Implementation and Evaluation of DBSCAN Algorithm

Pedro Lindeza, Marko Raguž, Rafael Belchior

June 17, 2018

Data Ming Project at Warsaw University of Technology 2017/2018

1 Task Description

This task consists of the implementation, in Java, of the clustering algorithm DBSCAN. Furthermore, 3 internal and 3 external evaluation methods are implemented in C++.

1.1 DBSCAN Algorithm

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996. It is a density-based clustering algorithm: given a set of points in some space, it groups together points that are closely packed together (points with many nearby neighbors), marking as outliers points that lie alone in low-density regions (whose nearest neighbors are too far away). DBSCAN is one of the most common clustering algorithms and also most cited in scientific literature.[2]

2 Assumptions

Consider a set of points in some space to be clustered. For the purpose of DBSCAN clustering, the points are classified as core points, (density-)reachable points and outliers, as follows:

- A point p is a core point if at least minPts points are within distance e (e is the maximum radius of the neighborhood from p) of it (including p). Those points are said to be directly reachable from p.
- A point q is directly reachable from p if point q is within distance e from point p and p must be a core point.
- A point q is reachable from p if there is a path p1, ..., pn with p1 = p and pn = q, where each pi+1 is directly reachable from pi (all the points on the path must be core points, with the possible exception of q).
- All points not reachable from any other point are outliers. (noise)

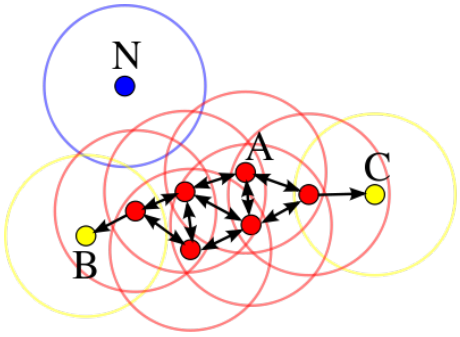


Figure 1: minPts = 4, red points are core points and N is a noise point.

3 Input

The input of my program must be a file in *.txt extension, belonging to the root directory "/datasets".

Each line of the input file correspond to a point, and its structure **must** be:

label_ id1,c1,...,cn
label_ idN,c1,...,cn

- *N* : total number of points and the same number of lines in the input file.
- *label_ id* : is identifier of the point. Usually a name/string.
- *c1 .. cn* : are the n values that define the point, normally integer or real.

3.1 Example of Input

```
A,0,0
B,0,1
C,0,2
...
K,12,5
L,12,6
O,12,7
```

4 Output

The output of the program will start by the number of clusters calculated following information about the input regarding the number of attributes (*n* in previous section) and the number of points (*N* also in the previous section). Then a list of points follows, in the form:

cluster_ number label_ id1 c1 ... cn

Where the cluster number means the reference of the cluster that point belongs in. E.g. if 2 points share the same cluster number, they both belong to the same cluster.

The output also contains a list with all noise points, and its values. And then in the end of the file there is the time of each execution phase of the algorithm

4.1 Example of Output

```
Clusters 2
Attributes 2
Points 18

1 A 0.0 0.0
1 B 0.0 1.0
1 C 0.0 2.0

2 K 12.0 5.0
2 L 12.0 6.0
2 O 12.0 7.0

Noise points
None

Read time (ms) 2
DBSCAN time (ms) 4
Total time (ms) 6
```

5 Design and Implementation

5.1 Design

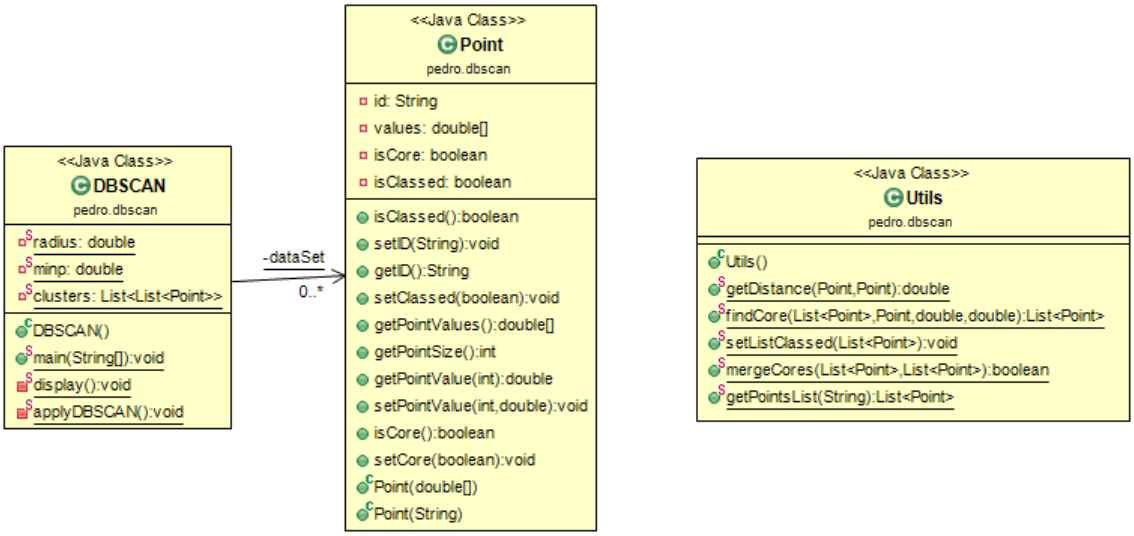


Figure 2: Class Diagram

5.2 Implementation Issues

The implementation of the algorithm is quite simple.

- Dynamic arrays are used.
- The distance metric to be calculated between two points is chosen by the user.
- Checks for every point p if they are core points, if so, a new list is generated with all points of that core circle.
- If a point is not at any of those lists is considered noise.
- Finally, the script joins the list that have points in common and defines it as a cluster.

6 User's Guide

The DBSCAN program can be found at the github repository in the link
https://github.com/pedrolindeza/dbscan

- Clone the project to your computer
- Choose the data set from */datasets/* folder. (DS from now)
- The metric value can be omitted, and in this case the default metric is Euclidean Distance. Otherwise the value must be one of these:

Metric	Command Value
Eucledian	1 (default)
Manhattan	2
Minkowski	3

- When Minkowski metric is chosen a value for root (m) must be specified.
- On the project's main directory run:

```
>> java -jar DBSCAN.jar DS.txt radius minp [metric [m]] > out.txt
```

- The output will be in the main directory as "out.txt".

The TIDBSCAN program can be found at the github repository in the link:
https://github.com/Smrtolet/TIDBSCAN

- Clone the project to your computer
- Choose the data set from */datasets/* folder. (DS from now)
- The metric value must be omitted. The value must be one of these:

Metric	Command Value
Eucledian	"eu"
Manhattan	"ma"
Minkowski	"mi (m)"

- When Minkowski metric is chosen a value for root (m) must be specified.
- The reference point(to which the distance is calculated and the points are sorted by that distance) type must be omitted. The values must be one of these:

Reference type	Command Value
Coordinate origin	"0"
Maximum attribute values	"max"
Minimum attribute values	"min"
Max,min,max,min.... attribute values	"maxmin"
Min,max,min,max.... attribute values	"minmax"

- On the project's main directory run:

```
>> java -jar TIDBSCAN.jar DS.txt radius minp [metric [m]] referenceType > out.txt
Example : java -jar TIDBSCAN.jar iris.txt 1 4 mi 2 max > out.txt
```

- The output will be in the main directory as "out.txt".

The Evaluation program can be found at the github repository at the link:
https://github.com/RafaelAPB/ML_EvaluationMeasures

- Clone the project
- If you wish, choose the data sets you want to use as input and place them on */input/* folder.
- On the project's main directory run:

  ```
  >> cmake CMakeLists.txt
  ```

- Go to the directory */exe* and run as administrator

  ```
  Evaluation.exe
  ```

7 Execution Example

We will test the execution on a data set (test.txt) with radius = 1 and minp = 3.

```
>> java -jar DBSCAN.jar test.txt 1 3 > out.txt
```

7.1 Input

```
A,0,0
B,0,1
C,0,2
D,0,3
E,0,4
F,0,5
G,12,1
H,12,2
I,12,3
J,12,4
K,12,5
L,12,6
M,0,6
N,0,7
O,12,7
P,0,8
Q,0,9
R,1,1
```

7.2 Output

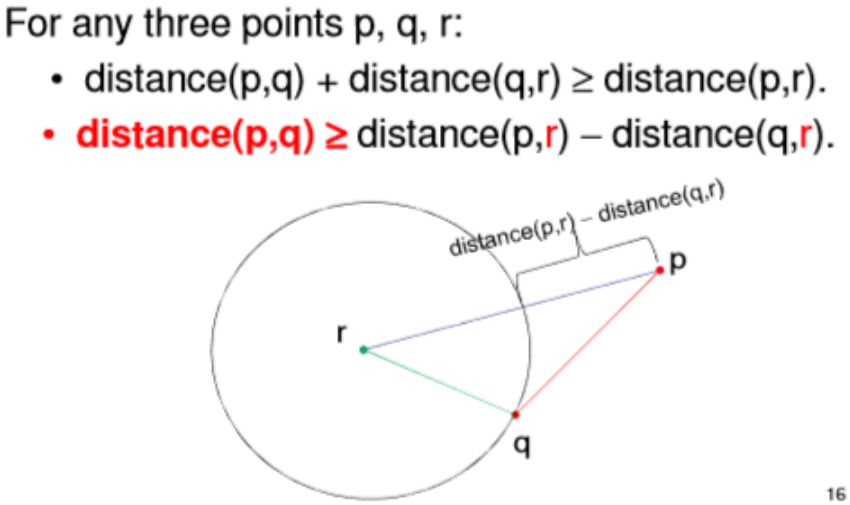
```
Clusters 2
Attributes 2
Points 18

1 A 0.0 0.0
1 B 0.0 1.0
1 C 0.0 2.0
1 R 1.0 1.0
1 D 0.0 3.0
1 E 0.0 4.0
1 F 0.0 5.0
1 M 0.0 6.0
1 N 0.0 7.0
1 P 0.0 8.0
1 Q 0.0 9.0

2 G 12.0 1.0
2 H 12.0 2.0
```

```
2 I 12.0 3.0
2 J 12.0 4.0
2 K 12.0 5.0
2 L 12.0 6.0
2 O 12.0 7.0

Noise points
None

Read time (ms) 2
DBSCAN time (ms) 4
Total time (ms) 6
```

8 TI-DBSCAN

We will optimize the DBSCAN algorithm by reducing the number of points for which the distance has to be calculated. TI-DBSCAN algorithm is a DBSCAN algorithm with Efficient Calculation of Eps-Neighboorhoods using triangle inequality property.

8.1 Triangle inequality property



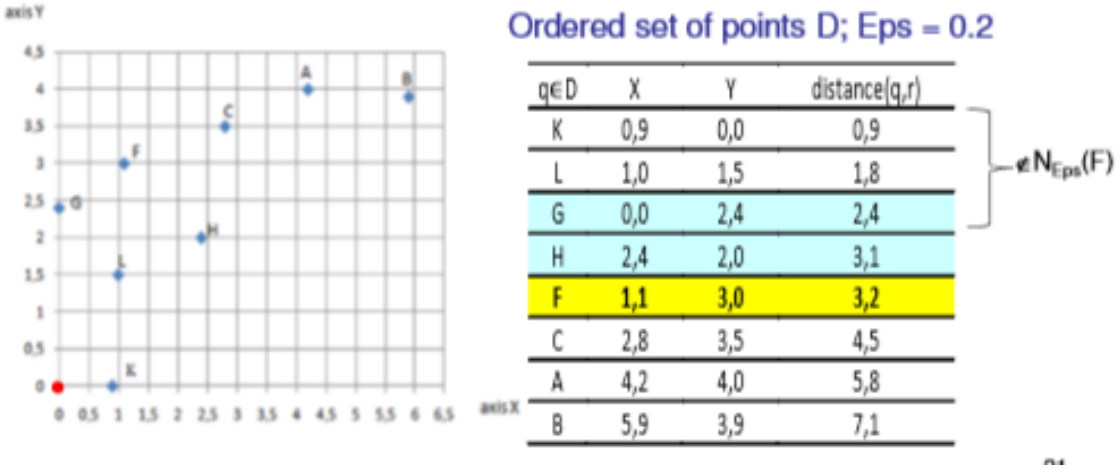
If the difference between 2 point's distances to a reference point is bigger than epsilon then we are sure that the real distance between these 2 points is equal or higher therefore it can never be inside epsilon radius of a point.

8.2 Applying TI to DBSCAN

Firstly we calculate distance from a reference point to all the points in the dataset. Reference point can be chosen in different ways as it is a parameter that we give to the program before execution. It can be a point with extreme maximums for values(out of all points) or minimums or combination of the two. It can also be an origin of the coordinate system (p = (0,0,0....)).

Then we sort them incrementally and save that list. When calculating the distance from a specific point we no longer calculate it between all points, but only for points for which the pessimistic distance (ie. difference between 2 point's reference distances) is not higher than radius.

We use sorted list in order to stop calculating pessimistic distance after we hit a point for which the pessimistic distance is higher than radius as show in the example below:



8.3 Implementation

Implementation issues are the same as previously written for DBSCAN with small modifications. Original DBSCAN code is modified as follows: -Addition of new parameter "ReferenceType" which allows us to choose the way we select the reference point.

-Point.class has new field "distanceToOrigin" which is used to sort and compare points by distance to the reference point. It is calculated only once per point.

-Utils.findCore method has been modified to calculate real distance only to the points for which the pessimistic distance is not higher than radius and to stop calculating pessimistic distance once that difference is preceded, instead of calculating it to every point.

The way it is used is described in the User's guide.

8.4 Comments

The output of TI-DBSCAN is the same as the DBSCAN for every dataset (the ordering of the elements might be different, but the sets in the clusters are the same). What has changed is the time of execution, as it is much faster with TI implementation, specially for larger datasets as as the time for preprocessing is a little higher, however the execution time is reduced exponentially.

9 Clustering Evaluation

Typical objective functions in clustering formalize the goal of attaining high intra-cluster similarity (documents within a cluster are similar) and low inter-cluster similarity (documents from different clusters are dissimilar). Clustering evaluation aims to provide tools and methods in order someone to know if a clustering is good or not. This section is based on Wikipedia's page on Clustering [3].

9.1 Internal Evaluation

Internal evaluation is the set of procedures that evaluates the clustering result based on the data that was clustered. Internal evaluation measures suffer from the problem that they represent functions that themselves can be seen as a clustering objective. One downfall is that by using internal measures for evaluation, we are comparing the similarity of the optimization problems, and not necessarily how useful the clustering is.

We implemented three internal evaluation measures:

9.1.1 Davies-Boulding Index

n - number of clusters
Cx - centroid of cluster x
σ - average distance of all elements in cluster x to centroid Cx
d(ci,cj) is the distance between centroids ci and cj

$$DB = \frac{1}{n}\sum_{i=1}^{n}\max_{j\neq i}\left(\frac{\sigma_i + \sigma_j}{d(c_i, c_j)}\right)$$

Figure 3: Davies-Boulding formula

The Davies–Bouldin index ≥ 0 is lower when algorithms produce clusters with low intra-cluster distances (high intra-cluster similarity) and high inter-cluster distances (low inter-cluster similarity). The clustering algorithm that produces a collection of clusters with the smallest Davies–Bouldin index is considered the best algorithm based on this criterion.

9.1.2 Silhouette Index

$$Silhouette(x) = \frac{b(x) - a(x)}{\max(b(x), a(x))}$$

Figure 4:
Silhouette Index formula

a(x) – the average distance between x and other objects in a group including x
b(x) – the minimum average distance between x and the nearest group.
This index is often used to determine the optimal number of clusters.
It has a value ∈[-1,1], where 1 means that the object is assigned to the best possible group, 0 the object is located between two groups, and -1, wrong assignment of the object [8]. Furthermore: n

$$GSilhouette = \frac{1}{N}\sum_{i=1}^{N} Silhouette(x_i)$$

Figure 5: Dataset Silhouette Index formula

- number of points in the dataset

9.1.3 Calinski-Harabasz Index

$[!CH(k) = [B(k)/W(k)] \times [(n-k)/(k-1))\backslash]$

where n = number of data points
k - number of clusters
W(k) - within cluster variation
B(k) - between cluster variation
CH criterion is based on ANOVA ideology. Hence, it implies that the clustered objects lie in Euclidean space of scale (not ordinal or binary or nominal) variables. CH criterion is most suitable in case when clusters are more or less spherical and compact in their middle (such as normally distributed, for instance). Other conditions being equal, CH tends to prefer cluster solutions with clusters consisting of roughly the same number of objects.

9.2 External Evaluation

In external evaluation, clustering results are evaluated based on data that was not used for clustering, such as known class labels and external benchmarks. Such benchmarks consist of a set of pre-classified items, and these sets are often created by (expert) humans. In this project, we used the class labels as a comparison to how the clustering algorithm performed.

$$\frac{1}{N}\sum_{m\in M}\max_{d\in D}|m\cap d|$$

Figure 6: Dataset Purity formula

9.2.1 Purity

Purity $\in[0,1]$ is a measure of the extent to which clusters contain a single class. M - set of clusters
D - some set of classes D
N - data points

Note that this measure doesn't penalize having many clusters. So for example, a purity score of 1 is possible by putting each data point in its own cluster. Also purity doesn't work well for imbalanced data.

9.2.2 F-Measure

$$F_\beta = \frac{(\beta^2+1)\cdot P\cdot R}{\beta^2\cdot P+R}$$

Figure 7: F-Measure formula

The F-measure ≥ 0 is a measure of a test's accuracy. It can be used to balance the contribution of false negatives by weighting recall through a parameter. Furthermore, P (precision) and R (recall) are given by:

$$P = \frac{TP}{TP+FP}$$
$$R = \frac{TP}{TP+FN}$$

Figure 8: Precision and Recall formulas

9.2.3 Rand Index

TP - number of true positives
TN - number of true negatives
FP - number of false positives
FN - number of false negatives

$$R = \frac{TP+TN}{TP+TN+FP+FN}$$

The Rand index $\in$ [0,1] computes how similar the clusters (returned by the clustering algorithm) are to the benchmark classifications. One can also view the Rand index as a measure of the percentage of correct decisions made by the algorithm.

10 Result Evaluation

In this section, a comprehensive evaluation of different datasets is performed. This evaluation is conducted by comparing the percentage number of points used on a dataset by DBSCAN and time. Furthermore, the results of internal and external evaluation methods are shown for each dataset. The F-Measure has always $\beta = 1$.

10.1 Iris Dataset

- https://archive.ics.uci.edu/ml/datasets/iris
- radius = 0.5 and minp = 10 metric = eu (referenceType = max)

10.1.1 Result

- For DBSCAN Algorithm

Points	Clusters	Preprocess Time (ms)	Exec Time (ms)	Total Time (ms)
50 (33%)	1	3	56	59
100 (66%)	2	4	85	89
150 (100%)	2	7	138	145

- For TIDBSCAN Algorithm

Points	Clusters	Preprocess Time (ms)	Exec Time (ms)	Total Time (ms)
50 (33%)	1	2	18	20
100 (66%)	2	4	55	59
150 (100%)	2	5	65	70

- External evaluation methods

Evaluation method	Value
Purity $\in[0,1]$	0,98
F-Measure ≥ 0	0.909985
Rand-Measure $\in[0,1]$	0.986667

Preprocess Time (ms) = 4016
Exec Time (ms) = 53

Analysis:
Purity = 1, therefore all the clusters contain only one single class.
F-Measure is around 0.9%, therefore we can conclude that the clustering was accurate.
Rand-Measure is around 0.99%, therefore we can conclude that the algorithm made mostly correct decisions.
The analysis for the next data sets, concerning the external evaluation methods are analogous.

- Internal evaluation methods

Evaluation method	Value
Davies-Boulding Index ≥ 0	1.5203
Silhouette Index $\in[-1,1]$	0.443144
Calinski-Harabasz Index ≥ 0	620.371

Preprocess Time (ms) = 4016
Exec Time (ms) = 53

The Davies-Boulding and the Calinsky-Harabasz indexes are used to perform benchmarking against other clustering algorithms.
The Silhouette Index is 0.44, which means the objects are assigned properly. The analysis for the next data sets, concerning the internal evaluation methods are analogous.

10.2 Absenteeism Dataset

- http://archive.ics.uci.edu/ml/datasets/Absenteeism+at+work
- radius = 10 and minp = 10 metric = eu (referenceType = max)

10.2.1 Result

- For DBSCAN Algorithm

Points	Clusters	Preprocess Time (ms)	Exec Time (ms)	Total Time (ms)
245 (33%)	2	18	336	354
490 (66%)	4	28	1034	1062
740 (100%)	9	38	1442	1480

- For TIDBSCAN Algorithm

Points	Clusters	Preprocess Time (ms)	Exec Time (ms)	Total Time (ms)
245 (33%)	2	12	188	200
490 (66%)	4	15	384	399
740 (100%)	9	20	557	577

- External evaluation methods

Evaluation method	Value
Purity $\in$[0,1]	1
F-Measure ≥ 0	0.672118
Rand-Measure $\in$[0,1]	0.972372

Preprocess Time (ms) = 3618
Exec Time (ms) = 971

- Internal evaluation methods

Evaluation method	Value
Davies-Boulding Index ≥ 0	8.48829
Silhouette Index $\in$[-1,1]	0.918352
Calinski-Harabasz Index ≥ 0	1814.33

Preprocess Time (ms) = 3618
Exec Time (ms) = 971

10.3 Cardio Dataset

- http://archive.ics.uci.edu/ml/datasets/Cardiotocography
- radius = 10 and minp = 10 metric = eu (referenceType = max)

10.3.1 Result

- For DBSCAN Algorithm

Points	Clusters	Preprocess Time (ms)	Exec Time (ms)	Total Time (ms)
710 (33%)	0	49	1464	1513
1420 (66%)	1	41	2796	2837
2129 (100%)	6	64	2597	2661

- For TIDBSCAN Algorithm

Points	Clusters	Preprocess Time (ms)	Exec Time (ms)	Total Time (ms)
710 (33%)	0	18	535	553
1420 (66%)	1	37	1083	1120
2129 (100%)	6	30	1492	1522

- External evaluation methods

Evaluation method	Value
Purity ∈[0,1]	0.937529
F-Measure ≥ 0	0.249992
Rand-Measure ∈[0,1]	0.989588

Preprocess Time (ms) = 3868
Exec Time (ms) = 9885

- Internal evaluation methods

Evaluation method	Value
Davies-Boulding Index ≥ 0	7.96099
Silhouette Index ∈[-1,1]	0.961363
Calinski-Harabasz Index ≥ 0	6388.63

Preprocess Time (ms) = 3618
Exec Time (ms) = 971

11 Conclusions

- We can observe that the TI-DBSCAN preforms at higher speed than DBSCAN, especially with the increase of data. DBSCAN's execution time compared to total number of points increases more polynomially than TIDBSCAN's execution time.
- The implemented DBSCAN algorithm (r = 10, minP = 10) performed the best on the Cardio dataset (2129 points, 6 clusters discovered), because it's Rand-Measure (0,989) and Silhouette Index (0,961) are the highest.

References

[1] mkr_Clustering45_notes.pdf

[2] https://en.wikipedia.org/wiki/DBSCAN

[3] https://en.wikipedia.org/wiki/Cluster_analysis

[4] https://stats.stackexchange.com/questions/86645/variance-within-each-cluster

[5] https://stats.stackexchange.com/questions/97429/intuition-behind-the-calinski-harabasz-index?rq=1

[6] https://link.springer.com/article/10.1007/s40595-016-0086-9

[7] https://stats.stackexchange.com/questions/21807/evaluation-measure-of-clustering-without-having-truth-labels

[8] http://staff.ii.pw.edu.pl/ gprotazi/