НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Кафедра комп’ютерних наук

Теорія розпізнавання образів та класифікації в системах штучного інтелекту

Лабораторна робота №5

Виконав:

Студент групи КН-20002Б

Кропивка Анатолій Анатолійович

Київ 2023

Тема: Дослідження методів кластерного аналізу у розпізнаванні образів.

Мета: Дослідження способів класифікації образів на основі методів кластерного аналізу. Розробка програмної системи, що реалізує розпізнавання образів на основі зазначеного методу. Дослідження функціонування програмного застосунку при вирішенні задач класифікації образів на основі методів кластеного аналізу. Отримання практичних навичок з розробки програмних систем розпізнавання образів.

Підготовка до роботи: Вивчити й уяснити тереотичні відомості щодо способів класифікації образів на основі методів кластерного аналізу.

Хід роботи:

1. Розробити програмний застосунок на мові програмування C++, що реалізує розпізнавання образів на основі зазначеного методу.
2. Дослідити функціонування розробленого програмного застосунку на прикладах.
3. За результатами досліджень скласти звіт з описом отриманих результатів та обґрунтованими висновками.

Завдання: Класифікувати довільні реальні образи (об’єкти) – банки, фірми, інвестиційні фонди, кредитні спілки тощо за кількома показниками. Наприклад, власний капітал, прибуток і тому подібне.

Опис програми

Кластерний аналіз використовується для групування схожих образів у кластери з метою виявлення закономірностей та визначення принципів класифікації.

Програма застосовує метод максимінної відстані для визначення кластерів у вхідних образах. Вона групує схожі образи разом, створюючи кластери. Після визначення кластерів, вона може використати ці кластери для класифікації нових образів. Також може призначити новий образ до певного кластеру на підставі його схожості з образами у кластері.

Основа програми будується на попередніх лабораторних роботах.

Код програми

```
#include <Windows.h>

#include <algorithm>
#include <cfloat>
#include <chrono>
#include <cmath>
#include <cstdlib>
#include <format>
#include <iostream>
#include <random>
#include <stdexcept>
#include <string>
#include <thread>
#include <vector>

using namespace std;

const double get_double()
{
    double value {};
    string str {};

    try {
        getline(cin, str);

        size_t position {};
        value = stod(str, &position);

        if (position != str.size()) {
            throw runtime_error("There are characters after the number!");
```

```
        }
        if (value < 0) {
            throw runtime_error("The entered value cannot be negative");
        }
    } catch (const exception& e) {
        cout << format("Exception: {}. The return value is 0!", e.what()) << endl;

        value = 0;
    }

    return value;
}

const int gen_random_int_in_range(size_t start, size_t end)
{
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<int> dis(start, end);

    return dis(gen);
}

class Animal {
protected:
    double height;
    double weight;
    int cluster;

public:
    Animal()
        : height {}
        , weight {}
```

```
    , cluster {}
{
}

Animal(double height, double weight)
    : height(height)
    , weight(weight)
    , cluster {}
{
}

const double get_height() const
{
    return height;
}

void set_height(double height)
{
    this->height = height;
}

const double get_weight() const
{
    return weight;
}

void set_weight(double weight)
{
    this->weight = weight;
}

const int get_cluster() const
```

```
{
    return cluster;
}

void set_cluster(int cluster)
{
    this->cluster = cluster;
}

Animal& from()
{
    cout << "Enter the data about an animal." << endl;

    cout << "Height: ";
    height = get_double();

    cout << "Weight: ";
    weight = get_double();

    cout << endl;
    return *this;
}

friend bool operator==(const Animal& lhs, const Animal& rhs)
{
    if (lhs.height == rhs.height && lhs.weight == rhs.weight) {
        return true;
    }

    return false;
}
```

```
    friend bool operator<(const Animal& lhs, const Animal& rhs)
    {
        if (lhs.height < rhs.height && lhs.weight < rhs.weight) {
            return true;
        }

        return false;
    }

    friend bool operator>(const Animal& lhs, const Animal& rhs)
    {
        if (lhs.height > rhs.height && lhs.weight > rhs.weight) {
            return true;
        }

        return false;
    }
};

const double distance(Animal& obj, Animal& centroid)
{
    double x { obj.get_weight() - centroid.get_weight() };
    double y { obj.get_height() - centroid.get_height() };
    return sqrt(pow(x, 2) + pow(y, 2));
}

vector<Animal> clustering(vector<Animal>& animals, int count, int iterations)
{
    vector<Animal> centroids(count);
    vector<int> cluster_sizes(count);

    centroids[0] = animals[gen_random_int_in_range(0, animals.size() - 1)];
```

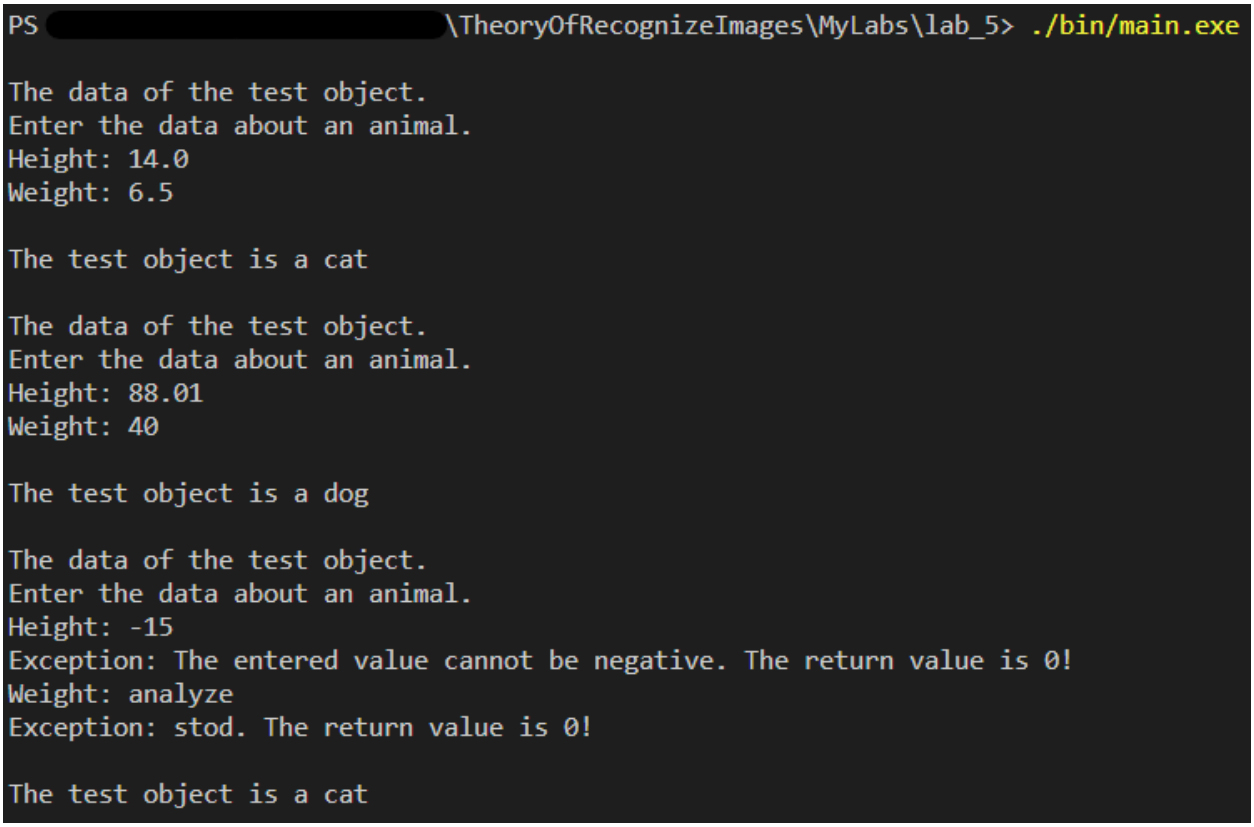
```
for (int i { 1 }; i < count; ++i) {
    centroids[i] = animals[gen_random_int_in_range(0, animals.size() - 1)];

    for (int j { 0 }; j < i; ++j) {
        if (centroids[j] == centroids[i]) {
            i--;
        }
    }
}

int iter {};
bool centroids_is_changed = true;

while (iter < iterations && centroids_is_changed) {
    centroids_is_changed = false;

    for (auto& animal : animals) {
        double min { DBL_MAX };
        for (int i {}; i < count; ++i) {
            double dist { distance(animal, centroids[i]) };
            if (min > dist) {
                min = dist;
                animal.set_cluster(i);
            }
        }

        cluster_sizes[animal.get_cluster()]++;
    }

    for (int i {}; i < count; ++i) {
        double height {};
```

```
            double weight {};

            for (auto& animal : animals) {
                if (animal.get_cluster() == i) {
                    height += animal.get_height();
                    weight += animal.get_weight();
                }
            }

            height /= cluster_sizes[i];
            weight /= cluster_sizes[i];

            if (height != centroids[i].get_height() || weight != centroids[i].get_weight()) {
                centroids_is_changed = true;
                centroids[i].set_height(height);
                centroids[i].set_weight(weight);
            }
        }

        fill(cluster_sizes.begin(), cluster_sizes.end(), 0);
        iter++;
    }

    sort(centroids.begin(), centroids.end());

    for (int i {}; i < count; i++) {
        centroids[i].set_cluster(i);
    }
    return centroids;
}

int main()
```

```
{
    thread go_away([]() {
        while (true) {
            this_thread::sleep_for(chrono::milliseconds(100));

            if (GetAsyncKeyState(VK_ESCAPE)) {
                exit(0);
            }
        }
    });

    vector<Animal> animals {
        // Cats
        Animal(20, 4),
        Animal(24, 4.2),
        Animal(22, 4.6),
        Animal(31, 5),
        Animal(27, 5.1),
        Animal(29, 4.4),

        // Dogs
        Animal(40, 6),
        Animal(60, 12),
        Animal(100, 20),
        Animal(55, 9),
        Animal(69, 15),
        Animal(36, 7),
    };

    vector<Animal> centroids(clustering(animals, 2, 100));

    while (true) {
```

```
        cout << endl;
        cout << "The data of the test object." << endl;
        Animal obj { Animal().from() };

        if (distance(obj, centroids[0]) < distance(obj, centroids[1])) {
            cout << "The test object is a cat" << endl;
        } else {
            cout << "The test object is a dog" << endl;
        }
    }

    return 0;
}
```

Знімки екрану



Висновок: Програма надає можливість дослідити та експериментувати з методоми кластерного аналізу для розпізнаванні образів. Вона сприяє кращому розумінню теорії розпізнавання образів та надає можливість провести практичне дослідження методів кластерного аналізу.