ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № Х

«Название лабораторной работы (Вашего задания)»

Выполнил работу

Дубовик Никита

Академическая группа J3110

Принято

Ментор, Владислав Вершинин

Санкт-Петербург

2024

**Введение:**

Цель работы – Решить NP полную задачу определения наиболее оптимальных порогов бинаризации каждого признака.

Задачи работы:

Установить нужные для работы библиотеки;

Реализовать собственный алгоритм перебора всех возможных бинаризаций признаков.

Проанализировать полученный результат, оценить время работы и количество затраченной памяти;

Построить график зависимости времени от числа признаков;

**Теоретическая подготовка:**

Для выполнения данной работы нужно иметь представление о матрицах и векторах из библиотеки Armadillo, а также о методах работы с ними. Необходимо знать определения бинаризации по порогу.

**Реализация:**

1. Установка нужных библиотек

С официального сайта библиотеки mlpack по данной там инструкции мною была установлена библиотека mlpack и нужные зависимости.

1. Реализаций алгоритма

Для оптимизации перед перебором были заранее просчитанные все возможные бинаризации для каждого признака:

std::vector<arma::mat> all\_features\_binarizations;

all\_features\_binarizations.reserve(features.n\_rows);

for (size\_t i = 0; i < features.n\_rows; ++i) {

all\_features\_binarizations.emplace\_back(5, features.n\_cols);

const double min\_value = features.row(i).min();

const double max\_value = features.row(i).max();

const double step = (max\_value - min\_value) / 6.0;

for (uint8\_t threshold\_i = 1; threshold\_i <= 5; ++threshold\_i) {

const double threshold = min\_value + threshold\_i \* step;

all\_features\_binarizations[i].row(threshold\_i - 1) = arma::conv\_to<arma::rowvec>::from(features.row(i) >= threshold);

}

}

Далее был произведён перебор всех возможных комбинаций бинаризованных признаков и вычислен оптимальный скор:

double best\_score = std::numeric\_limits<float>::max();

double result\_score = 0.0;

std::vector<size\_t> current(features.n\_rows, 0);

arma::mat testing\_dataset(features.n\_rows, features.n\_cols);

while (true) {

for (size\_t j = 0; j < features.n\_rows; ++j) {

testing\_dataset.row(j) = all\_features\_binarizations[j].row(current[j]);

}

result\_score = evaluate\_dataset(testing\_dataset, target\_row);

if (result\_score < best\_score) {

best\_score = result\_score;

}

long long i = features.n\_rows - 1;

while (i >= 0) {

if (current[i] < 4) {

current[i]++;

break;

}

else {

current[i] = 1;

i--;

}

}

if (i < 0) {

break;

}

}

return std::pair<float, float> (best\_score, result\_score);

1. Получение результатов

Написанная функция возвращает лучший из всех рассчитанных скоров и последний полученный скор. Для оценки скорости работы производился замер времени выполнения (в секундах) с помощью функций стандартной библиотеки chrono (C++ 11), которая используется для работы со временем и временными интервалами:

const auto start\_time = std::chrono::high\_resolution\_clock::now();

// Код алгоритма

const std::chrono::duration<double> duration = std::chrono::high\_resolution\_clock::now() - start\_time;

Алгоритм запускался для разного количества признаков из выданного датасета для оценки скорости работы при разном количестве признаков.

1. Анализ алгоритма и полученных результатов

После получения результатов и проверки их на корректность была произведена асимптотическая оценка скорости работы алгоритма и используемой им памяти. Полученные теоретические оценки времени выполнения алгоритма были сопоставлены с фактическими измерениями времени. Был построек график зависимости фактического времени выполнения от количества обрабатываемых признаков.

**Экспериментальная часть:**

Подсчёт по памяти:

std::pair<float, float> get\_best\_binarization\_scores(const arma::mat& features, const arma::subview\_row<double>& target\_row) noexcept {

std::vector<arma::mat> all\_features\_binarizations;

all\_features\_binarizations.reserve(features.n\_rows);

for (size\_t i = 0; i < features.n\_rows; ++i) {

all\_features\_binarizations.emplace\_back(5, features.n\_cols);

const double min\_value = features.row(i).min();

const double max\_value = features.row(i).max();

const double step = (max\_value - min\_value) / 6.0;

for (uint8\_t threshold\_i = 1; threshold\_i <= 5; ++threshold\_i) {

const double threshold = min\_value + threshold\_i \* step;

all\_features\_binarizations[i].row(threshold\_i - 1) = arma::conv\_to<arma::rowvec>::from(features.row(i) >= threshold);

}

}

// all\_features\_binarizations будет хранить 11 \* 5 \* 1143 \* 8 байт (обобщённо n\_features\_rows \* n\_thresholds \* n\_features\_cols \* 8 байт)

double best\_score = std::numeric\_limits<float>::max();

double result\_score = 0.0;

std::vector<size\_t> current(features.n\_rows, 0); // 8 \* 11 байт (обобщённо n\_features\_rows \* 8 байт)

arma::mat testing\_dataset(features.n\_rows, features.n\_cols); // 11 \* 1143 \* 8 байт(обобщённо n\_features\_rows \* n\_features\_cols \* 8 байт) резервируем память

const auto start\_time = std::chrono::high\_resolution\_clock::now();

while (true) {

for (size\_t j = 0; j < features.n\_rows; ++j) {

testing\_dataset.row(j) = all\_features\_binarizations[j].row(current[j]);

}

result\_score = evaluate\_dataset(testing\_dataset, target\_row);

if (result\_score < best\_score) {

best\_score = result\_score;

}

long long i = features.n\_rows - 1;

while (i >= 0) {

if (current[i] < 4) {

current[i]++;

break;

}

else {

current[i] = 1;

i--;

}

}

if (i < 0) {

break;

}

}

const std::chrono::duration<double> duration = std::chrono::high\_resolution\_clock::now() - start\_time;

std::cout << "Execution Time: " << duration.count() << " seconds\n";

// Суммарная Space complexity составляет n\_features\_rows \* (n\_thresholds \* n\_features\_cols + 1 + n\_features\_cols)) \* 8 байт

return std::pair<float, float> (best\_score, result\_score); // Возвращает 16 байт

}

Подсчёт временной асимптотики:

std::pair<float, float> get\_best\_binarization\_scores(const arma::mat& features, const arma::subview\_row<double>& target\_row) noexcept {

std::vector<arma::mat> all\_features\_binarizations;

all\_features\_binarizations.reserve(features.n\_rows); // O(n\_features\_rows)

for (size\_t i = 0; i < features.n\_rows; ++i) { // O(K \* n\_features\_cols \* n\_features\_rows)

all\_features\_binarizations.emplace\_back(5, features.n\_cols); // O(K \* n\_features\_cols)

const double min\_value = features.row(i).min(); // O(n\_features\_cols)

const double max\_value = features.row(i).max(); // O(n\_features\_cols)

// К сожалению min\_max нету

const double step = (max\_value - min\_value) / 6.0;

for (uint8\_t threshold\_i = 1; threshold\_i <= 5; ++threshold\_i) { // Cуммарно O(K \* n\_features\_cols)

const double threshold = min\_value + threshold\_i \* step;

all\_features\_binarizations[i].row(threshold\_i - 1) = arma::conv\_to<arma::rowvec>::from(features.row(i) >= threshold); // O(n\_features\_cols)

}

}

double best\_score = std::numeric\_limits<float>::max();

double result\_score = 0.0;

std::vector<size\_t> current(features.n\_rows, 0); // O(n\_features\_rows)

arma::mat testing\_dataset(features.n\_rows, features.n\_cols); // O(n\_features\_rows \* n\_features\_cols)

const auto start\_time = std::chrono::high\_resolution\_clock::now();

while (true) { // Суммарно O(K^n\_features\_rows \* (n\_features\_cols \* n\_features\_rows + O(LinearRegression)))

for (size\_t j = 0; j < features.n\_rows; ++j) { // O(n\_features\_cols \* n\_features\_rows)

testing\_dataset.row(j) = all\_features\_binarizations[j].row(current[j]); // O(n\_features\_cols)

}

result\_score = evaluate\_dataset(testing\_dataset, target\_row); // O(LinearRegression)

if (result\_score < best\_score) {

best\_score = result\_score;

}

long long i = features.n\_rows - 1;

while (i >= 0) { // Отдельно рассматривать этот цикл нет смысла.

if (current[i] < 4) {

current[i]++;

break;

}

else {

current[i] = 1;

i--;

}

}

if (i < 0) {

break;

}

}

const std::chrono::duration<double> duration = std::chrono::high\_resolution\_clock::now() - start\_time;

std::cout << "Execution Time: " << duration.count() << " seconds\n";

// Суммарная Time complexity составляет O(K^n\_features\_rows \* (n\_features\_cols \* n\_features\_rows + O(LinearRegression)))

return std::pair<float, float> (best\_score, result\_score);

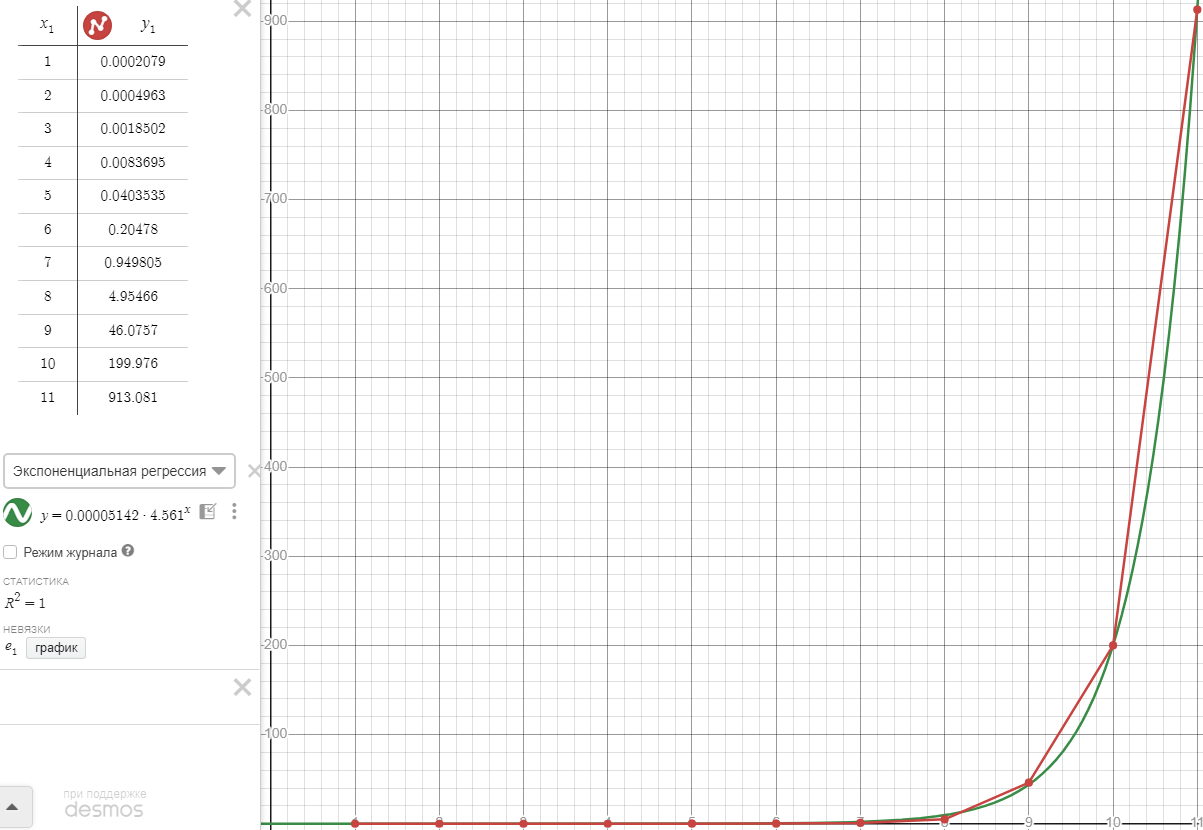
}

Согласно требованиям моего варианта, на вход к моему алгоритму подаётся датасет с признаками. Теоретически заданная сложность задачи составляет O(K^n\_features\_rows \* (n\_features\_cols \* n\_features\_rows \* O(LinearRegression)) и более. Для тестирования алгоритма была собрана статистика, приведенная в таблице №1 (с немного округлёнными значениями).

Таблица №1 - Подсчёт сложности реализованного алгоритма

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Размер входного набора признаков | 1 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| Время выполнения программы, с | 0.0002 | 0.04 | 0.2 | 0.9 | 4.95 | 46 | 200 | 913 |

График представляющий визуально удобный формат данных представлен на изображении №1.



Изображение №1 - График работы алгоритма

Экспоненциальная регрессия смогла с очень хорошей точностью аппроксимировать возможную функцию. На основании полученных экспериментальным путём данных можно подтвердить экспоненциальную природу зависимости времени выполнения данного алгоритма от количества входных данных.

**Заключение:**

В ходе выполнения работы мною был реализован алгоритм полного перебора и расчёта бинаризаций всех признаков изначального датасета. Цель работы была достигнута путём применения данного алгоритма к признакам выданного датасета. Полученные результаты также совпадают с теоретическими оценками сложности алгоритма.

В качестве дальнейших исследований можно предложить оптимизацию алгоритма с точки зрения уменьшения затрат использования памяти (если это критично), но жертвуя скоростью работы, а также рассмотреть распараллеливание полного перебора. Если идеальная точность не требуется, а скорость выполнения слишком большая, можно применить различные эвристические и жадные алгоритмы.

**Приложения:**

ПРИЛОЖЕНИЕ А

Исходный код файла main.cpp

#include <iostream>

#include <fstream>

#include <string>

#include <sstream>

#include <vector>

#define MLPACK\_PRINT\_INFO

#define MLPACK\_PRINT\_WARN

#include <mlpack.hpp>

#include <mlpack/methods/linear\_regression/linear\_regression.hpp>

#include "modelling.cpp"

std::pair<float, float> get\_best\_binarization\_scores(const arma::mat& features, const arma::subview\_row<double>& target\_row) noexcept {

std::vector<arma::mat> all\_features\_binarizations;

all\_features\_binarizations.reserve(features.n\_rows); // Выделяем память | O(n\_features\_rows)

for (size\_t i = 0; i < features.n\_rows; ++i) { // O(K \* n\_features\_cols \* n\_features\_rows)

all\_features\_binarizations.emplace\_back(5, features.n\_cols); // Резервируем память | O(K \* n\_features\_cols)

const double min\_value = features.row(i).min(); // O(n\_features\_cols)

const double max\_value = features.row(i).max(); // O(n\_features\_cols)

// К сожалению min\_max нету

const double step = (max\_value - min\_value) / 6.0;

for (uint8\_t threshold\_i = 1; threshold\_i <= 5; ++threshold\_i) { // Cуммарно O(K \* n\_features\_cols)

const double threshold = min\_value + threshold\_i \* step;

all\_features\_binarizations[i].row(threshold\_i - 1) = arma::conv\_to<arma::rowvec>::from(features.row(i) >= threshold); // O(n\_features\_cols)

}

}

// all\_features\_binarizations будет хранить 11 \* 5 \* 1143 \* 8 байт (обобщённо n\_features\_rows \* n\_thresholds \* n\_features\_cols \* 8 байт)

double best\_score = std::numeric\_limits<float>::max();

double result\_score = 0.0;

std::vector<size\_t> current(features.n\_rows, 0); // 8 \* 11 байт (обобщённо n\_features\_rows \* 8 байт) | O(n\_features\_rows)

arma::mat testing\_dataset(features.n\_rows, features.n\_cols); // 11 \* 1143 \* 8 байт(обобщённо n\_features\_rows \* n\_features\_cols \* 8 байт) резервируем память | O(n\_features\_rows \* n\_features\_cols)

const auto start\_time = std::chrono::high\_resolution\_clock::now();

while (true) { // Суммарно O(K^n\_features\_rows \* (n\_features\_cols \* n\_features\_rows + O(LinearRegression)))

for (size\_t j = 0; j < features.n\_rows; ++j) { // O(n\_features\_cols \* n\_features\_rows)

testing\_dataset.row(j) = all\_features\_binarizations[j].row(current[j]); // O(n\_features\_cols)

}

result\_score = evaluate\_dataset(testing\_dataset, target\_row); // O(LinearRegression)

if (result\_score < best\_score) {

best\_score = result\_score;

}

long long i = features.n\_rows - 1;

while (i >= 0) { // Отдельно рассматривать этот цикл нет смысла.

if (current[i] < 4) {

current[i]++;

break;

}

else {

current[i] = 1;

i--;

}

}

if (i < 0) {

break;

}

}

const std::chrono::duration<double> duration = std::chrono::high\_resolution\_clock::now() - start\_time;

std::cout << "Execution Time: " << duration.count() << " seconds\n";

// Суммарная Space complexity составляет n\_features\_rows \* (n\_thresholds \* n\_features\_cols + 1 + n\_features\_cols)) \* 8 байт

// Суммарная Time complexity составляет O(K^n\_features\_rows \* (n\_features\_cols \* n\_features\_rows + O(LinearRegression)))

return std::pair<float, float> (best\_score, result\_score); // Возвращает 16 байт

}

inline void run\_tests() {

arma::mat dataset;

if (!mlpack::data::Load("../data/WineQT.csv", dataset)) { // Загружаем датасет 13 \* 1143 \* 8 байт (обобщённо n\_rows \* n\_cols \* 8 байт) | O(n\_cols \* n\_rows)

throw std::runtime\_error("Could not read \*.csv!");

}

else {

std::pair<float, float> scores = get\_best\_binarization\_scores(dataset.submat(0, 0, dataset.n\_rows - 3, dataset.n\_cols - 1), dataset.row(11)); // Получили 16 байт // -3

std::cout << scores.first << "\n" << scores.second << "\n";

}

}

int main() {

run\_tests();

}

ПРИЛОЖЕНИЕ B

Исходный код изменённого файла modelling.cpp

#include <array>

#include <vector>

#include <algorithm>

#define MLPACK\_PRINT\_INFO

#define MLPACK\_PRINT\_WARN

#include <mlpack.hpp>

/\*\*

\* @brief Функция обучения модели МО (черного ящика), возвращающая метрику качества

\*

\* Функция принимает на вход набор данных, и индексы столбцов целевого и индекс столбцов.

\* Главые действия: удалить индекс столбец, если он есть; извлечь целевой столбец, обучить модель

\* и вернуть метрику качества

\*

\* @param[in] dataset Набор данных

\* @param[in] target\_column\_index Индекс целевого столбца (считать в датасете с индекса 0)

\* @param[in] id\_column\_index Индекс id или index столбца если он есть, который не важен модели при обучении

\* @return Метрика качества RMSE

\*/

inline double evaluate\_dataset(const arma::mat& features, const arma::subview\_row<double>& target) {

// Создать объект модели

mlpack::regression::LinearRegression lr;

// Обучить модель и получить метрику качества

const double rmse = lr.Train(features, target);

return rmse;

}