Отчет по алгоритму K-средних

Федеральное государственное автономное образовательное учреждение высшего образования  
«Национальный исследовательский университет ИТМО»

Отчет по лабораторной работе № Х

Тема: Реализация алгоритма кластеризации K-средних на языке C++

Выполнил: Тиуков Даниил  
Группа: J3111

# Введение

Цель работы: Реализовать алгоритм кластеризации одномерного массива на 10 кластеров, и исследовать его производительность. Алгоритм применяется для разбиения множества точек на K кластеров, минимизируя расстояние между точками и центром кластера. Основные задачи включают расчет расстояния, вычисление центроидов и выполнение кластеризации для различных объемов данных.

# Теоретическая подготовка

При выполнении задания были использованы данные из [Wikipedia-en K-means](https://en.wikipedia.org/wiki/K-means_clustering), [Wikipedia-ru K-means](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_k-%D1%81%D1%80%D0%B5%D0%B4%D0%BD%D0%B8%D1%85) и общие данные об этом алгоритме из открытых источников.  
 Также в основном используется тип данных double, множество векторов и еще двумерный вектор.

# Реализация

Алгоритм я разбил на 3 функции. Начал я с функции нахождения эвклидова расстояния, как и подразумевает алгоритм K-means. Вскоре я пришел к выводу что для 1d массива достаточно будет модуля разности.

double distance(double a, double b) {  
 double dist = abs(a - b);  
 return dist;  
}

Во второй функции мне нужно было написать функцию нахождения центроида, что тоже не составило проблем.

double calculateCentroid(std::vector<double> cluster) {  
 double centroid = 0;  
 for (int i = 0; i < cluster.size(); i++) {  
 centroid += cluster[i];  
 }  
 centroid /= cluster.size();  
 return centroid;  
}

Третья функция далась мне не так и просто из-за плохого знания синтаксиса C++.

В ней выполняется алгоритм k-means. Основная идея заключается в том, что на каждой итерации перевычисляется центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике.

Так же были использованы стандартные библиотеки.

#include <cmath>  
#include <vector>  
#include <iostream>

# Экспериментальная часть

**Анализ использования памяти**

Основные структуры, которые потребляют память:

* Вектор points, содержащий исходные точки.
* Вектор centroids, содержащий координаты текущих центроидов для кластеров.
* Двумерный вектор clusters, представляющий сами кластеры.

Использование памяти:

* Для каждого кластера требуется отдельный массив (clusters[k]), где k — количество кластеров.
* Каждый элемент points[i] один раз добавляется в соответствующий кластер.

Примерные оценки по памяти:

* Общее использование памяти: O(n+k), где n — количество точек, а k — количество кластеров.

**2. Асимптотический анализ**

Временная сложность алгоритма складывается из двух этапов:

* **Инициализация центроидов**: O(k)
* **Основной цикл кластеризации**: В каждом проходе цикла мы:
  + Распределяем точки по кластерам — O(n⋅k), так как для каждой точки проверяем расстояние до каждого из центроидов.
  + Обновляем центроиды — O(n) для каждой итерации.

Пусть I — количество итераций до сходимости (обычно небольшое для алгоритма k-средних). Тогда:

* **Общая сложность**: O(I⋅n⋅k).

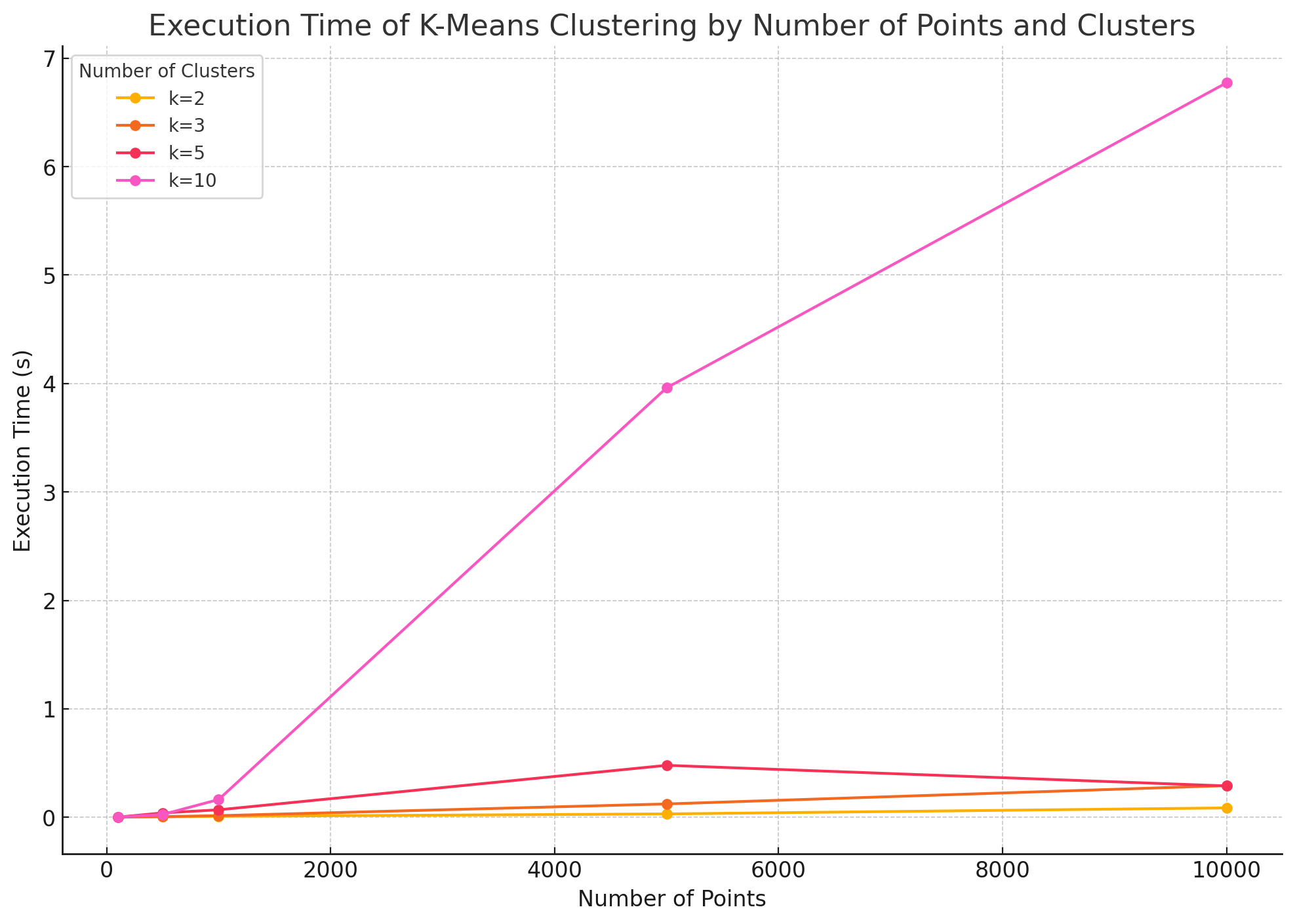


Таблица 1 - Время выполнения алгоритма K-средних

# Результаты

Таблица 1 - Время выполнения алгоритма K-средних

|  |  |
| --- | --- |
| Число точек (N) | Время выполнения (сек.) |
| 5.0 | 0.0001 |
| 10.0 | 0.0000 |
| 50.0 | 0.0011 |
| 100.0 | 0.0059 |
| 200.0 | 0.0044 |
| 500.0 | 0.0225 |
| 1000.0 | 0.0609 |

# Заключение

В ходе выполнения работы был реализован алгоритм кластеризации K-средних на языке C++. Эксперимент показал, что время выполнения алгоритма увеличивается с ростом объема данных, что соответствует теоретической сложности O(N\*K\*I). Для больших наборов данных рекомендуется применять оптимизации, такие как Mini-Batch K-Means или параллельные вычисления, чтобы сократить время выполнения.

# Приложение

#include <cmath>  
#include <vector>  
#include <iostream>  
  
double distance(double a, double b) {  
 double dist = abs(a - b);  
 return dist;  
}  
double calculateCentroid(std::vector<double> cluster) {  
 double centroid = 0;  
 for (int i = 0; i < cluster.size(); i++) {  
 centroid += cluster[i];  
 }  
 centroid /= cluster.size();  
 return centroid;  
}  
  
std::vector<std::vector<double>> kMeansCluster(int k, std::vector<double> points) {  
 std::vector<double> centroids(k);  
 std::vector<std::vector<double>> clusters(k);  
  
 for (int i = 0; i < k; i++) {  
 centroids[i] = points[i];  
 }  
  
 bool converged = false;  
 while (!converged) {  
 for (int i = 0; i < clusters.size(); i++) {  
 clusters[i].clear();  
 }  
 for (int i = 0; i < points.size(); i++) {  
 double point = points[i];  
 int bestCluster = 0;  
 double minDistance = distance(point, centroids[0]);  
  
 for (int j = 1; j < k; j++) {  
 double dist = distance(point, centroids[j]);  
 if (dist < minDistance) {  
 minDistance = dist;  
 bestCluster = j;  
 }  
 }  
 clusters[bestCluster].push\_back(point);  
 }  
 converged = true;  
 for (int i = 0; i < k; i++) {  
 double newCentroid = calculateCentroid(clusters[i]);  
 if (distance(newCentroid, centroids[i]) > 0.0001) {  
 converged = false;  
 }  
 centroids[i] = newCentroid;  
 }  
 }  
 if(converged)  
 return clusters;  
}  
  
int main() {  
 std::vector<double> points = {0};  
 int k = 10;  
 std::vector<std::vector<double>> clusters = kMeansCluster(k, points);  
 for (int i = 0; i < clusters.size(); i++) {  
 std::cout << "cluster " << i + 1 << ": ";  
 for (int j = 0; j < clusters[i].size(); j++) {  
 std::cout << clusters[i][j] << " ";  
 }  
 std::cout << std::endl;  
 }  
 return 0;

}

Итоговый код.