**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра ТВ**

**отчЁт**

**по лабораторной работе № 2**

**по дисциплине «Интеллектуальный анализ данных»**

**Тема: Кластеризация с помощью расстояния Махаланобиса**

|  |  |  |
| --- | --- | --- |
| Студенты гр. 9105 |  | Шаривзянов Д. Р. |
|  |  | Чугунов Р. |
| Преподаватель |  | Поздеев А. А. |

Санкт-Петербург

2024

КЛАСТЕРИЗАЦИЯ С ПОМОЩЬЮ РАССТОЯНИЯ МАХАЛАНОБИСА

**Цель работы:** моделирование алгоритма кластеризации с помощью расстояния Махаланобиса.

1. Код программы

#ifndef MAHALANOBIS\_H

#define MAHALANOBIS\_H

#include <iostream>

#include <fstream>

#include <sstream>

#include <string>

#include <vector>

#include <map>

#include <random>

#include <numeric>

#include <chrono>

#include <opencv2\opencv.hpp>

#include <opencv2\core\core.hpp>

#include "logger.hpp"

using namespace std;

using namespace cv;

struct IrisData {

    vector<double> features;  // Вектор признаков (длина чашелистика, ширина чашелистика, длина лепестка, ширина лепестка)

    string label;             // Метка класса: "Iris-setosa", "Iris-versicolor", "Iris-virginica"

};

/\*\*

 \* @brief Функция для загрузки данных из файла

 \*

 \* Функция принимает на вход путь к файлу и возвращает вектор структур IrisData,

 \* каждый элемент которого содержит вектор признаков (features) и метку класса (label)

 \*

 \* @param filename путь к файлу

 \* @return вектор структур IrisData

 \*/

vector<IrisData> loadData(const string& filename) {

    vector<IrisData> dataset;

    ifstream file(filename);

    string line;

    // Если файл не открыт, то выводим ошибку

    if (!file.is\_open()) {

        logger.error("Failed to open the file: {}", filename);

        return dataset;

    }

    // Читаем файл построчно

    while (getline(file, line)) {

        IrisData data;

        stringstream ss(line);

        string token;

        // Читаем токены в строке

        while (getline(ss, token, ',')) {

            // Если токен начинается с цифры, то добавляем его к вектору признаков

            if (isdigit(token[0]))

                data.features.push\_back(stod(token)); // Добавляем числовые данные в вектор признаков

            else

                data.label = token; // Последний токен — это метка класса

        }

        // Если вектор признаков не пустой и метка класса не пустая, то добавляем data к dataset

        if (!data.features.empty() && !data.label.empty())

            dataset.push\_back(data);

        else

            logger.warn("Empty data row or invalid format: {}", line);

    }

    logger.info("Loaded {} samples", dataset.size());

    return dataset;

}

/\*\*

 \* @brief Функция для разделения данных на обучающую (train) и тестовую (test) выборки

 \*

 \* @details

 \* Функция принимает на вход полный набор данных (dataset), а также

 \*   2 пустых контейнеров (map<string, vector<IrisData>>) для хранения

 \*   обучающей (trainData) и тестовой (testData) выборок.

 \*   Она разделяет данные на 2 выборки в соотношении train\_ratio : (1 - train\_ratio)

 \*   с помощью генератора случайных чисел.

 \*

 \* @param dataset полный набор данных

 \* @param trainData контейнер для хранения обучающей выборки

 \* @param testData контейнер для хранения тестовой выборки

 \* @param train\_ratio соотношение между обучающей и тестовой выборками

 \*/

void splitData(const vector<IrisData>& dataset, map<string, vector<IrisData>>& trainData, map<string, vector<IrisData>>& testData, double train\_ratio) {

    logger.info("Splitting data into train and test sets with ratio {}/{}", 100 - train\_ratio\*100, train\_ratio\*100);

    // Получаем текущее время в качестве seed

    unsigned seed = std::chrono::system\_clock::now().time\_since\_epoch().count();

    // Создаем генератор случайных чисел

    std::default\_random\_engine generator(seed);

    // Определяем распределение (например, равномерное от 1 до 100)

    std::uniform\_int\_distribution<int> distribution(0, 100);

    // Проходим по каждому элементу данных (data)

    for (const auto& data : dataset) {

        // Генерируем случайное число от 0 до 1

        double random\_number = (rand() / static\_cast<double>(RAND\_MAX));

        // double random\_number = (distribution(generator) / static\_cast<double>(100));

        // Если случайное число меньше заданного соотношения (train\_ratio), то добавляем данные к обучающей выборке (trainData), иначе добавляем данные к тестовой выборке (testData)

        if (random\_number <= train\_ratio)

            trainData[data.label].push\_back(data);

        else

            testData[data.label].push\_back(data);

    }

    // Подсчитываем количество элементов в обучающей выборке

    size\_t train\_size = accumulate(trainData.begin(), trainData.end(), size\_t(0), [](size\_t sum, const auto& pair) {

        // Суммируем количество элементов в каждом векторе

        return sum + pair.second.size();

    });

    // Подсчитываем количество элементов в тестовой выборке

    size\_t test\_size = accumulate(testData.begin(), testData.end(), size\_t(0), [](size\_t sum, const auto& pair) {

        // Суммируем количество элементов в каждом векторе

        return sum + pair.second.size();

    });

    logger.info("Splited data ({} samples) into training ({} samples) and testing ({} samples)", train\_size + test\_size, train\_size, test\_size);

}

/\*\*

 \* @brief Функция для расчета центроида

 \*

 \* @param[in] data Вектор структур данных, содержащих вектор признаков (features)

 \* @return Вектор средних значений признаков

 \*/

vector<double> calcCentroid(const vector<IrisData>& data) {

    // Определяем количество признаков

    int num\_features = data[0].features.size();

    // Создаем вектор для хранения средних значений

    vector<double> centroid(num\_features, 0.0);

    // Проходим по каждому элементу данных (data)

    for (const auto& sample : data)

        // Проходим по каждому признаку (feature)

        for (int i = 0; i < num\_features; ++i)

            // Складываем значения признака

            centroid[i] += sample.features[i];

    // Делим сумму значений признака на количество элементов данных

    for (double& val : centroid)

        val /= data.size();

    // Возвращаем расчитанный центроид

    return centroid;

}

/\*\*

 \* @brief Функция для логгирования ковариационной матрицы

 \*

 \* @param matrix Матрица, которую необходимо вывести построчно

 \*/

void logCovarianceMatrix(const Mat& matrix) {

    // Проходим по строкам матрицы

    for (int i = 0; i < matrix.rows; ++i) {

        stringstream ss;

        ss << "[";

        for (int j = 0; j < matrix.cols; ++j) {

            // Добавляем элемент матрицы в строку с разделителем запятая

            ss << matrix.at<double>(i, j);

            if (j < matrix.cols - 1)

                ss << ", ";

        }

        ss << "]";

        // Логируем каждую строку матрицы

        logger.info("{}", ss.str());

    }

}

/\*\*

 \* @brief Функция для вычисления ковариационной матрицы

 \*

 \* @param data Вектор структур данных, содержащих вектор признаков (features) и метку класса (label)

 \* @param centroid Вектор средних значений признаков

 \* @return Матрица ковариации размером (num\_features x num\_features)

 \*/

Mat calculateCovarianceMatrix(const vector<IrisData>& data, const vector<double>& centroid) {

    int num\_features = centroid.size();

    Mat covariance = Mat::zeros(num\_features, num\_features, CV\_64F);

    // Проходим по каждому элементу данных

    for (const auto& sample : data) {

        // Создаем матрицу разности между текущим элементом данных и центроидом

        Mat diff = Mat(sample.features) - Mat(centroid);

        // Вычисляем произведение diff на транспонированную матрицу и добавляем к матрице ковариации

        covariance += diff \* diff.t();

    }

    // Делим матрицу на количество элементов данных

    covariance /= data.size();

    // Если матрица ковариации пуста, выводим ошибку

    if (covariance.empty())

        logger.error("Covariance matrix is empty.");

    else {

        // Выводим матрицу ковариации в лог

        logger.info("Covariance matrix {}x{}:", covariance.rows, covariance.cols);

        logCovarianceMatrix(covariance);

    }

    return covariance;

}

/\*\*

 \* @brief Функция для вычисления обратной матрицы

 \*

 \* @param matrix Матрица, для которой необходимо вычислить обратную

 \* @return Обратная матрица (матрица, умноженная на которую,

 \*          даст единичную матрицу)

 \*/

Mat calculateInverse(const Mat& matrix) {

    if (matrix.empty()) {

        logger.error("Matrix is empty, cannot calculate inverse.");

        return Mat();

    }

    Mat inv\_matrix = matrix.inv();

    if (inv\_matrix.empty())

        // Если матрица не может быть обращена, то выводим ошибку

        logger.error("Inversion of the matrix failed.");

    else

        // Если матрица может быть обращена, то выводим информацию о размере полученной матрицы

        logger.info("Matrix inversion successful. Size: {}x{}", inv\_matrix.rows, inv\_matrix.cols);

    return inv\_matrix;

}

/\*\*

 \* @brief Рассчитывает расстояние Махаланобиса между вектором признаков (sample) и центроидом (centroid)

 \*        с учетом ковариационной матрицы (cov\_inv)

 \*

 \* @param sample Вектор признаков

 \* @param centroid Центроид

 \* @param cov\_inv Обратная ковариационная матрица

 \* @return Расстояние Махаланобиса

 \*/

double mahalanobisDistance(const vector<double>& sample, const vector<double>& centroid, const Mat& cov\_inv) {

    // Создаем матрицу разности между вектором признаков (sample) и центроидом (centroid)

    Mat diff = Mat(sample) - Mat(centroid);

    // Если матрица разности пуста, то выводим ошибку

    if (diff.empty()) {

        logger.error("One of the matrices (diff or cov\_inv) is empty.");

        return -1.0;

    }

    // Создаем транспонированную матрицу разности

    Mat diff\_t = diff.t();

    // Вычисляем скалярное произведение матрицы разности (diff\_t) на

    // обратную ковариационную матрицу (cov\_inv) и на матрицу разности (diff)

    // и сохраняем результат в матрице result

    Mat result = diff\_t \* cov\_inv \* diff;

    // Если матрица result пуста, то выводим ошибку

    if (result.empty()) {

        logger.error("Resulting matrix is empty.");

        return -1.0;

    }

    // Вычисляем сумму квадратов элементов матрицы result

    double sum\_of\_squares = sum(result)[0];

    // Возвращаем квадратный корень из суммы квадратов

    return sqrt(sum\_of\_squares);

}

/\*\*

 \* @brief Функция классификации по расстоянию Махаланобиса

 \*

 \* @param sample Вектор признаков, который нужно классифицировать

 \* @param means Карта, где ключ - метка класса, а значение - Центроид признаков для этого класса

 \* @param cov\_inverses Карта, где ключ - метка класса, а значение - обратная ковариационная матрица для этого класса

 \* @return Метка класса, к которому принадлежит вектор признаков

 \*/

string classify(const IrisData& sample, const map<string, vector<double>>& means, const map<string, Mat>& cov\_inverses) {

    string best\_class;

    double min\_distance = numeric\_limits<double>::max();

    // Проходим по каждому классу

    for (const auto& [label, centroid] : means) {

        // Рассчитываем расстояние Махаланобиса между вектором признаков (sample)

        // и центроидом (centroid) для текущего класса

        double distance = mahalanobisDistance(sample.features, centroid, cov\_inverses.at(label));

        // Если расстояние меньше минимального, то обновляем минимальное расстояние

        // и запоминаем метку класса

        if (distance < min\_distance) {

            min\_distance = distance;

            best\_class = label;

        }

    }

    // Возвращаем метку класса, к которому принадлежит вектор признаков

    return best\_class;

}

// Основная программа

int lab2\_Mahalanobis() {

    logger.info("Lab 2: Mahalanobis clustering");

    logger.info("Attribute Information:");

    logger.info("   1. sepal length in cm");

    logger.info("   2. sepal width in cm");

    logger.info("   3. petal length in cm");

    logger.info("   4. petal width in cm");

    logger.info("Classes:");

    logger.info("   1. Iris Setosa");

    logger.info("   2. Iris Versicolour");

    logger.info("   3. Iris Virginica");

    // Шаг 1. Загрузка данных

    string iris\_path = "../../resources/iris db/iris.data";

    vector<IrisData> dataset = loadData(iris\_path);

    // Шаг 2. Разделение на обучающую (90%) и тестовую (10%) выборки

    float trained\_percent = 0.9;

    map<string, vector<IrisData>> trainData, testData;

    splitData(dataset, trainData, testData, trained\_percent);

    // Шаг 3. Оценка средних векторов и ковариационных матриц

    map<string, vector<double>> means;

    map<string, Mat> covariances, cov\_inverses;

    for (const auto& [label, data] : trainData) {

        means[label] = calcCentroid(data);

        logger.info("Centroid for class {}: {}", label, fmt::format("[{}]", fmt::join(means[label], ", ")));

        Mat covariance = calculateCovarianceMatrix(data, means[label]);

        covariances[label] = covariance;

        cov\_inverses[label] = covariance.inv();  // Обратная ковариационная матрица

    }

    // Шаг 4. Тестирование классификатора на тестовой выборке

    logger.info("Classifying test samples...");

    int correct\_predictions = 0, total\_predictions = 0;

    for (const auto& [label, test\_samples] : testData)

        for (const auto& sample : test\_samples) {

            string predicted\_label = classify(sample, means, cov\_inverses);

            logger.info("Test sample from class '{}' assigned to a cluster '{}'", label, predicted\_label);

            if (predicted\_label == label)

                correct\_predictions++;

            total\_predictions++;

        }

    // Шаг 5. Вывод точности классификации

    double accuracy = static\_cast<double>(correct\_predictions) / total\_predictions \* 100;

    logger.info("Accuracy: {:.2f}%", accuracy);

    return 0;

}

#endif MAHALANOBIS\_H

1. Описание экспериментов
2. Загрузка данных: всего было загружено 150 образцов с четырьмя признаками: длина чашелистика, ширина чашелистика, длина лепестка и ширина лепестка. Образцы были разделены на тренировочные и тестовые выборки в двух различных конфигурациях.
3. Эксперимент 1:

Разделение данных: Данные были разделены на 136 тренировочных образцов и 14 тестовых.

Центроиды классов:

Iris-setosa: [5.01, 3.42, 1.46, 0.24]

Iris-versicolor: [5.89, 2.76, 4.22, 1.31]

Iris-virginica: [6.63, 2.99, 5.57, 2.04]

Ковариационные матрицы: для каждого класса были вычислены ковариационные матрицы. Например, для Iris-setosa:

[0.118397, 0.101901, 0.0127705, 0.0108692]

[0.101901, 0.150186, 0.0112856, 0.0111]

[0.0127705, 0.0112856, 0.0292168, 0.00495247]

[0.0108692, 0.0111, 0.00495247, 0.00917157]

Результаты: Все 14 тестовых образцов были правильно классифицированы, что дало точность классификации 100%.

1. Эксперимент 2:

Разделение данных: В этом случае данные были разделены на 13 тренировочных образцов и 137 тестовых.

Центроиды классов:

Iris-setosa: [5.17, 3.6, 1.55, 0.3]

Iris-versicolor: [5.80, 2.88, 4.14, 1.38]

Iris-virginica: [7.6, 3.35, 6.35, 1.9]

Ковариационные матрицы: например, для Iris-versicolor:

[0.132, 0.026, 0.098, -0.002]

[0.026, 0.0096, 0.0108, 0.0016]

[0.098, 0.0108, 0.1384, 0.0088]

[-0.002, 0.0016, 0.0088, 0.0056]

Результаты: Точность модели значительно снизилась до 35.04%, так как многие образцы, особенно из класса Iris-setosa, были неверно отнесены к классу Iris-virginica.

1. Результат работы программы

Ниже приведён листинг лога при эксперименте отношения обучающей выборки к тестовой как 90/10.

[16.10.2024 01:02:21.470] [info] Lab 2: Mahalanobis clustering

[16.10.2024 01:02:21.470] [info] Attribute Information:

[16.10.2024 01:02:21.470] [info]    1. sepal length in cm

[16.10.2024 01:02:21.470] [info]    2. sepal width in cm

[16.10.2024 01:02:21.470] [info]    3. petal length in cm

[16.10.2024 01:02:21.470] [info]    4. petal width in cm

[16.10.2024 01:02:21.470] [info] Classes:

[16.10.2024 01:02:21.470] [info]    1. Iris Setosa

[16.10.2024 01:02:21.470] [info]    2. Iris Versicolour

[16.10.2024 01:02:21.471] [info]    3. Iris Virginica

[16.10.2024 01:02:21.471] [info] Loaded 150 samples

[16.10.2024 01:02:21.471] [info] Splitting data into train and test sets with ratio 10.000002384185791/89.99999761581421

[16.10.2024 01:02:21.471] [info] Splited data (150 samples) into training (136 samples) and testing (14 samples)

[16.10.2024 01:02:21.471] [info] Centroid for class Iris-setosa: [5.010638297872339, 3.421276595744682, 1.4595744680851062, 0.2382978723404255]

[16.10.2024 01:02:21.472] [info] Covariance matrix 4x4:

[16.10.2024 01:02:21.472] [info] [0.118397, 0.101901, 0.0127705, 0.0108692]

[16.10.2024 01:02:21.472] [info] [0.101901, 0.150186, 0.0112856, 0.0111]

[16.10.2024 01:02:21.472] [info] [0.0127705, 0.0112856, 0.0292168, 0.00495247]

[16.10.2024 01:02:21.472] [info] [0.0108692, 0.0111, 0.00495247, 0.00917157]

[16.10.2024 01:02:21.472] [info] Centroid for class Iris-versicolor: [5.886666666666665, 2.755555555555556, 4.224444444444444, 1.3111111111111111]

[16.10.2024 01:02:21.473] [info] Covariance matrix 4x4:

[16.10.2024 01:02:21.473] [info] [0.236267, 0.0700741, 0.164993, 0.0481481]

[16.10.2024 01:02:21.473] [info] [0.0700741, 0.0949136, 0.0755309, 0.0402716]

[16.10.2024 01:02:21.473] [info] [0.164993, 0.0755309, 0.213402, 0.0695062]

[16.10.2024 01:02:21.473] [info] [0.0481481, 0.0402716, 0.0695062, 0.038321]

[16.10.2024 01:02:21.473] [info] Centroid for class Iris-virginica: [6.627272727272725, 2.9909090909090903, 5.5681818181818175, 2.038636363636363]

[16.10.2024 01:02:21.473] [info] Covariance matrix 4x4:

[16.10.2024 01:02:21.473] [info] [0.427893, 0.0997934, 0.325186, 0.0516736]

[16.10.2024 01:02:21.474] [info] [0.0997934, 0.0990083, 0.0728926, 0.0448967]

[16.10.2024 01:02:21.474] [info] [0.325186, 0.0728926, 0.325806, 0.0521384]

[16.10.2024 01:02:21.474] [info] [0.0516736, 0.0448967, 0.0521384, 0.0764618]

[16.10.2024 01:02:21.474] [info] Clustering test samples...

[16.10.2024 01:02:21.474] [info] Test sample from class 'Iris-setosa' assigned to a cluster 'Iris-setosa'

[16.10.2024 01:02:21.474] [info] Test sample from class 'Iris-setosa' assigned to a cluster 'Iris-setosa'

[16.10.2024 01:02:21.474] [info] Test sample from class 'Iris-setosa' assigned to a cluster 'Iris-setosa'

[16.10.2024 01:02:21.474] [info] Test sample from class 'Iris-versicolor' assigned to a cluster 'Iris-versicolor'

[16.10.2024 01:02:21.475] [info] Test sample from class 'Iris-versicolor' assigned to a cluster 'Iris-versicolor'

[16.10.2024 01:02:21.475] [info] Test sample from class 'Iris-versicolor' assigned to a cluster 'Iris-versicolor'

[16.10.2024 01:02:21.475] [info] Test sample from class 'Iris-versicolor' assigned to a cluster 'Iris-versicolor'

[16.10.2024 01:02:21.475] [info] Test sample from class 'Iris-versicolor' assigned to a cluster 'Iris-versicolor'

[16.10.2024 01:02:21.475] [info] Test sample from class 'Iris-virginica' assigned to a cluster 'Iris-virginica'

[16.10.2024 01:02:21.475] [info] Test sample from class 'Iris-virginica' assigned to a cluster 'Iris-virginica'

[16.10.2024 01:02:21.475] [info] Test sample from class 'Iris-virginica' assigned to a cluster 'Iris-virginica'

[16.10.2024 01:02:21.476] [info] Test sample from class 'Iris-virginica' assigned to a cluster 'Iris-virginica'

[16.10.2024 01:02:21.476] [info] Test sample from class 'Iris-virginica' assigned to a cluster 'Iris-virginica'

[16.10.2024 01:02:21.476] [info] Test sample from class 'Iris-virginica' assigned to a cluster 'Iris-virginica'

[16.10.2024 01:02:21.476] [info] Accuracy: 100.00%

**Вывод:** в ходе работы был реализован алгоритм кластеризации данных с использованием расстояния Махаланобиса. Было проведено два эксперимента с разбитием выборки на обучающую и тестовую выборки:  
1) В первом эксперименте с достаточным количеством тренировочных данных (136) модель показала высокую точность классификации (100%).

2) Однако во втором эксперименте с ограниченным объемом тренировочной выборки (13 образцов) точность резко упала до 35.04%. Это подтверждает важность объема тренировочных данных для качественной кластеризации на основе расстояния Махаланобиса.