PhyDBSCAN Project Report

Nadia Tahiri, PhD
Thibaut Leval, Trainee bioinformatics researcher - 2023

Université de Sherbrooke

Table des matières

Introduction

In this phyDBSCAN project’s report, we give a summary of the project's goals, scope, and problem encountered. We also include the project's difficulties.

Project Overview

The C++ phyDBSCAN project, which employs the DBSCAN algorithm for tree classification based on distances between data points, is described in full in this article. Through innovative methods, the project seeks to improve phylogenetic tree construction by considering only distances between points (without points coordinates), automatically determining the best value for epsilon and minPoints.

For detailed technical information about the functions used, you can find direct references in the code itself. Each function is documented within its corresponding header file.
Each description is presented as follows:

```
/**
 * @brief …
 *
 * @param …
 * @return …
 */
```

To find out how to use the program, compile and run it, and to see an example of its use, consult the “README.md” file on github.

Access the project (github URL): https://github.com/tahiri-lab/phyDBSCAN

Implementation and Functionality

Program architecture

The architecture of the phyDBSCAN project is designed to facilitate readability, and maintainability. This section provides an overview of the project's organization and the key components that contribute to its structure.

The project directory is organized as follows:

```
phyDBSCAN/
|-- include/
|   |-- FileIO.h
|   |-- Clustering.h
|   |-- Point.h
|   |-- HyperparametersCalculator.h
|-- resources/
|   |-- input_data.txt - file to be filled in with the dataset to be processed
|-- src/
|   |-- poc.cpp - Proof of Concept file containing calls to functions to run the program, it's the
main entry point
|   |-- FileIO.cpp - Implementation of functions for reading and writing data to files
|   |-- Clustering.cpp - Implementation of the DBSCAN clustering algorithm and related
functionalities
|   |-- HyperparametersCalculator.cpp - Implementation of hyperparameter calculation functions
|   |-- ARI.cpp - Implementation of the Adjusted Rand Index calculation function
|-- CMakeLists.txt - to compile and launch the project on CLion
|-- MakeFile - to compile and launch the project
```

DBSCAN Algorithm on distance matrix and Clustering with Project constraints

The DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm has been employed in this project for the purpose of tree classification using distance matrices. Unlike traditional clustering methods that utilize coordinate-based points, the input dataset consists of an array of distances between points, forming a distance matrix.

The main functionality of the DBSCAN algorithm is orchestrated through the "poc.cpp" file. This file serves as the entry point of the application and coordinates the different steps of the algorithm, from data reading to results printing.

The "FileIO.cpp" file handles the reading of the input data from the "input_data.txt" file located in the resources directory (*file to be filled in with the dataset to be processed*). Each line of the input file corresponds to a point and contains a set of distance values to other points. These distance values are then stored in the Point objects.

The clustering process takes place in the “Clustering.cpp” file. Core points are identified based on the minPoints parameter, which represents the minimum number of neighbouring points required for a point to be considered core. The algorithm then recursively expands clusters by associating connected core points and assigning points on the borders accordingly. Once the core points research and clusters are determined, the determineBorderAndOutlier function labels border and outlier points. Border points are assigned to the group of a core point if they are within the epsilon radius, while outliers do not belong to any cluster and are labelled as such (corresponding number = -1).

Finally, the program displays the group assignments and point types (Core, Border or Outlier) for each point in the dataset.

Determining Optimal Epsilon

By analysing the input distance matrix, the algorithm calculates the average maximum difference in sorted distances for each data point (the number with the highest difference in value from the next number). This value serves as an estimation of an appropriate epsilon parameter, which defines the maximum distance for points to be considered part of the same cluster. This calculation is in the findBestEpsilon method, in “HyperparametersCalculator.cpp” file.

Adjusted Rand Index (ARI) calculation

The Adjusted Rand Index (ARI) is a metric used to evaluate the performance of clustering algorithms, including DBSCAN. The primary objective of ARI is to assess the similarity between the groups identified by the algorithm and the expected ground truth groups.

ARI permits to evaluate the quality of a clustering algorithm by comparing the groups found by the algorithm.

The ARI calculation involves comparing the assignments of points into clusters as determined by DBSCAN with the reference groupings. It evaluates the agreement between these two sets of assignments, considering both the similarities and differences. If the ARI is close to 1, the groups found by the algorithm are like the expected groups, it means the algorithm is good.

Areas for improvement

There are some areas that can be improved in the project, we will give more details here.

Find the best minPoints value using DSets-DBSCAN

In the pursuit of enhancing the DBSCAN algorithm's accuracy and adaptability to various datasets, this project delves into the optimization of the minPoints parameter selection, a critical factor in determining the minimum number of neighbouring points required for a point to be considered a core point within a dense region. The currently implemented method calculates the best minPoints value based on the average density of points within a pre-calculated epsilon distance for each data point. While functional, this approach could benefit from improvements.

After further research, I believe there is another, more suitable solution that has not yet been implemented due to lack of time. This would involve using DSets-DBSCAN

In the DSets-DBSCAN algorithm, the best minPts is determined automatically depending on the dataset. Here's how it works:

1. First, the DSets-DBSCAN algorithm uses the DSets algorithm to perform an initial clustering step. This generates dominant sets from the data.
2. Next, the DBSCAN algorithm is used to extend the clusters generated by DSets. However, instead of using a fixed minPts, DSets-DBSCAN automatically determines the optimal minPts for each cluster based on the dominant sets. In other words, for each cluster, DSets-DBSCAN calculates the local density of points within the corresponding dominant set. Using this local density, it determines the optimal minPts for that cluster (the mathematical part is explained in the DSets-DBSCAN article cited below).
3. Once the optimal minPts is determined for each cluster, the DBSCAN algorithm is used to extend the clusters using the corresponding minPts values.

In summary, DSets-DBSCAN uses the information from the dominant sets generated by DSets to automatically determine the optimal minPts for each cluster. This allows the minPts to be adapted to the specific characteristics of each cluster and the dataset.

Implementing DSets-DBSCAN could offers the potential to achieve greater precision in minPoints selection, leading to more accurate and reliable clustering results.

Source, complete article on the DSets-DBSCAN subject containing all the information and mathematical formulae: https://ieeexplore.ieee.org/document/7460951
Authors: Jian Hou; Huijun Gao; Xuelong Li

Visual representation

I worked on the matrix visual representation. I would have liked to represent points with different colours depending on their group and with the corresponding label while respecting distances between them. However, it's not easy because we don't have the points coordinates, it's possible to estimate them with some strategies but there is another problem. It is not possible to visually represent the points on a scatter plot because of the triangular inequality. In this case, RF is not Euclidean.
I think it would be a good idea to use matplotlib for the visual representation, but it's not possible to implement this feature now with the triangular inequality.

My research on this subject is on the branch “back-up-dbscan-visual-representation-poc” on the github project.

Pseudo-code description

This part of the report provides a better understanding of how the parameters calculation functions work.

Epsilon parameter calculation

The following section presents a pseudo-code description of the algorithm used to calculate the optimal epsilon value for the DBSCAN algorithm.

```
Input: points - A list of points with their distances to other points
Output: bestEpsilon - The calculated best epsilon value for DBSCAN

Function findBestEpsilon(points: vector of Point)

    Initialize distanceArray as an empty 2D list

    For each point in points
        Add point's distances to distanceArray

    Initialize numBeforeIncreaseList as an empty list

    For each distances in distanceArray
        sortedDistances = Sort(distances)

        maxDiff = 0.0
        numBeforeIncrease = 0.0

       // Explanation: The loop starts at index 2 to avoid comparing the point's distance with
       itself,
       // which is at index 0. We are interested in finding the difference between distances to
       other points
       // Starting at index 2 ensures that we consider differences between the current distance
       and the previous one
        For i = 2 to size of sortedDistances - 1
           // Calculate the difference between the current sorted distance and the previous one
            diff = sortedDistances[i] - sortedDistances[i - 1]

            // Check if the calculated difference is greater than the maximum difference seen so
              far
           // and if the previous distance was 0 (if it's 0 we do an average)
            If diff > maxDiff and sortedDistances[i - 1] = 0
                maxDiff = diff
                numBeforeIncrease = (sortedDistances[i] + sortedDistances[i - 1]) / 2
            // Update the maximum difference and store the previous distance as the
              numBeforeIncrease
           // numBeforeIncrease will be used to calculate an average and find epsilon
            Else if diff > maxDiff
                maxDiff = diff
                numBeforeIncrease = sortedDistances[i - 1]
        Append numBeforeIncrease to numBeforeIncreaseList

    totalNumBeforeIncrease = 0.0
    For each numBeforeIncrease in numBeforeIncreaseList
        totalNumBeforeIncrease += numBeforeIncrease

    bestEpsilon = totalNumBeforeIncrease / size of numBeforeIncreaseList

    Return bestEpsilon
End
```

MinPoints parameter calculation

The following section presents a pseudo-code description of the algorithm used to calculate the optimal minPts value for the DBSCAN algorithm. This should be improved with the DSets-DBSCAN algorithm explained earlier in this report.

```
Input: points - A list of points with their distances to other points
       epsilon - The epsilon value used to define the maximum distance for points in the same
cluster

Output: bestMinPoints - The calculated best minPoints value for DBSCAN

Function findBestMinPoints(points: vector of Point, double: epsilon value)

    Initialize minPoints as an empty list

    For each point in points
        count = 0
        For each distance in point.distances
            // Count the number of distances within the defined epsilon distance
            If distance <= epsilon
                Increment count

        Append count to minPoints

    totalMinPoints = 0
    For each count in minPoints
        totalMinPoints += count

    // Calculate the average minPoints value
    bestMinPoints = totalMinPoints / size of minPoints

    Return bestMinPoints
End
```

DBSCAN Clustering Algorithm

The provided pseudo-code offers an abstract depiction of the DBSCAN algorithm, showcasing its essential steps in density-based clustering using distance metrics between data points.

```
// Expands a cluster from a core point
function expandCluster(point, group, points, epsilon):
    point.group = group

    for each neighbor in points:
        distance = distance between point and neighbor
        if distance <= epsilon and neighbor.group == 0:
            neighbor.group = group
            if neighbor.type == CORE:
                expandCluster(neighbor, group, points, epsilon)


// Traverses the points to find "Core" points and determine their groups
function determineCoreAndGroup(points, epsilon, minPoints):
    groupCounter = 0

    for i = 0 to points.size() - 1:
        nbNeighbours = 0
        for j = 0 to points.size() - 1:
            if i != j:
                distance = points[i].distances[j]
                if distance <= epsilon:
                    nbNeighbours += 1
        if nbNeighbours >= minPoints:
            points[i].type = CORE
            if points[i].group == 0:
                groupCounter += 1
                expandCluster(points[i], groupCounter, points, epsilon)


// Determines the Border and Outlier point types
function determineBorderAndOutlier(points, epsilon):
    for i = 0 to points.size() - 1:
        if points[i].type != CORE:
            for j = 0 to points.size() - 1:
                if points[i].distances[j] <= epsilon and points[j].type == CORE:
                    points[i].type = BORDER
                    points[i].group = points[j].group
              break
            if not foundBorder:
                points[i].type = OUTLIER
                points[i].group = -1
```

Conclusion

In summary, this project successfully introduced an innovative approach to phylogenetic tree classification using the DBSCAN algorithm.
The project contains several important parts like optimal parameter calculation, core point identification, outlier handling, and cluster evaluation through the Adjusted Rand Index. While

the project marks a positive stride in enhancing accuracy, future work could focus on integrating advanced techniques like DSets-DBSCAN to refine parameter selection.

Contact

For inquiries, feedback, or additional information, please feel free to reach out to us:

- Nadia.Tahiri@USherbrooke.ca
- Thibaut.Leval@USherbrooke.ca or Linkedin