DBSCAN
A project work on the course Clustering Methods

Ville Kumpulainen
University of Eastern Finland

February 29, 2012

Abstract

This paper is a semi-scientific paper representing density-based spatial clustering of applications with noise (DBSCAN) and some experimentations in an attempt to induce convenient parameters for DBSCAN automatically.

Contents

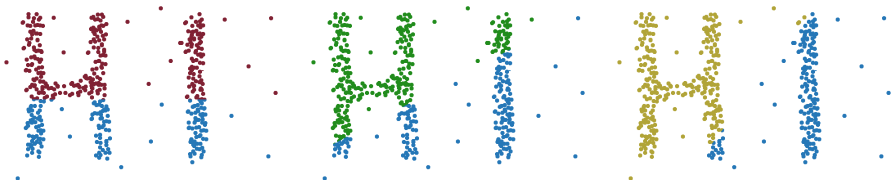
1 Introduction

1.1 Issues of traditional clustering methods

Traditional clustering methods, such as K-Means or Randomized Local Search, introduce variety of problems. These issues are related (but not limited) to optimal parameters, varying initial settings, clusters with arbitrary shape and noisy measurements.

The major challenge in finding optimal parameters for an algorithm is usually number of clusters, as in most cases it is not concluded by the algorithm itself. Non-optimal number of clusters may result visually uniform and dense sections being splitted in halves (usually from the middle of a segment, which tends to be the densest part) or joined under one cluster.

Varying initial settings, such as initial position of centroids, have an considerable impact to the result of the algorithm. Therefore some of the outputs provided by a clustering method are not optimal with respect to the selected clustering method. Effect of random initial positioning of centroids is demonstrated in figure 1.



(a) Result 1 (b) Result 2 (c) Result 3

Figure 1: Illustration of K-means' dependency on initial centroid positioning

Traditional methods based on local repartitioning after moving centroids according to their current partitioning, such as K-means or Randomized Local Search, are based on repartitioning the clusters according to distance metric, usually Euclidean metric. Due to this repartitioning criterion those methods are generally efficient for finding packed sets of data that span uniformly to all dimensions, therefore forming sets of spherical shapes. In real life measurements, however, occuring patterns may have non-spherical shapes. Therefore some of the traditional methods are generally less than ideal for detecting patterns with arbitrary shape.

In practical problems data usually has distorted isolated measurements due to measurement errors or randomness of the measured object. These outliers usually interfere the clustering process, hence reducing and intro-

ducing randomness to the overall quality of the clustering.

1.2 Pros of DBSCAN

DBSCAN addresses some of the mentioned issues.

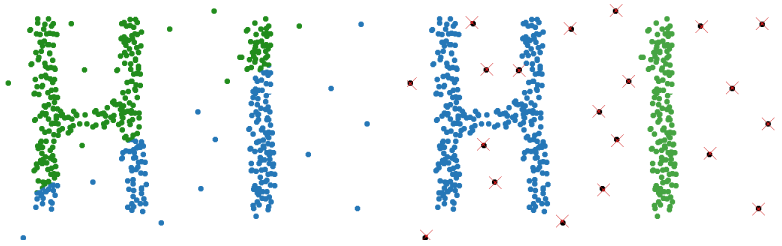
Implementations of DBSCAN ideally have no randomness; the output of the method is repeatable due to its definitions. Therefore there is no need for repetition "in case that the initial clustering was just abnormally bad".

As opposed to some of the traditional methods, DBSCAN is able to detect noise. This results more accurate clustering without interfering outliers.

Number of clusters is induced automatically during the process. Due to automatic inducing of clusters based on densities there is no possibility of DBSCAN splitting dense (dense w.r.t provided parameters, that is) segments in multiple clusters.

DBSCAN classifies dense visually uniform segments with arbitrary shape to the same cluster. It outperforms CLARANS, another arbitrary shape detecting clustering algorithm, in terms of computing time [1].

Illustration emphasizing some of these mentioned aspects can be seen in figure 2.



(a) Possible outcome on K-means (b) DBSCAN

Figure 2: Illustration of possible results on K-means and DBSCAN with data that is packed in visually distinct segments

1.3 Cons of DBSCAN

Criterion for density remains the same during the clustering process. This is, however, not optimal procedure for data having clusters with varying densities: either sparse clusters may be considered as a mixture of noise and small isolated clusters as a result of too hard criterion or noise is incorrectly classified as a part of a cluster as a result of more allowing criterion. In either case the result may differ from desired outcome.

MSE, or any other function emphasizing minimal distances inside a cluster, is a bad measure for evaluating goodness of DBSCAN mainly for two reasons: cluster centroids are not defined in DBSCAN and, if they were, MSE of DBSCAN clustering is rarely the best one: DBSCAN detects optimal shapes in terms of density rather than minimal intra-cluster distances. In addition, as most traditional methods concentrate on minimizing the MSE or at least detect clusters with spherical shapes, which is approximately ideal shape in terms of minimal MSE, DBSCAN seems the least desirable algorithm in comparison. The difference is likely to increase in favor of the other clustering algorithms when clusters detected by DBSCAN are elongated or concave.

2 DBSCAN

2.1 Definitions

Let

- D be the number of data points
- dist(x,y) be the Euclidian distance between vectors x and y
- Eps and MinPts be parameters of DBSCAN

Eps-neighborhood Eps-neighborhood of a point p w.r.t. Eps is $N_{Eps}(p) = \{q \in D | dist(p,q) \leq Eps\}$. N_{Eps} is therefore a set of points having at most distance of Eps to p.

Core point Point p is a core point w.r.t. Eps and MinPts iff $|N_{Eps}(p)| \geq MinPts$. A core point has at least MinPts surrounding points within the radius of Eps.

Direct density-reachability Point p is directly density-reachable from point q w.r.t. Eps and MinPts iff $|N_{Eps}(q)| \geq MinPts$ and $p \in N_{Eps}(q)$. That is, q is a core point and p resides within radius of Eps from q ($p \in N_{Eps}(q) \Leftrightarrow dist(p,q) \leq Eps$).

Density-reachability Point p_n is density-reachable from point p_1 w.r.t. Eps and MinPts iff there is a set of points $\{p_1, p_2, ..., p_{n-1}, p_n\}$ so that p_{k+1} is directly density-reachable from $p_k (\forall k \leq n-1)$ w.r.t. Eps and MinPts. Let it be noted that according to the definition of direct density-reachability $p_1, p_2, ..., p_{n-1}$ are core points, but p_n necessarily is not.

Density-connectivity A point p is density-connected to point q (and vice versa) iff there is a point o so that both p and q are density-reachable from o w.r.t. Eps and MinPts.

By using these definitions cluster and noise are defined as follows

Cluster C is a non-empty subset of D satisfying the following conditions:

Maximality $\forall p, q \in D$: $q \in C$ and p is density-reachable from $q \Rightarrow p \in C$

Connectivity $\forall p, q \in C$: p is density-connected to q

Noise is a set of points not belonging to any cluster; $noise = \{p \in D | \forall i : p \notin C_i\}$

2.2 The algorithm

The algorithm below is taken from [1]. It consists of two functions, DBSCAN and ExpandCluster. ExpandCluster checks whether the given point Point is a core point w.r.t. Eps. If it is not, Point is classified as noise. Otherwise a breadth-first search[1] is executed starting from Point, until all points (the above maximality criterion must be satisfied) density-connected to Point have been checked and classified as ClId. After a search DBSCAN updates ClusterId and executes ExpandCluster for succeeding point, until ExpandCluster has been executed for all points.

```
function DBSCAN(SetOfPoints, Eps, MinPts)          ▷ SetOfPoints is
UNCLASSIFIED
    ClusterId := nextId(NOISE);
    for i = 1 → SetOfPoints.size do
        if Point.ClId = UNCLASSIFIED then
            if ExpandCluster(SetOfPoints, Point, ClusterId, Eps, MinPts)
then
                ClusterId := nextId(ClusterId);
            end if
        end if
    end for
end function
function ExpandCluster(SetOfPoints, Point, ClId, Eps, MinPts)
    seeds := SetOfPoints.regionQuery(Point, Eps);
    if seeds.size < MinPts then                          ▷ no core point
        SetOfPoint.changeClId(Point, NOISE);
        return false
    else              ▷ all points in seeds are density-reachable from Point
        SetOfPoints.changeClIds(seeds, ClId);
        seeds.delete(Point);
        while seeds != Empty do
            currentP := seeds.first();
            result := SetOfPoints.regionQuery(currentP, Eps);
            if result.size >= MinPts then
                for i = 1 → result.size do
                    resultP := result.get(i);
                    if resultP.ClId ∈ UNCLASSIFIED, NOISE then
                        if resultP.ClId = UNCLASSIFIED then
```

[1] the provided implementation uses depth-first search instead

```
                seeds.append(resultP);
            end if
            SetOfPoints.changeClId(resultP, ClId);
        end if                          ▷ UNCLASSIFIED or NOISE
    end for
end if                                  ▷ currentP was core point
end while
return true
end if
end function
```

2.2.1 Time complexity

When breadth-width search is being repeated for each of the points, but the results of the earlier searches are considered (search is not being repeated for points that already have been found and classified as a part of a cluster), each point is handled at most two times; either once as a point found by search and second time as a seed point, or vice versa for boundary points (at first they may be classified as noise).

Clearly the most demanding task in terms of time complexity is regionQuery. It is called at most once for each data point. The most sophisticated implementations utilize spatial search trees such as R* tree, hence reducing time complexity of each query to $O(\log n)$ and time complexity of approximately n queries to $O(n \log n)$. A more coarse approach, such as inspecting each of the distances for the regional query by a brute force approach, yields total time complexity of $O(n^2)$. [2]

2.3 Inducing convenient parameters for DBSCAN

It has been suggested that in practise MinPts = 4 is an optimal value between computing time and accuracy. Furthermore, it has been suggested that value for Eps could be chosen based on the following algorithm. [1]

[2] In the provided implementation data points are sorted along to x-axis, thereby notably, but yet linearly, reducing amount of points needed to inspect for regional query. The worst case considered, time complexity of the implementation remains $O(n^2)$. Actually it would have been tempting to experiment with initializing a spatial search tree using hierarchical clustering methods or hierarchical K-means repeated iteratively on centroids, but in most cases the worst, yet considerably rare, case of regional query still had been $O(n^2)$. In addition, using a clustering method in order to use another clustering method would have been awkward.

For each $p \in D$

- find point q being (MinPts)th nearest to p
- array.add(dist(p,q))

Sort array to descending order

Find a point x in array where decreasing of values significantly reduces ("knee point")

Set Eps = array[x]

However, in [1] finding the desired point x was discussed only shortly and implementation of such an algorithm was generally left for the reader to consider. There are some algorithms available, such as F-ratio variance test or Knee Point Detection on BIC, but I intended an algorithm on my own instead out of curiosity. The algorithm is discussed below.

A good algorithm should find a distinctive knee point of the global skew rather than random local abnormalities that are likely to occur in the graph as well. Therefore an algorithm searching for candidates should inspect its local skew on a global scale rather than on a local scale. It very rapidly occured to me that instead of comparing each point to a set number of its immediate succeeding values (which undesirably emphasizes local abnormalities) I should inspect the difference of candidate's value related to more distant values. The algorithm then quickly emerged more as a result of sound reasoning and combination of trial and error rather than formal concluding.

At first the results of the algorithm were visually pleasing to the human observer. As promising as the initial results were, I later discovered that the output of the algorithm in general is unpredictable, hence suggesting that promising initial results were just due to good luck.

2.3.1 The algorithm

The algorithm, illustrated in figure 3, proceeds iteratively as follows:

Split graph horizontally to n sections

Estimate approximate angle on each section by inspecting the first and last point of a section

Find two sections next to each other having the biggest difference in angles

Join at most m (where m is even) sections neighboring the found boundary

Repeat for graph inside the joined area until resolution of the restricted graph is too small for splitting

Select average graph value from remaining area to Eps

A pseudo code representation is below [3].

```
function FindEps(Array, m, n)                    ▷ Array is already sorted
    Range := from 1 to Array.length
    while Range.length > m do
        Ranges := Range.split(n)
        best_difference := 0
        Angles[0] := [tan((Array[Ranges[0].start] - Array[Ranges[0].end]) /
Ranges[0].length), 0]
        for i = 1 → n − 1 do
            Angles[i mod 2] := tan((Array[Ranges[i].start] - Array[Ranges[i].end])
/ Ranges[i].length)
            if |Angles[0] − Angles[1]| ≥ best_difference then
                best_difference := |Angles[0] − Angles[1]|
                if i − 1 − m/2 < 0 then
                    Surroundings := from Ranges[0].start to Ranges[i − 1 +
m/2].end
                else if i − 1 + m/2 ≥ n then
                    Surroundings := from Ranges[i−1−m/2].start to Ranges[n−
1].end
                else
                    Surroundings := from Ranges[i−1−m/2].start to Ranges[i−
1 + m/2].end
                end if
            end if
        end for
        Range := Surroundings
    end while
    sum := 0
    for i = Range.start → Range.end do
        sum := sum + Array[i]
    end for
    return sum / Array.length
end function
```

[3] Implementation of this can be found from DBSCAN::initialize in the provided DBSCAN implementation

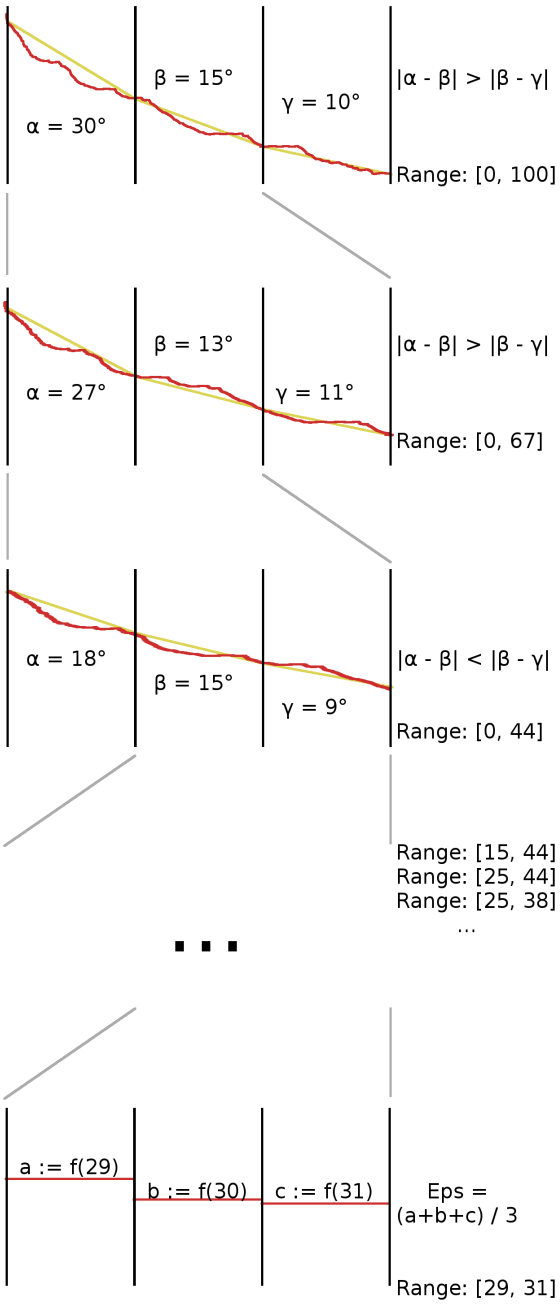


Figure 3: Illustrated example of the algorithm with graph f(x), $x \in [0, 100]$

2.3.2 Time complexity of the algorithm

Within each iteration at least $(n - m)/n$ of array cells are discarded from later inspections (at most m from n sections are kept). Let k be the number of cells in the array. Time complexity is now $O(mlog_{n/m}k)$.

2.3.3 Evaluation of the algorithm

Initially the algorithm was simply implemented with hard-coded values of $n = 3$ and $m = 2$. On this setup the results on data sets S1, S2, S3 and S4 were promising; Eps value returned by the algorithm yielded visually pleasing DBSCAN clusterings. After initial results m and n were parameterized for further experimentation.

Table 1 demonstrates experimentations on parameter n. Parameter m was set to a biggest power of two less than n in order to split approximately equally or less than half on each iteration. DBSCAN was run on achieved value for Eps and number of clusters C and noise percentage NP were recorded from each result.

These experimentations show that increase in value n introduces randomness to yielded Eps values. In general probability of low Eps is likely to increase within n, but unpredictable exceptions also occur.

Although low values of n are more likely to yield visually pleasing results, there is considerable instability among those Eps-values as well, hence suggesting that every possible value of n yields is able to conduct bad Eps for some data. That probability is likely to increase within n.

On table 2 different values of m were experimented in order to find convenient Eps. Clustering was then applied to the given dataset and resulting number of clusters C and noise percentage NP was recorded. These results suggest that for larger values of n there is no such value m that would yield merely acceptable Eps for some data sets; most notably amount of points detected as noise in S1 and S2 are considerably high. It should be noted that S1 and S2 have the most distinctive patterns.

m	n	s1	s2	s3	s4
2	3	$\varepsilon = 13118.1$ C=30 NP=9.0%	$\varepsilon = 17411.5$ C=19 NP=7.4%	$\varepsilon = 13494.9$ C=53 NP=12.9%	$\varepsilon = 15448.8$ C=19 NP=5.8%
2	4	$\varepsilon = 13610.7$ C=27 NP=8.3%	$\varepsilon = 28644.9$ C=3 NP=0.8%	$\varepsilon = 14360.6$ C=33 NP=10.8%	$\varepsilon = 14850.8$ C=23 NP=6.5%
4	5	$\varepsilon = 6032.2$ C=80 NP=40.0%	$\varepsilon = 16210.6$ C=23 NP=9.0%	$\varepsilon = 13567.6$ C=50 NP=12.6%	$\varepsilon = 14591.1$ C=25 NP=6.7%
4	6	$\varepsilon = 8540.1$ C=67 NP=23.2%	$\varepsilon = 18521.5$ C=19 NP=6.1%	$\varepsilon = 13904.3$ C=44 NP=11.7%	$\varepsilon = 14593.1$ C=25 NP=6.7%
4	7	$\varepsilon = 12739.4$ C=34 NP=9.7%	$\varepsilon = 15976.1$ C=23 NP=9.3%	$\varepsilon = 12590.0$ C=62 NP=16.1%	$\varepsilon = 12734.3$ C=53 NP=10.6%
4	8	$\varepsilon = 9161.4$ C=63 NP=19.6%	$\varepsilon = 11037.1$ C=63 NP=22.3%	$\varepsilon = 13497.9$ C=52 NP=12.8%	$\varepsilon = 12450.8$ C=52 NP=11.7%
8	9	$\varepsilon = 9071.4$ C=61 NP=20.2%	$\varepsilon = 1513.3$ C=27 NP=93.0%	$\varepsilon = 11149.0$ C=94 NP=21.7%	$\varepsilon = 12754.8$ C=54 NP=10.3%
8	10	$\varepsilon = 5886.4$ C=79 NP=41.5%	$\varepsilon = 1984.6$ C=32 NP=90.0%	$\varepsilon = 12779.3$ C=60 NP=15.4%	$\varepsilon = 12720.9$ C=53 NP=10.7%
8	11	$\varepsilon = 1525.4$ C=35 NP=90.3%	$\varepsilon = 875.0$ C=9 NP=98.0%	$\varepsilon = 13497.9$ C=52 NP=12.8%	$\varepsilon = 11739.1$ C=61 NP=14.4%
8	12	$\varepsilon = 4031.2$ C=91 NP=61.8%	$\varepsilon = 1526.5$ C=25 NP=92.9%	$\varepsilon = 13634.1$ C=47 NP=12.3%	$\varepsilon = 12120.7$ C=50 NP=13.4%
8	15	$\varepsilon = 1690.2$ C=34 NP=88.8%	$\varepsilon = 1033.1$ C=19 NP=96.5%	$\varepsilon = 11133.0$ C=95 NP=21.7%	$\varepsilon = 14436.1$ C=25 NP=7.0%
16	31	$\varepsilon = 3407.2$ C=70 NP=70.8%	$\varepsilon = 8013.1$ C=93 NP=37.2%	$\varepsilon = 6050.8$ C=96 NP=61.7%	$\varepsilon = 8661.7$ C=127 NP=32.4%
64	100	$\varepsilon = 4526.7$ C=93 NP=55.6%	$\varepsilon = 8302.2$ C=84 NP=35.2%	$\varepsilon = 4388.0$ C=70 NP=76.7%	$\varepsilon = 7889.1$ C=130 NP=39.6%
2048	2500	$\varepsilon = 2594.2$ C=49 NP=80.2%	$\varepsilon = 19291.1$ C=20^{14} NP=5.2%	$\varepsilon = 4184.3$ C=62 NP=78.3%	$\varepsilon = 18605.4$ C=7 NP=3.7%

Table 1: Experimentation with different values of n

m	n	s1	s2	s3	s4
2	15	$\varepsilon = 2243.4$ C=38 NP=84.4%	$\varepsilon = 1122.5$ C=25 NP=95.7%	$\varepsilon = 12637.7$ C=62 NP=15.9%	$\varepsilon = 12753.4$ C=55 NP=10.3%
4	15	$\varepsilon = 950.2$ C=18 NP=96.1%	$\varepsilon = 998.3$ C=14 NP=97.2%	$\varepsilon = 13506.0$ C=50 NP=12.8%	$\varepsilon = 12451.7$ C=52 NP=11.7%
6	15	$\varepsilon = 1799.7$ C=35 NP=87.5%	$\varepsilon = 1510.0$ C=27 NP=93.0%	$\varepsilon = 11146.3$ C=94 NP=21.7%	$\varepsilon = 13706.3$ C=37 NP=8.2%
8	15	$\varepsilon = 1690.2$ C=34 NP=88.8%	$\varepsilon = 1033.1$ C=19 NP=96.5%	$\varepsilon = 11133.0$ C=95 NP=21.7%	$\varepsilon = 14436.1$ C=25 NP=7.0%
10	15	$\varepsilon = 1530.6$ C=35 NP=90.2%	$\varepsilon = 1985.9$ C=32 NP=90.0%	$\varepsilon = 9651.9$ C=104 NP=30.7%	$\varepsilon = 10450.7$ C=85 NP=20.2%
12	15	$\varepsilon = 4305.9$ C=87 NP=59.0%	$\varepsilon = 1250.9$ C=19 NP=95.2%	$\varepsilon = 11133.0$ C=95 NP=21.7%	$\varepsilon = 8237.5$ C=130 NP=36.6%
14	15	$\varepsilon = 2409.3$ C=44 NP=82.0%	$\varepsilon = 1597.9$ C=27 NP=92.2%	$\varepsilon = 11179.5$ C=93 NP=21.6%	$\varepsilon = 7695.2$ C=126 NP=42.0%

Table 2: Experimentation with different values of m

2.3.4 Conclusions

Although lower n yields better results, the procedure of the algorithm is essentially the same $\forall n \geq 3$, hence suggesting that the algorithm is also likely to yield bad value for Eps on n = 3 with some carefully selected measurement data. The algorithm can be used but with consideration.

2.3.5 Future work

An intresting alternative approach could be to calculate weighted local average from neighboring values rather than taking directly the value of a split point. This would reduce effect of possibly abnormal value on the splitting boundary. Experimentations on this are, however, not discussed due to lack of time preserved for this project work.

The proposed idea is illustrated in figure 4: the graph reduces constantly right before the split boundary and this trend continues a while after passing the boundary, but unfortunately there is a steep temporary decline near the boundary reducing the representativeness of approximate lines. As the temporary decline happens to be inside the region inspected for local average, the average is able to reduce the effects of this temporary abnormality of a graph, hence yielding better approximation of the graph.

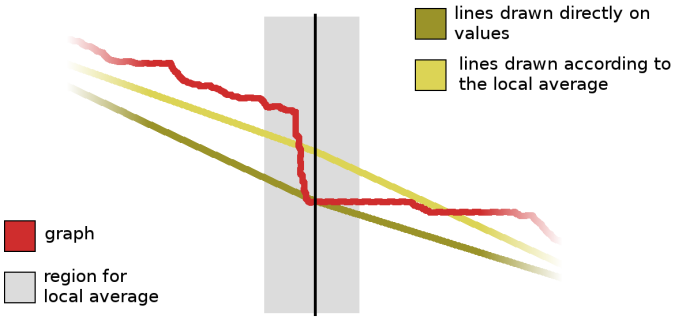


Figure 4: Illustration of taking the local average to consideration

References

[1] Martin Ester; Hans-Peter Kriegel; Jörg Sander; Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of 2nd International Conference on Knowledge Discovery and Data Mining (KDD-96)*, 1998.

A Software usage

A.1 Preparations

The software has been implemented in Ruby, so no compiling is required. In order to run the software, following preparations must be considered:

Install Ruby from http://www.ruby-lang.org/en/downloads/

Install required Ruby gems by executing these lines in command prompt / terminal

- gem install rubysdl-mswin32-1.9 (On Windows)
- gem install rubysdl (On Linux)

In case of problems on Linux consider using Ruby SDL implementation provided by your Linux distribution specific package manager instead.

A.2 Usage

Using command prompt / terminal change local directory to project directory and type "ruby run.rb -?" for instructions. Some possible examples of usage:

ruby run.rb s1.txt -o output.bmp Cluster s1.txt and write the resulting image to output.bmp.

ruby run.rb s1.txt s2.txt -l log.txt -b Run clustering for s1.txt and s2.txt and write a log file to log.txt. Run in batch mode; once clustering is finished, clustering s2.txt is started without user input.

ruby run.rb s3.txt s4.txt -l log.txt -o %n-output.bmp -b -s 1024 768 Write a log file log.txt and resulting images s3-output.bmp and s4-output.bmp with resolution 1024x768.

B Material of the course project work

B.1 Access to the material

Material for the course project work is available at http://vilikki.kapsi.fi/Opiskelu/KLU/work/. Login credentials for the site are

USER NAME rykelma

PASSWORD dbscan

B.2 Contents of the directory

Directory has at least these files:

article.pdf This paper.

presentation.pdf Presentation of the course project work presenting the basic idea behind DBSCAN and discussing pros and cons of DBSCAN.

rykelma.zip Implemented clustering software. Available under X11 license.

dbscan_s2.avi A MPEG-4 encoded video of DBSCAN running on S2. Intended to use as a part of presentation to demonstrate progress of depth-first search implementation.