Lab1 - Curse of Dimensionality

Cesar Lengua Málaga

August 28, 2024

1. Introduction

This task consists of implementing a C++ program which generates random points in k-dimensions and then calculating the Euclidian distance between these points. For the tests, 100 points were used in different dimensions 10,50,100,500,1000,2000,5000.

2. Implementation

The program ”LAB1 - Distance between k-dimensional pointsçovers two functions, which I will detail in a moment.

2.1. Double vector function

This function contains two parameters: The first parameter **numPoints** that uses a **const int** because even though the dimension is different the number of points does not change and the second parameter which is the **dimension** that we are going to pass to it to do the tests. To create the random numbers we use some functions given in the laboratory document **UniformRealDistribution**.

Inside the function we create a double vector and pass these two parameters and creating a double vector in the dimension and we make two loops to save the points in the vector and then we return it.

```cpp

vector<vector<double>> createPoints(const int numPoints, int dimension){

    random_device rd;  // Will be used to obtain a seed for the random number engine
    mt19937 gen(rd()); // Standard mersenne_twister_engine seeded with rd()
    uniform_real_distribution<> dis(0.0, 1.0);

    vector<vector<double>> points(numPoints, vector<double> (dimension));

    for(int i=0; i<numPoints; i++){
        for(int j=0; j<dimension;j++){
            points[i][j] = dis(gen);
        }
    }
    return points;
}
```

Codigo 1: createPoints function

2.2. Euclidian Distance function

This function is very simple, what we do with this function is pass it two parameters that are **two vectors** to be able to calculate the **distance** of the **points**, we apply the **Euclidian Distance** formula and return the distance.

```
double euclidianDistance(vector<double>& p1, vector<double>& p2){
    double sum = 0;
    for(int i=0; i<p1.size(); i++){
        sum += pow(p1[i] - p2[i], 2);
    }

    return sqrt(sum);
}
```

Codigo 2: euclidianDistance function

2.3. Main

In the main we create an integer that is for the dimension and we change it manually for the different dimensions, additionally we create a double vector for the points and a vector for the distance, after that we create two nested loops and pass the **euclidianDistance** function to it.

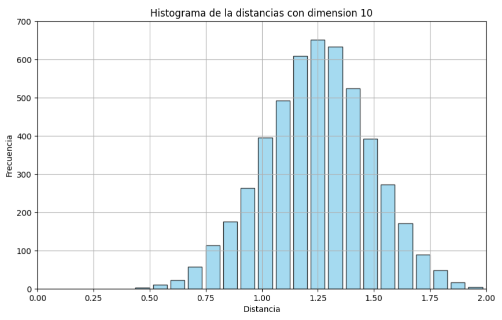
I realized that doing cout did no generate the 4500 distances in the terminal so I had to do a **pushback** to the distances vector and then save it in a file called **distances.txt**

```
int main(){

    //vector<int> dimensiones = {10,50,100,500,1000,2000,5000};
    int d = 5000;

    vector<vector<double>> points = createPoints(100,d);
    vector<double> distances;

    for(int i=0; i<points.size();i++){
        for(int j=i+1; j<points.size(); j++){
            //cout << euclidianDistance(points[i], points[j]) << endl;
            distances.push_back(euclidianDistance(points[i],points[j]));
        }
    }

    ofstream file("distancias.txt");
    for (const auto& distancia : distances) {
        file << distancia << endl;
    }
    file.close();

    return 0;
}
```

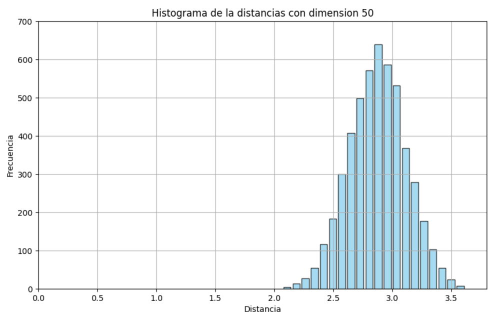
Codigo 3: main

3. Results

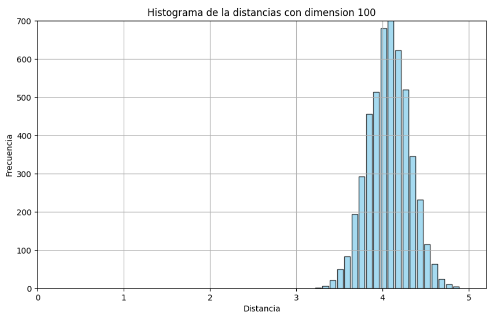
After carrying out the tests with the requested dimensions, the following results were obtained, which will now be represented in the form of histograms.



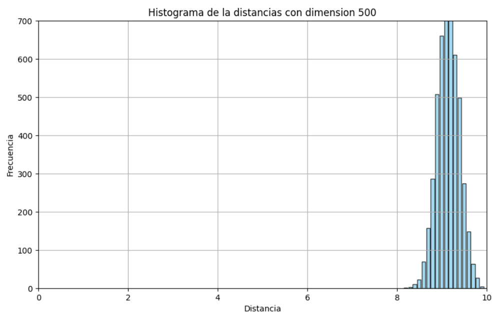
(a) dimension 10



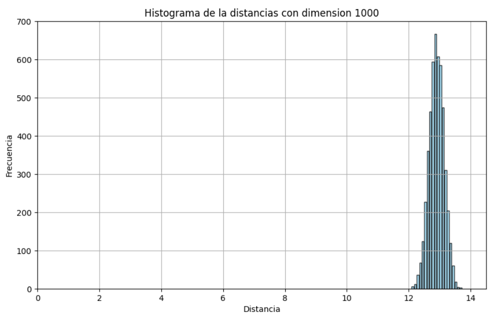
(b) dimension 50



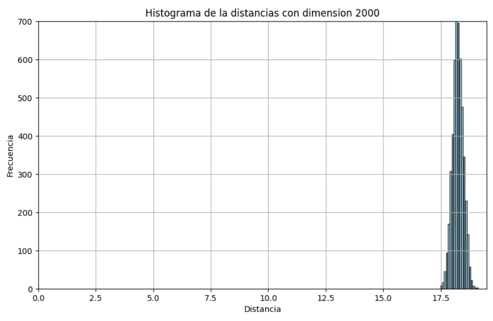
(c) dimension 100



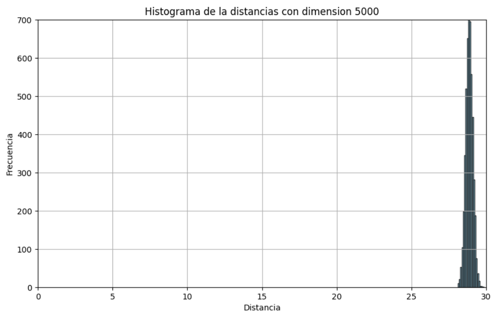
(d) dimension 500



(e) dimension 1000



(f) dimension 2000



(g) dimension 5000

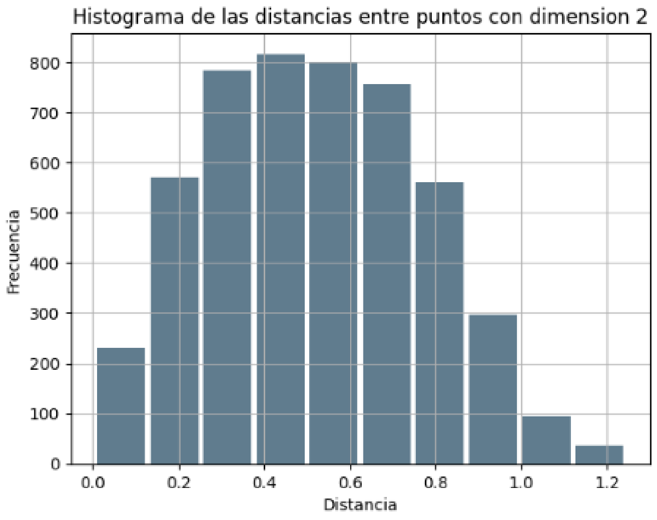
After analyzing the histograms we realize a behavior, as the number of dimensions for each point increases, the distance between them also increases exponentially.

3.1. Analysis

We see that the minimum distance between the points is further away from zero.

This has an explanation because if we take the case that it was a single dimension, the maximum distance that the points can take is the difference between the highest value that a point can take and the lowest value. In the case of two dimensions we use the Euclidean distance with the greatest possible distance that the points can take $\sqrt{(1)+(1)}$ so the operation gives us 1.414213 and we see that the maximum value in the figure of dimension 2.

That is as an example in the laboratory.



Taking into account that we have 100 points of k dimensions and each dimension can take values between 0 and 1 with 4 decimal figures $0{,}0001^{k*100}$, thanks to this we see why it is separated from 0, noting that the probability that all the points have the same position is very low since the same formula is applied.

4. Github

- https://github.com/rxsh/EDA/tree/main/Lab1