадковими точками даних.

    for _ in range(max_iterations):

        # Створення порожніх списків кожного кластера.

        clusters = {i: [] for i in range(k)}

        # Ітерація з кожної точки даних.

        for point in data.values:

            # Обчислення відстані між точкою та кожним центроїдом.

            distances = [distance_minkowski(point, centroid ,2) for centroid in centroids]
```

```
            # Визначення індексу кластера з найменшою відстанню.
            cluster_index = np.argmin(distances)
            # Додавання крапки у відповідний кластер.
            clusters[cluster_index].append(point)
        # Обчислення нових центроїдів з урахуванням середніх значень точок у кожному кластері.
        new_centroids = [np.mean(clusters[i], axis=0) for i in range(k)]

        # Перевірка на рівність старих та нових центроїдів. Якщо центроїди не змінилися, вихід із циклу.
        if np.all([np.allclose(a, b, rtol=1e-5) for a, b in zip(centroids, new_centroids)]):
            break
        centroids = new_centroids
    return clusters
```

Розібравши функцію на складові можемо побачити такі ключові етапи :

`centroids = data.sample(k).values`	Розбиття набору данних за допомогою `sample` на `k` частин та знаходження середніх значень для координат центроїд
`clusters = {i: [] for i in range(k)}`	Створення проміжних списків для кожного кластеру
`distances =`	Обраховуємо відстань від

`[distance_minkowski(point, centroid ,2) for centroid in centroids]`	центроїди до точки за допомогою відстані Мінковського для кожної центроїди зі списку центроїд
`cluster_index = np.argmin(distances)`	Визначаємо індекс кластеру з найменшою відстанню
`clusters[cluster_index].append(point)`	Додаємо крапку у відповідний кластер
`new_centroids = [np.mean(clusters[i], axis=0) for i in range(k)]`	Обчислюємо середні значення координат точок та переміщуємо центроїду туди
`if np.all([np.allclose(a, b, rtol=1e-5) for a, b in zip(centroids, new_centroids)]):`	Перевірка на те , чи співпадають координати старих та нових центроїд . Якщо співпадають - виконується команда `break` , якщо не співпадають - `centroids = new_centroids`

Для обрахунку відстані , відповідно до методичних указань , я маю використовувати відстань Мінковського . Відстань Мінкоського собою являє загальну назву сімейства n-вимірних метрик , яка об’єднує два поняття : відстань Евкліда та відстань Манхеттену . В застосунку я реалізував це за допомогою вункції `distance_minkowski` , код якої виглядає так :

```
def distance_minkowski(point1, point2, p):

    if len(point1) != len(point2):

        raise ValueError("Кількість координат в точках не співпадає")

    if p < 1:

        raise ValueError("Параметр p повинен бути не менше 1")
```

```
    sum_of_powers = sum([abs(point1[i]-point2[i])**p
for i in range(len(point1))])

    distance = sum_of_powers**(1/p)

    return distance
```

Функція приймає 3 параметра - координата центроїди , координата точки , та індексний параметр p . Параметр p повинен мати ціле значення та бути більшим за 0 , тільки при такій умові функція може рахувати значення.

`if len(point1) != len(point2):` `  raise ValueError("Кількість координат в точках не співпадає")`	Перевірка на кількість елементів кортежів , так як це є опорним значенням для коректного обрахування . Якщо в одному з кортежів значень буде більше або менше - програма не буде працювати коректно
`if p < 1:` `  raise ValueError("Параметр p повинен бути не менше 1")`	Перевірка значення індексного покажчика p . Так як він повинен відповідати умові p<1
`sum_of_powers = sum([abs(point1[i]-point2[i])**p for i in range(len(point1))])` `distance = sum_of_powers**(1/p)`	Обрахунок математичної формули відстані мінковського для кожної з точок

Для виведення інфографіки я використовую функцію `cluster_purity`. Функція приймає два параметра : визначені значення алгоритмом та істинні значення . Код функції виглядає так :

`label_count = {}` `for label in ground_truth_labels:`	В цьому фрагменті значення визначенного кластеру порівнюють зі значення істиного кластеру ,

`if label in label_count: label_count[label] += 1`	якщо воно співпадає - додається 1 до лічильника
`else: label_count[label] = 1`	Якщо значення не збігається - значення не змінюється
`majority_count = max(label_count.values())`	Знаходження найбільшого індексу в `label_count`
`purity = majority_count / len(ground_truth_labels)`	Знаходження чистоти шляхом ділення максимального показника лічильника на кількість істинних значень

Для виводу інформації та аналітики використовується наступний фрагмент коду :

```
print("Number of instances in each cluster:")

for i, cluster in clusters.items():

    cluster_labels = []

    for point in data.values:

        for c in cluster:

            if np.array_equal(point, c):


cluster_labels.append(labels[data.index[data.apply(l
ambda x: np.array_equal(x, point),
axis=1)].tolist()[0]])

                break

    purity, majority_label =
cluster_purity(cluster_labels, labels)

    print(f"Cluster {i + 1}: {len(cluster)}
instances, Majority label: {majority_label}, Purity:
{purity:.2f}")
```

```
species_counts =
iris_data['variety'].value_counts().sort_index()

print("\nKnown class distribution:")

for index, count in species_counts.items():

    print(f"{index}: {count} instances")
```

Цей фрагмент відповідає виключно за вивід інформації на консоль користувача .

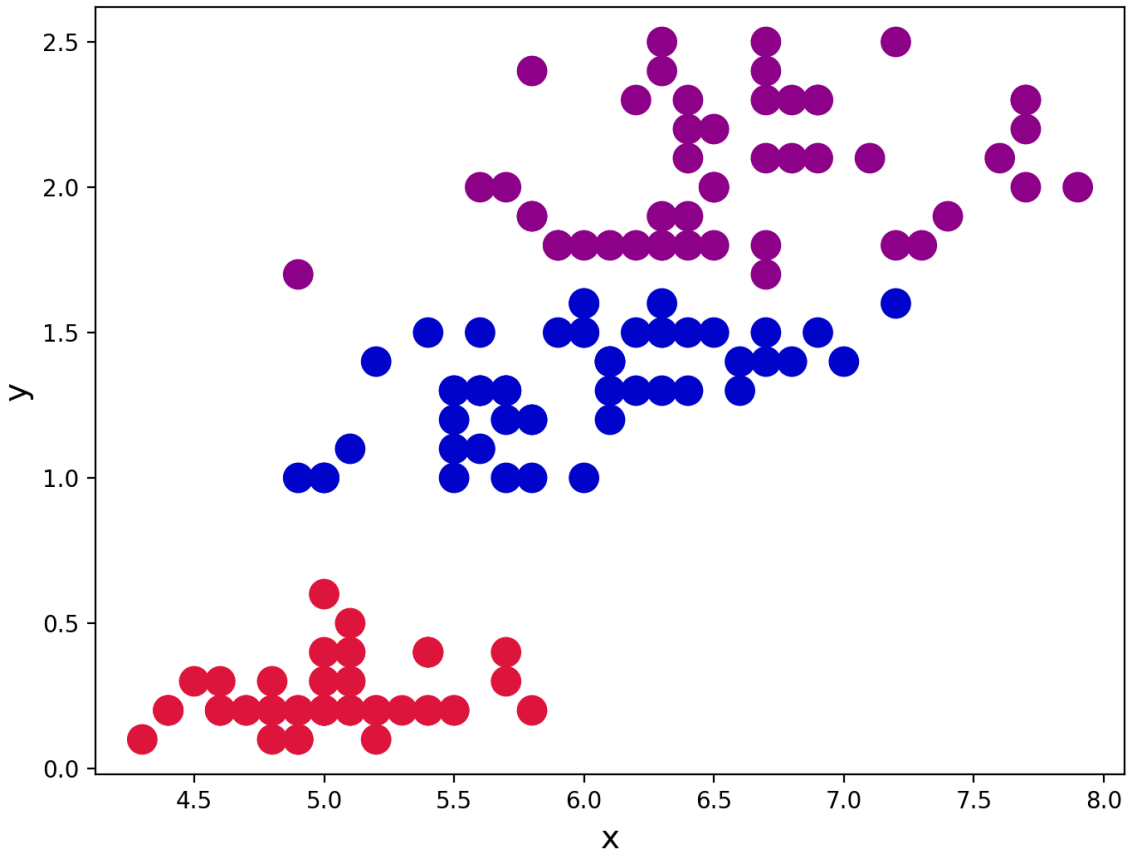
`for i, cluster in clusters.items():`	Цикл , який проходить по всіх кластерах , де “`i`” - індекс поточного кластера а “`cluster`” множина точок , які належать до цього кластера
`if np.array_equal(point, c):`	Перевірка на те , чи збігається точка з поточною точкою кластера
`cluster_labels.append(labels[data.index[data.apply(lambda x: np.array_equal(x, point), axis=1)].tolist()[0]])`	Якщо точки збігаються , то до списку міток кластеру додається мітка цієї точки
`purity, majority_label = cluster_purity(cluster_labels, labels)`	Визначається чистота та більшість міток для поточного кластера за допомогою функції `cluster_purity`
`species_counts = iris_data['variety'].value_counts().sort_index()`	Змінна `species_counts`, яка зберігає в собі кількість екземплярів кожного кластеру відсортована за індексом

Для наочної екранізації використовується фрагмент коду :

```
CustomColorMap =
ListedColormap(['crimson','mediumblue','darkmagenta'
])

fix , ax = plt.subplots(figsize=(8,6))

plt.scatter(x=iris_data['sepal.length'],y=iris_data[
'petal.width'],s=150,c=iris_data['petal.width'].asty
pe('category'),

cmap= CustomColorMap)

ax.set_xlabel(r'x', fontsize=14)

ax.set_ylabel(r'y', fontsize=14)

plt.show()
```

Результат роботи программи :

```
/Users/macbook/Desktop/KHY/venv/bin/python /Users/macbook/Desktop/KHY/venv/main.py
Number of instances in each cluster:
Cluster 1: 38 instances, Majority label: Setosa, Purity: 0.33
Cluster 2: 62 instances, Majority label: Setosa, Purity: 0.33
Cluster 3: 50 instances, Majority label: Setosa, Purity: 0.33

Known class distribution:
Setosa: 50 instances
Versicolor: 50 instances
Virginica: 50 instances
```



Висновок :

Проаналізувавши результат роботи програми можна зробити висновок :

Программа справно виконує задачу кластеризації та розділяє дата сет на відповідну кількість кластерів . Проте проаналізувавши данні , які

виводяться в консоль видно , що программа допускає деяку неточності , наприклад чистота данних становить 33 відсотка . Для відстані Мінковського , яку згідно методичних вказань я мав використовувати є дві умови : показник p для квадратного кореню повинен бути :

- Індекс p повинен бути цілим числом
- Індекс p повинен бути більшим 1

При значенню індексу p=1 - відстань Мінковського стає еквівалентною Евклідові відстані , при p=2 еквівалентною манхеттенською . Найвища точність , якої вийшло досягнути є при значенні p=1 . В такому випадку , можна сказати , що відстань є Евклідовою . При порівнянні значень з додатком , який рахує Евклідову відстань - результати зійшлися . З чого можна зробити висновок , що програма є досить точною