Eötvös Loránd University
Faculty of Informatics
Department
of Information Systems

Generating test data from a sample population using data mining methods*

Supervisor:
dr. Kiss, Attila
Associate Professor and Head of Department,
Department of Information Systems

Written by:
Fazekas, Bálint
Masters of Computer Science
Department of Software Technology And Methodology

Budapest, 2017

*This work was supported by EFOP-3.6.3- VEKOP-16- 2017-00002 project.

Abstract

The aim of this paper is to generate a large database based on the clustering and statistical properties of a small dataset sample. For the clustering of the small dataset, the *"k–means"* and the *"DBSCAN"* algorithms are being used and compared. By embedding the k–means algorithm in the DBSCAN, the paper describes a much more efficient method of dataset clustering, which avoids the weaknesses of both algorithms when they are not combined. The results of the *"hybrid method"* are smaller sub–clusters of which their centroids and distribution properties can be calculated upon observation. Based on the acquired properties of these sub–clusters, an attempt is made to regenerate the original dataset sample, with less, equal amount, and more data points.

Contents

List of Figures

1 Introduction

Big-data generation is the process of creating a single, or several clusters of data, in order to simulate a truthful dataset. The generated data should obey several rules and restrictions, as a real database can contain more than one data–type, and therefore, there are restrictions on those data–types. Let us consider the example of a *Person* table in a database, with three data–types: *Name:String, Age:Integer,* and *Date of birth:Date.* Each of these attributes should obey the restrictions of their data–types, so that there are no numbers in the name, and no letters in the numerical data. In a real database, a table as such would contain occurrences where each attribute would have a unique statistical property, that can be mathematically analyzed.

This paper focuses on the methods of which a smaller dataset (with attribute restrictions) can be analyzed in order to artificially create a dataset with very similar attributes as the original one. This process will be referred as *data regeneration* from this point onwards.

Data regeneration is a very powerful procedure in the field of Big–Data research, as it can create a truthful, but artificial dataset that can be used to optimize database algorithms for faster performance. The truthful aspect ensures the correct *shape, distribution, and size* of the dataset, while the artificial aspect ensures the legal use of the regenerated data.

The main focus of this paper is to introduce, explain, and combine statistical properties and *clustering algorithms* that are necessary for successful data regeneration.

Throughout this paper, we will consider a dataset that contains 2–dimensional coordinates on the x and y axes; *mean, centroid, variance, and distribution* as statistical properties, and clustering algorithms such as the *k–means* and *DBSCAN* algorithms.

This paper also discusses the advantages and disadvantages of using either algorithms, and using the combined method. The accuracy of the generated data is compared in the latter part of the paper, as well as the time analysis of how much time the data regeneration needs for different sizes of datasets.

Hypothesis

The hypothesis of the paper states that regenerating data by finding the centroids of sub–clusters created by the DBSCAN and the k–means algorithms combined, gives a much better result than trying to regenerate the original data by using only one of these algorithms.

2 Definitions

The mathematical terms and definitions used by this paper are defined below, in order to avoid the confusion of the reader.

2.1 Point

In this paper, we define a *point* as a single location on the 2–dimensional (x and y) plane. Due to graphical representational intentions, each point has an x and y co–ordinate residing in the positive quadrant of the 2–dimensional, x–and–y–plane.

Formally:

$$\text{Let } P \in Points$$

$$P = (x, y), \text{where } 0 \leq x, y \in \mathbb{R} \text{ or } \mathbb{R}^2_{\geq 0}[9]$$

and the *Points* set contains P points as defined above.

In the graphical representation of a set of P points, the y–coordinates *increase downwards on the screen.* It is important to note that the points can have any sort of finite numerical value, and their strict positive nature is chosen due to the similarity of the screen coordinate system, therefore their easy presentation with minimal alteration, if needed. A point has not other property than a position on the screen–coordinate system.

It is a principal of this paper to minimize the changes of the points contained in a dataset, as to try to preserve their fix nature. If a point could have a negative attribute as one of its co–ordinates, then the entire dataset would have to be translated in the direction of the largest difference between the origin and a point in a negative quadrant. It would have to be done so, in order to keep the relative distances between all the points in the dataset.

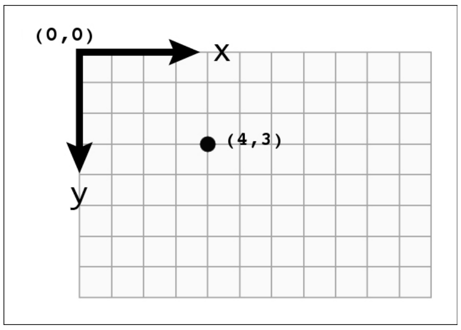


Figure 1: A point on the screen–coordinate system

2.2 Vector

In order to define mathematical functions on the *Points*, we need to handle each $P \in Points$ as a vectorial value. This allows us to define the following:

Let $P \cong Vec$, where $Vec \in Vectors$.

In this sense, we can deduce that: $Points \cong Vectors$

Verbally explined: we define vectors similarly as points – the only difference between the two is that vectors have a direction and a magnitude. For this paper, the direction of a vector can be neglected, as it does not play a significant role in the solution of the problem.

Note, that in this definition, the attributes of a vector are the same as of a P point, and resides in the same 2–dimensional plane quadrant.

From here on, the terms *point* and *vector* will be used interchangeably.

2.2.1 Magnitude of a vector

The magnitude of a vector is its length in the Descartes–coordinate system.

Formally:

$$\|Vec\| = \sqrt[2]{Vec(x)^2 + Vec(y)^2}, Vec \in Vectors$$

where we denote the x and y component of a vector as Vec(x) and Vec(y) respectively.

2.2.2 Multiplication

$$(\times) : \mathbb{R}_{\geq 0} \times Vec \to Vec$$

The positive real parameter of the multiplication works by multiplying both x and y coordinates of the Vec parameter. Let us consider the example of

$$a \times v$$

where $a \in \mathbb{R}_{\geq 0}$, and $v \in Vec$. The result would be:

$$a \times v = Vec(a \times v(x), a \times v(y))$$

where $Vec(...)$ constructs the result with the multiplied x and y components of the original vector. Evidently, if $a \in (0, 1)$, then the magnitude of the vector will decrease, hence in that case, it can be considered as a division. If $a = 1$, then the result and the original vector is equivalent. If $a \in (1, \infty)$, then the resulting vector will be an enlargement of the original vector. Intuitively, if $a = 0$, then the operation reduces the original vector to the origin of the coordinate system.

Multiplying all points in a dataset keeps their relative distances.

As outlined in the previous part, we consider values that reside in the positive quadrant of a 2–dimensional coordinate system. Hence, it is important that the real number parameter of the multiplication operation is always greater or equal to 0. If we would allow $a < 0$, then the operation would change the residence of an original vector to the negative quadrant of the coordinate system, which would be inadequate for displaying the point.

In this paper, we will not define the operation of *translation* of points within the coordinate system, as translating a single point can change the relative locations regarding other points, as well as change the spread, mean, and other properties of the dataset that should not be changed during the data regeneration.

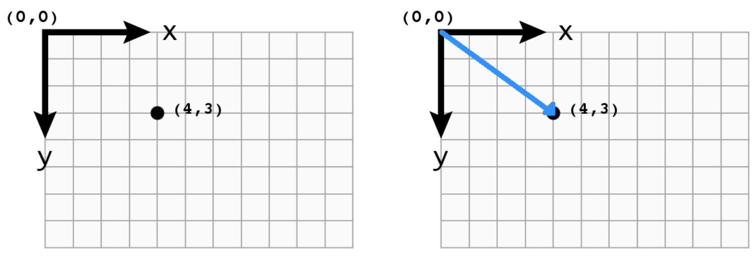


Figure 2: point vs. vector

2.3 Length of a set

The length of a set is defined by the number of elements within the set. We only consider *sets that contain the same type of elements.*

Formally:

$$\|set\| = \text{number of elements in the set}$$

$$\text{For example: } set := \{\alpha, \beta, \gamma\} \Rightarrow \|set\| = 3$$

where α, β, and γ are not sets.

2.4 Cluster

As this paper is about clusters, let us define what it considers as a cluster. Even visually explained, a *cluster* does not have a concrete definition, as its clarity depends on several parameters. To begin with, a cluster must contain at least one, but preferably a higher number of individual Vec vectors (defined above).

Formally:

$$\text{Let } C \in Cluster$$

$$C = \{Vectors : \|Vectors\| > 0\}$$

Even though the definition itself is adequate – as it states that a cluster is a non-empty set of points –, it does not tell much about the nature of a cluster – the only property we know is that a cluster must contain at least one Vec vector. Due to this, we need to state an assumption of the dataset. Let us assume that the original, fix dataset contains at least 30 points. The assumed number of points in the dataset is arbitrary, and can be any integer that is greater than 0.

However, the aim of this paper is to correctly find clusters in an original, truthful dataset, which usually has a much larger number of points. For the sake of clarity, this paper deals with fix datasets that have at least several hundred data points.

Each of the original datasets can be considered as a single cluster, however, due to the non-obvious shapes of a cluster, we need further definitions that explain the rules of how a cluster can be ”broken up” to sub–clusters. (A sub–cluster of a cluster is just a subset of points of the original cluster.)

Before we state the definitions that will help us define these sub–clusters, let us define a few statistical definitions.

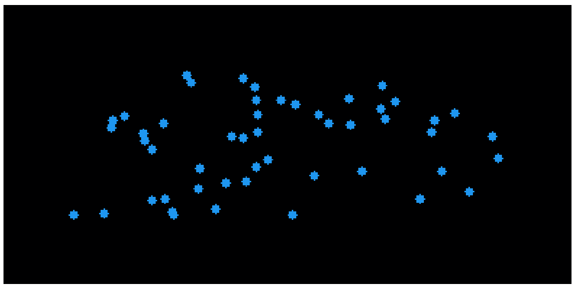


Figure 3: An example of a cluster – a dataset

2.5 Mean

The *mean* of a set of values is the sum of the values divided by the number of values in the set.

Formally:

$$\text{Let } set := \{x_1, x_2, ..., x_n\}$$

then, the mean value of the set is:

$$\bar{x} = \frac{\sum_{i=1}^{\|set\|} x_i}{\|set\|} \qquad \text{If n} := \|set\|\text{, then: } \bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

2.6 Centroid

A centroid is basically the mean of a dataset, where the points can have any dimensions – as long as all the data–points have the same number of dimensions.

We will define the centroid using vectors.

Let $\bar{v} \in Vec$ denote the centroid of a set of vectors (dataset). Then,

$$\bar{v} = \frac{Vec(\sum_{i=1}^{n} a_i, \sum_{i=1}^{n} b_i, ..., \sum_{i=1}^{n} z_i)}{n}$$

In the nominator of the equation above, the symbols $a, b, ..., z$ denote the number of dimensions of the data–points in the set.

We can use double indices to shorten the definition of the centroid:

$$\bar{v} = \frac{Vec(\sum_{i=1}^{n} \sum_{j=1}^{m} v_{i_j})}{n}$$

where $n = \|set\|$ and m is the number of dimensions.

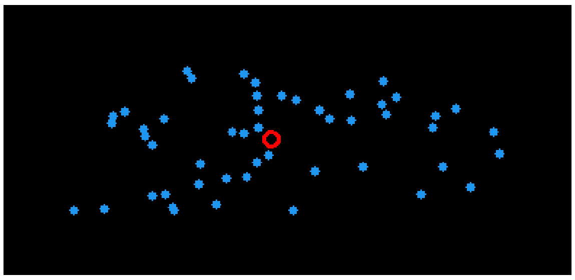


Figure 4: An example of a cluster and its centroid (marked with the red circle)

2.7 Altering the original dataset

The previously defined *centroid* is a very useful function when it comes to altering a dataset. As it was explained in the beginning of this chapter, the only dataset we let ourselves to alter (in the most minimal way possible) is the original dataset that we would like to regenerate.

By calculating the centroid of a dataset, we obtain a single vector, which can be considered as a point. By observing the magnitude of both coordinates of the result, we can approximate a *scaling factor* that we can multiply the original dataset with, in order to magnify or demagnify the entire dataset. This is especially useful when the points of a dataset are too large to be shown on the screen, and we want to find a scaling factor that is adequate for projecting the original points onto the field of view (the screen).

The centroid also has another powerful use that we will explain latter on in this paper in the data regeneration part.

2.8 Sample Standard Deviation

Standard deviation is a value that is used to quantify the amount of dispersion in a dataset. In other words, it is used to indicate how far does a data point fall from the centroid of the dataset.

If the standard deviation of a dataset is low – compared to the centroid of the dataset – then the dispersion of the data points is low as well, and most of the data points can be found close to the centroid. If, however, the difference between the centroid and the standard deviation is comparable to the magnitude of the centroid itself, then there is a significantly large dispersion of the data points around the centroid.

Formally:

Let $\bar{x}$ denote the centroid of a dataset. Then, the standard deviation is:

$$\sigma = \sqrt[2]{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

where n is the number of data points. Since we are dealing with a high number of population of data points, we rather use the *Sample Standard Deviation, which is:*

$$\sigma = \sqrt[2]{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Note that when we observe sub–clusters later on in this paper, the centroids and the standard deviations are always calculated on the sub–cluster which is being observed at that moment. Therefore, we only apply these operations on one cluster at a time.

Together with the centroid, the standard deviation of a cluster (or a set of data points) can be used in the data regeneration process to generate random data points around the centroid with the given standard deviation.

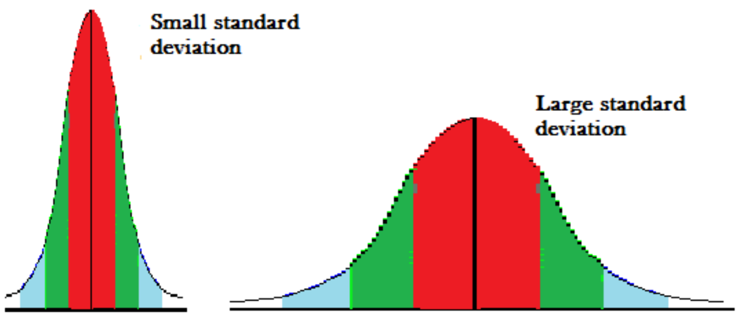


Figure 5: Example of high and low standard deviations [1]

2.9 Voronoi Cell and Voronoi Diagram

A *Voronoi Diagram* is the partitioning of a 2–dimensional plane into smaller sections, given a set of data points that lie on the plane. This definition by itself is much more complex, but is simplified for the purpose of understanding the relation of Voronoi Cells and the current problem of this paper. More can be read about *Spacialk Tessalation* in the *The International Encyclopedia of Geography* [10]. The partitioning procedure is the following:

Let us first choose $k \in \mathbb{N}^+ (k > 0)$ points randomly on the plane. For every data point P in the given dataset (a predefined set of *Points*), measure the distance between P and $k_i, i \in [1, k]$. P will correspond to k_i, if $min\{P - k_i\}$ – in other words, P will correspond to the closest k_i point. A P point in the dataset will always correspond to a k_i point.

Let $V \in Cells$ denote a partition on the 2–dimensional plane, where V is called a *Voronoi Cell*. Also, let V_{k_i} denote the marker of the i–th Voronoi Cell. A V cell is defined as a plane, where the boundaries are of the furthest data points which correspond to V_{k_i}.

On Figure 6 below, the V_{k_i} markers are presented as the black dots, and the V_k Voronoi Cells are the differently colored regions of the plane.

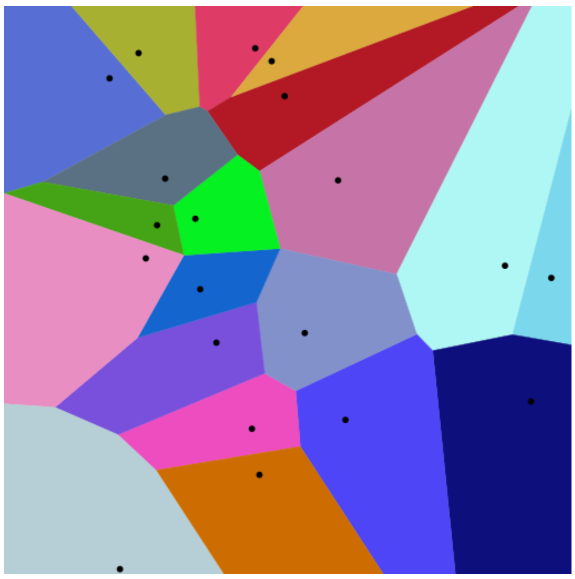


Figure 6: Example of a Voronoi Diagram [2]

3 Clustering Algorithms

Now that the basic definitions are stated, it is time to define the algorithms that will help create sub–clusters of a dataset, and with the help of the sub–clusters, regenerate the data.

3.1 K–means algorithm

The first algorithm this paper considers as a powerful clustering method, is the so called *k–means* algorithm. It is very closely related to the creation of the previously defined Voronoi Cells.

This algorithm requires two parameters: the first is the original dataset, the second is the k, the number of clusters (Voronoi cells) we would want for the algorithm to create. Since each V_{k_i} marker of a Voronoi cell is defined by the closest point to all points in the given partition, it can be considered as the "centroid" of the cluster. Even though the marker is not the centroid of the cluster as we defined it (mathematically) in the beginning of the paper, if the k is relatively high, the markers converge to the centroids of each partition. [11]

3.1.1 Steps

It is important to clarify that the k–means algorithm is a *non-deterministic* algorithm, meaning it can give several different results with the same parameters. The non-determinism comes from the fact that there are a few estimations and random number generations involved in the first few steps.

In the first step, the algorithm analyses the dataset, and estimates a uniform probability of position where most of the data points could possibly occur. This position is the initial marker of the entire dataset.

Starting with the initial marker, the algorithm additionally iterates $k - 1$ more times, further partitioning the 2–dimensional plane in each iteration.

With each new iteration, a new random point is chosen within the range of the data points and added to the list of markers. The newly chosen points are always different from the previously ones. It can be said that the markers of the k–clusters are defined in a set (where no repetition of a member is allowed). Then, the distances of all points are measured from all the markers, and reallocate their correspondences to the closest marker.

The iteration stops when the plane is divided into k partitions, and all the data points are corresponded to a cluster marker.

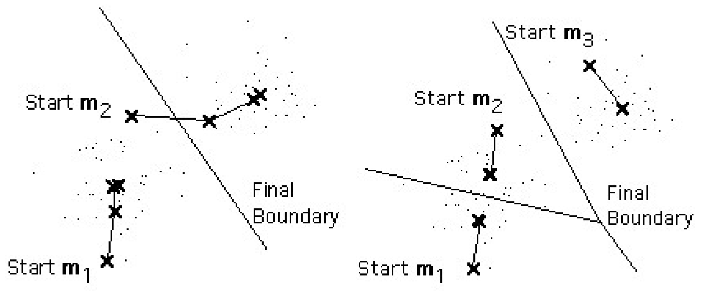


Figure 7: A step of the k–means algorithm initialization, from $k = 2$ to $k = 3$ [3]

The next step in the algorithm is to fine–adjust the created clusters, which is done by the following method:

For each cluster created in the *initialization* process of the algorithm, the centroids of the clusters are calculated, and the markers are moved to their corresponding centroids. Then, the distances are measured once again for all the data points, and assigned to the closest falling marker. At this stage of the algorithm, the data points might change their yield to a cluster, and might fall into a new, neighboring partition.

There are several different implementations of the initialization and fine–adjustment of the k–means algorithm. In this paper, the *K–means++* algorithm is outlined and used later on in the data regeneration.

Since there are a finite number of points in the dataset, the algorithm is expected to stop the fine–adjustment phase after a few iterations. However, since this paper does not consider to find a proof for this statement, it is important to outline a few methods that ensure the termination of the k–means algorithm.

3.1.2 Stop until unchanged means

The first method for granting termination is to continue the fine–adjustment process until the markers of the clusters are identical to the markers created in the previous iteration. If this condition is met, then it means that the algorithm is not capable of further adjusting the markers, therefore, the algorithm could be considered finished.

However, this paper would like to generalize the algorithms to as precise datasets as computationally possible. When dealing with very precise floating point numbers, the calculation of the centroids might cause uncertainties in the resulting values, and therefore the ever calculated markers in each iteration might change over–and–over again, even by a very small bit.

To avoid the extended running of the fine–adjustment – which is caused by the non–stopping deviation of certain markers – when checking for equivalence of the markers in the previous iteration, a small *neighborhood* should be considered rather than hard equivalence of the values. In other words, rather than checking the equivalence of the old and new markers, the difference should be analyzed between the old and the new markers. If the difference is relatively small, then the two markers can be said to be equal.

Even with the correction of analyzing the difference between the markers of each iteration, the fine–adjustment might take a very long time to converge to an acceptable value, therefore another, more efficient method is used.

3.1.3 The *Elbow method*

Similarly to the idea of checking the difference of magnitude between two markers, the elbow method measures the average distance of all the points in a given cluster to the marker to that cluster. In the beginning, it is expected the difference to be high, but converge to a small value throughout the iterations. This method along with others are explained in the book *Mastering Machine Learning with scikit-Learn* by *Gavin Hackeling*[12].

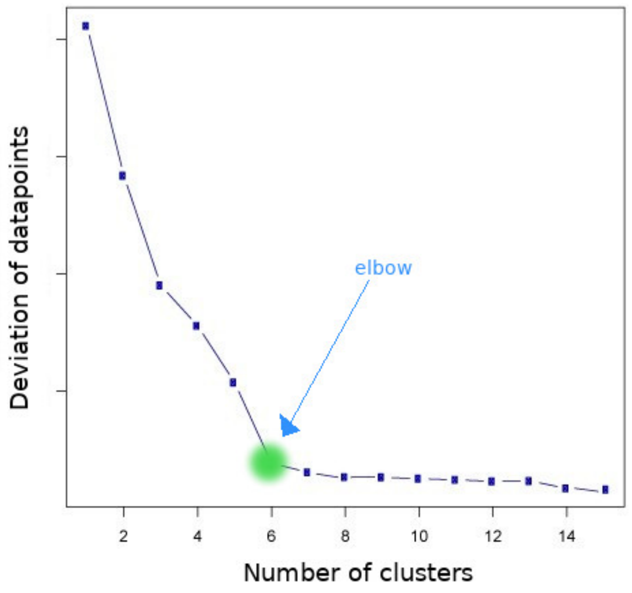


Figure 8: Graphing the Elbow method

Finding the elbow is not a straightforward task, as the data need to be analyzed as of how big the difference between the deviation of the data points of a cluster and their corresponding markers. These values should be compared in at least three neighboring iteration each time. For example, after three iterations, the value of the second iteration can be compared to its prior and successor values. If the value prior is much larger than the value of the next iteration by a certain scale (predefined by preference), then the algorithm has found the elbow value, and the iteration can be stopped.

Note that if the predefined scale used for finding the elbow value is too big, then the algorithm would detect all calculated differences as elbow values, and therefore stop very quickly – after a few (< 10) iterations. This might result in inaccurate fine–adjustments of the created clusters. On the other hand, if the the predefined scale is relatively small, then the algorithm would could run much further than it might need to. There are no real setbacks for the algorithm running longer, further fine–adjusting the clusters other than the loss of computational time.

In this paper, the first, (*corrected stop until unchanged means*) method is used to terminate the fine–adjustment of the k–means algorithm, however the *Elbow method* is considered to be a much better choice.

After the termination of the k–means algorithm, a group of k sub–clusters should have been created.

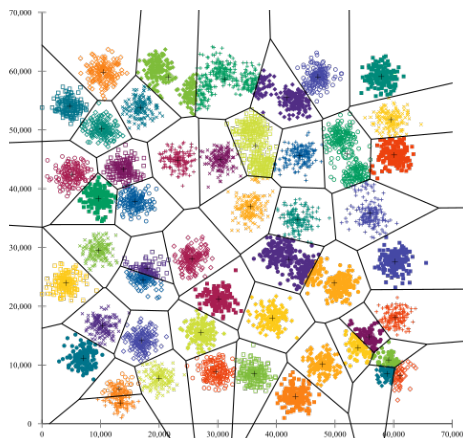


Figure 9: An example of a well–clustered dataset by the K–means algorithm [4]

3.1.4 Flaws of the k–means algorithm

It is easy to see that there are several problems with partitioning the plane as identifying clusters, rather than checking the relation between data points. For example, if the dataset contains concave–shaped clusters, then the k–means algorithm is not able to identify the containing data points as one cluster, but would rather partition the concave cluster into several more, identifying smaller sub–clusters. This is most problematic when there are several concave clusters neighboring each other – in this case, the k–means algorithm might create clusters that overlap more than one concave cluster from the original dataset.

Another problem of the k–means algorithm, is that it easily breaks apart clusters, which should be considered as one single cluster.

The following diagrams show the problems raised by using the k–means algorithm.

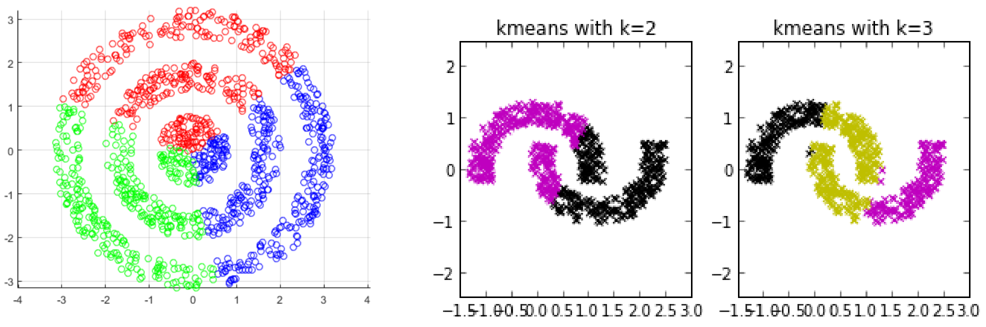


Figure 10: Problems when using the K–means algorithm [5] [6]

Figure 10 clearly shows the problems that might occur when using this clustering method. On the left diagram, we get a visual representation of the entire dataset, as well as the clusters created by the algorithm (in this case, $k = 3$). Intuitively, the three clusters would be made up of the two outer bolt–circle–shaped, and the middle circle–shaped sub–datasets. However, since the k–means algorithm does not take the relative distances between the data points into consideration – only the spatial partitioning – the clusters created are quite unintuitive.

The other two diagrams on the right side of Figure 10 show the faulty sub–clustering of the datasets.

Before explaining the process of data regeneration using the k–means algorithm, first, lets define another clustering algorithm.

3.2 DBSCAN algorithm

DBSCAN stands for *density-based spatial clustering of applications with noise.* In this algorithm, the spaces between each data points are considered to be the main aspect of cluster creation, rather than spatial partitioning. This allows us to identify concave–shaped clusters, without coinciding its neighboring clusters.

This algorithm thoroughly explained in the book called *Data & Knowledge Engineering* by *Birant et al.* [13]

3.2.1 Steps

The DBSCAN algorithm requires three parameters to work. The first is the entire dataset we would like to generate clusters for. The second parameter is an $\epsilon \in \mathbb{R}, (\epsilon > 0)$ number, which determines the density of how dense the data points should be to each other in order to consider them as one single cluster.

The third parameter is an $m \in \mathbb{N}^+, (m \geq 1)$ number, which stands for the minimum number of points that is needed for a set of data points to be considered as a cluster. If this minimum density of points is not met, then the data points in hand will be considered as noise, or outliers.

The first step of the algorithm is the arbitrarily choose a point from the dataset. The DBSCAN algorithm is a deterministic algorithm, therefore the choice of the starting point is irrelevant to the result of the clustering. With the same parameters, the same clusters will be created for a given dataset.

After a starting point has been chosen, the algorithm checks the neighborhood of ϵ radius of the point, and attaches those points to the starting point. In other words, a net of data points is created that are chained together if they reside within each others' neighborhood. If the thus created net contains as many data points as the minimum required, then a cluster is defined. If not, then the points are considered as outliers, and will not be a member of any cluster.

The algorithms runs the test (of checking the neighborhood of ϵ radius) for each and every data point in the dataset. If a new point is already attached to a cluster, then it does not need to be reattached again, however, it can expand the scope of the cluster itself by attaching points that were too far from its previously defined cluster.

Figure 11 shows the concept of the DBSCAN algorithm. On the diagram, point A is the starting point of the algorithm. Points B and C are the borders of the created cluster, that link to much less number of points than points in near the middle of the cluster. Since they are linked to a cluster that have a higher number of minimum points, they are considered part of the cluster. Point N is noise, which does not have enough neighboring points to belong to a cluster.

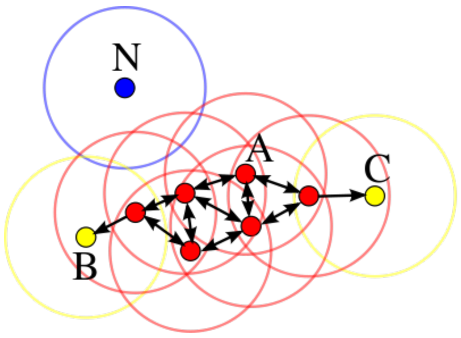


Figure 11: Concept of the DBSCAN algorithm [7]

With contrast to the k–means algorithm, there are no problems with the initialization process, and the shape of the clusters do not cause any problems of finding correct, and intuitive clusters within the given dataset. However, this algorithm have other kind of difficulties when it comes to data regeneration.

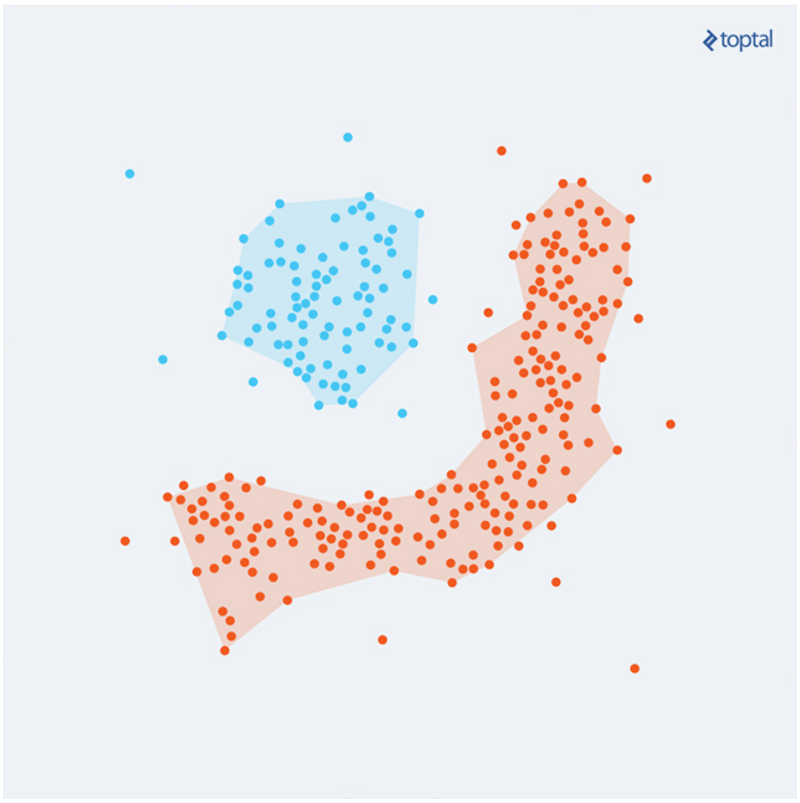


Figure 12: A correctly clustered dataset by the DBSCAN algorithm [8]

3.2.2 Problems with the DBSCAN algorithm

The main problem of the DBSCAN algorithm is finding a correct ϵ radius, and a correct m minimum number of points.

The minimum number of points can be any arbitrary low number if we would want to make sure that relatively small clusters can be created as well. There are, however, much bigger problems of finding a – so to say – "relatively good" ϵ radius, since there is no clear way of checking for smaller clusters and calculating an average of distance between the data points of a sub–cluster beforehand running the DBSCAN algorithm itself. This paper does not detail or consider the methods of finding a "good" value for ϵ, other than suggesting a trial–and–error process.

Another problem of this algorithm comes from the shape of the created clusters.

Since we would want to make sure that correct and truthful clusters are created when processing a dataset, we cannot neglect the fact that the clusters could have concave shapes. DBSCAN can correctly identify these clusters as well, but the problem arises when we would like to regenerate the result clusters.

In theory, there would be no other problems with this algorithm, however the speed of scanning of the neighborhood of a point can greatly vary based on the used ϵ. During the testing phase of the DBSCAN algorithm, it was found that there was a huge drop of performance and speed when the dataset contains very large numbers ($\approx 10^5$), and a large spread (more than ≈ 50 units). This problem did not occur when the magnitude of the dataset was reduced to the magnitude of 10^2. This is the main reason why the points are also represented as vectors, so that we can reduce the magnitude of these vectors without altering the shape of the dataset.

4 Introduction to *Data Regeneration*

In the previous sections, several methods of clustering a given dataset were outlined with both strengths and weaknesses of each algorithm.

In this section, we are going to introduce how to use the combination of these algorithms to regenerate a truthful dataset based on the original dataset.

The main tool of the data regeneration is to use the markers of the k–means algorithms, and generate points around it in some orderly fashion. For simplicity, this paper uses covariant normal distribution to generate these artificially created data. This paper does not go into detail of how the generation of covariant normally distributed points is achieved.

4.1 Combining the algorithms

The combined approach of clustering a dataset is based on the algorithms outlined in the previous part. The theory behind the use of combination is the following.

It was stated that the k–means algorithm is not adequate for identifying several clusters due to its bad approach of partitioning the plain of the dataset. The DBSCAN algorithm, however, overcomes this problem, and easily identifies clusters of any shape and size. When trying to regenerate a single cluster, we would like to use its centroid as the center of the covariant normal distribution – the point which we regenerate the artificial data around. For this reason, the result clusters of the DBSCAN algorithm is not good enough. For example, if a cluster has a ”Z–shape”, then the centroid of the cluster is somewhere on the middle, but if we tried to regenerate the cluster around the calculated centroid, then we would not get back the shape we wanted to.

The solution to this problem is to use the k–means algorithm on each and every cluster that we gained from the DBSCAN algorithm. Since spatially partitioning a single cluster only creates more precise sub–clusters, it will not cause a problem when trying to regenerate the cluster itself.

Therefore the idea behind the combined algorithm, is to separate a dataset into single clusters by using the DBSCAN algorithm, and then apply the k–means algorithm on each and every found cluster.

This way, we gain markers which lie on the most populated points of the original dataset, and therefore we can use these points alongside with the covariant normal distribution to regenerate artificial data around these locations.

5 Implementation

For the software of the combined algorithm (from now on referred as *Hybrid algorithm*) is implemented using C++ programming language due to its efficiency in large number of calculations, and the C++ OpenCV library. OpenCV includes already defined data types – such as 2–dimensional vectors – which are both adequate for storing the data points, and apply vector–methods on them.

The first step is to read the dataset that needs to be clustered. Assume, that the dataset is read from a text file, where each line of the file contains two floating point numbers, separated by spaces. These numbers define the x and y component of a vector. The dataset is read from the file and stored in an *std::vector*. At this point, the data is reduced so that the average magnitude of the data points is $\approx 10^2$. The reason for this is explained in the problems with the DBSCAN part of the paper, above this section of the paper.

The data is now ready to be processed by both clustering algorithms.

Let us see what happens in each case when regenerating the data with the clustering algorithms separately, and then with the hybrid method.

5.1 Regeneration with k–means

After the data is read and "shrinked", the vector which holds the data points is passed onto the k–means algorithm. The algorithm creates a *pair*, where the first member is a vector of the created clusters, and the second is a vector of cluster identification numbers. The vector of clusters is stored as a list of cluster id. numbers, where each data matches its point in the original dataset. For example, if the first member of the *vector of clusters* is 3, then the first data point belongs to the third cluster, and so on. Using this information, the clusters are stored as "real clusters" in a vector which can be used for easier display.

The data regeneration phase examines the clusters, and passes each cluster to the covariant normal distribution data generator one–by–one. The data generator calculates the centroid and the distribution of these clusters, and regenerates the data based on the calculated values.

The following diagrams show the results of the k–means data regeneration on two sample datasets.

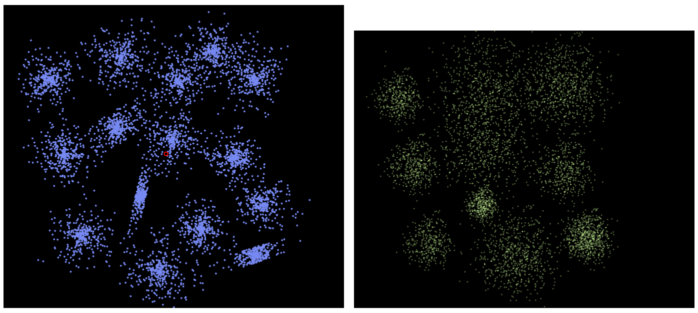


Figure 13: The original (*on the left*), and the regenerated cluster with k–means (*on the right*). k = 10

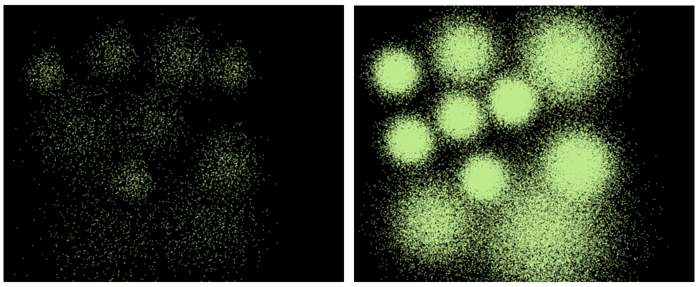


Figure 14: Regeneration with k–means, half the amount of data points (*left*) and 25 times the data point (*right*). k= 10

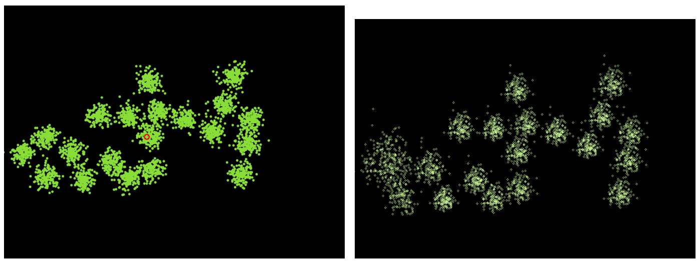


Figure 15: The original (*on the left*) and the regenerated cluster with k–means (*on the right*). k = 19

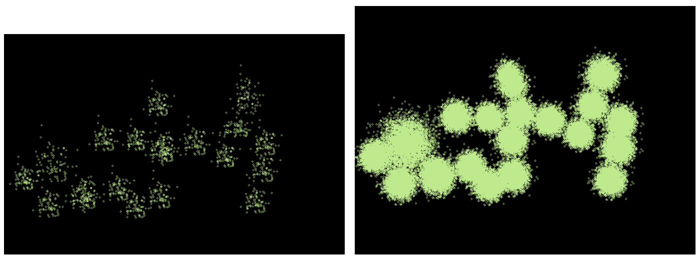


Figure 16: Regeneration with k–means, half the amount of data points (*left*) and 25 times the data point (*right*). k= 19

On Figure 14 we can see that the k–means algorithm could not completely detect the correct number of clusters, and hence, it was unable to regenerate a set of data which would resemble the original dataset. Figure 16 shows how the k–means algorithm is able to regenerate the original dataset, if it is able to find a ”good” number of clusters. However, it was previously outlined what happens, when the k–means has to cluster a concave–shaped dataset.

5.2 Regeneration with DBSCAN

The reading of the data happens the same way for the DBSCAN as for the k–means algorithm. In this case however, it is important to analyze the magnitude of the average data point, so that the dataset can be shrunk down. This is to avoid huge performance drops in the process of the algorithm.

After the algorithm is finished processing the original dataset, it returns a *vector of vectors* where the inner vector contains 2–dimensional data points. An inner vector stores the points of a single cluster, the outer stores all the clusters. Note that the outliers are *not* stored anywhere, as they do not belong in any cluster – therefore outliers are ignored during the regeneration process.

The following diagrams show the regeneration of the same dataset as the k–means algorithm, but this time with the DBSCAN algorithm.

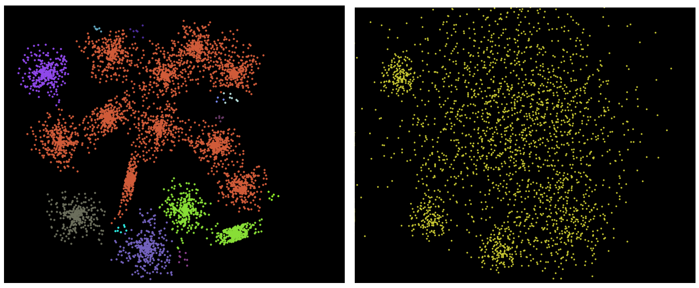


Figure 17: The original (*left*) and the regenerated cluster with DBSCAN (*right*).

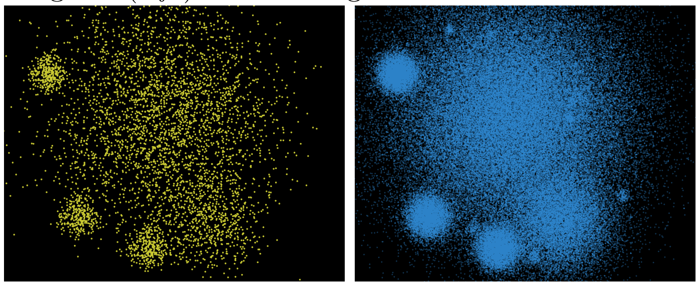


Figure 18: Regeneration with DBSCAN, half the amount of data points (*left*) and 25 times the data point (*right*).

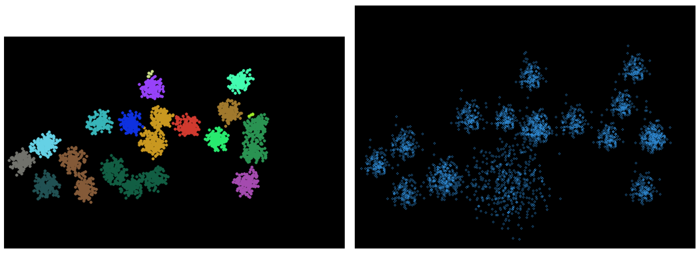


Figure 19: The original (*left*) and the regenerated cluster with DBSCAN (*right*).

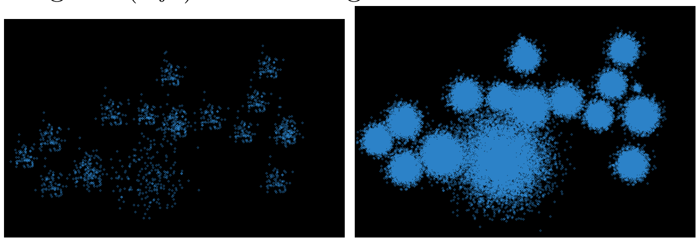


Figure 20: Regeneration with DBSCAN, half the amount of data points (*left*) and 25 times the data point (*right*).

The diagram on the left of Figure 17 shows the clusters found by the DBSCAN clustering. Each cluster has a unique color. It can be seen that there is a central cluster (marked orange), where the points were too close to each other to break up into several smaller clusters. On the regeneration figures this cluster is generated as one huge cluster with an enormous spread.

The idea of combining the two clustering algorithms arose from this state of the testing, since further clustering the ”bigger” cluster on Figure 17, can lead to a much more precise identification of its sub–clusters. The same case is shown on Figures 19 and 20 – if a concave cluster is found, then the regeneration becomes hard with using only the DBSCAN algorithm.

5.3 Regeneration with the hybrid algorithm

As detailed above in this paper, the hybrid algorithm first uses the DBSCAN algorithm to find all all possible clusters correctly, then sub–partitions each of the clusters by running the k–means algorithm on them.

Since both algorithms were explained above, we go straight to the results of the hybrid algorithm.

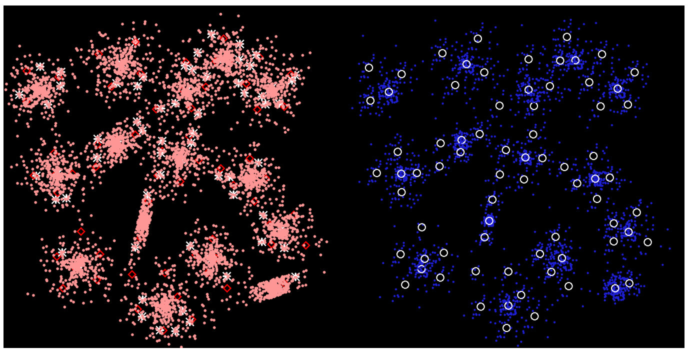


Figure 21: Regeneration half the data points using the hybrid algorithm.

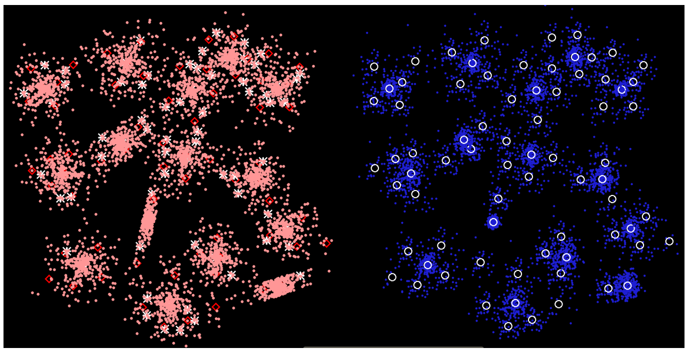


Figure 22: Regeneration of the original dataset using the hybrid algorithm.

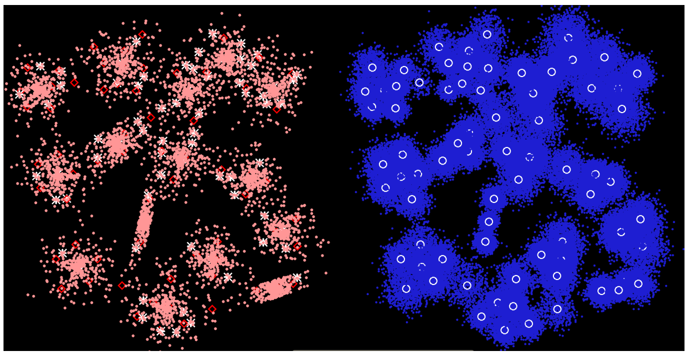


Figure 23: Regeneration 25x the data points of the original dataset using the hybrid algorithm.

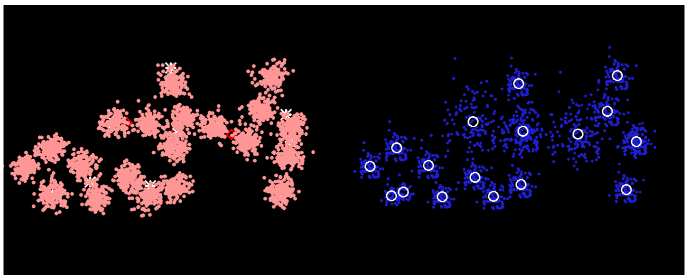


Figure 24: Regeneration half the data points using the hybrid algorithm.

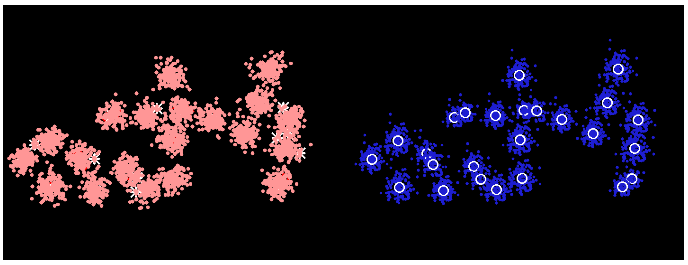


Figure 25: Regeneration of the original dataset using the hybrid algorithm.

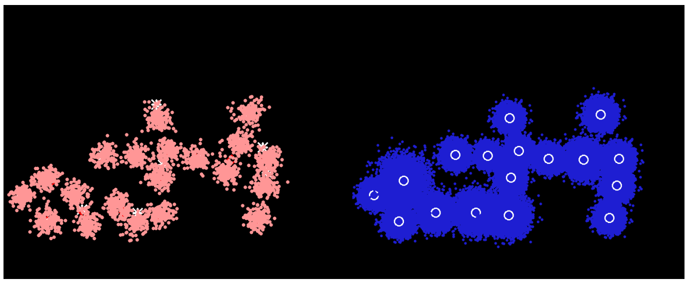


Figure 26: Regeneration 25x the data points of the original dataset using the hybrid algorithm.

The white circles on the regenerated dataset indicate the markers found by the k–means algorithm, which are the centers of the sub–clusters.

6 Conclusion

The regeneration of data from a given population of a dataset is neither an easy nor a straightforward task.

However, the clustering methods described in this paper can be used in combination to make this task much easier, and also give a satisfactory result. The DBSCAN is able to identify any clusters correctly – assuming that the parameters are "correctly chosen" – while the k–means algorithm is able to create sub–clusters that partition a sing cluster several times. This hybrid method creates sub–clusters that closely reflect the clusters of the original dataset. By observing the thus created clusters and their properties, the data can be regenerated using covariant normal distribution. The result of the regeneration gives artificial data – not violating legal rights – but also create a truthful dataset which can be further used in database algorithm optimizations.

6.1 Ideas for further development

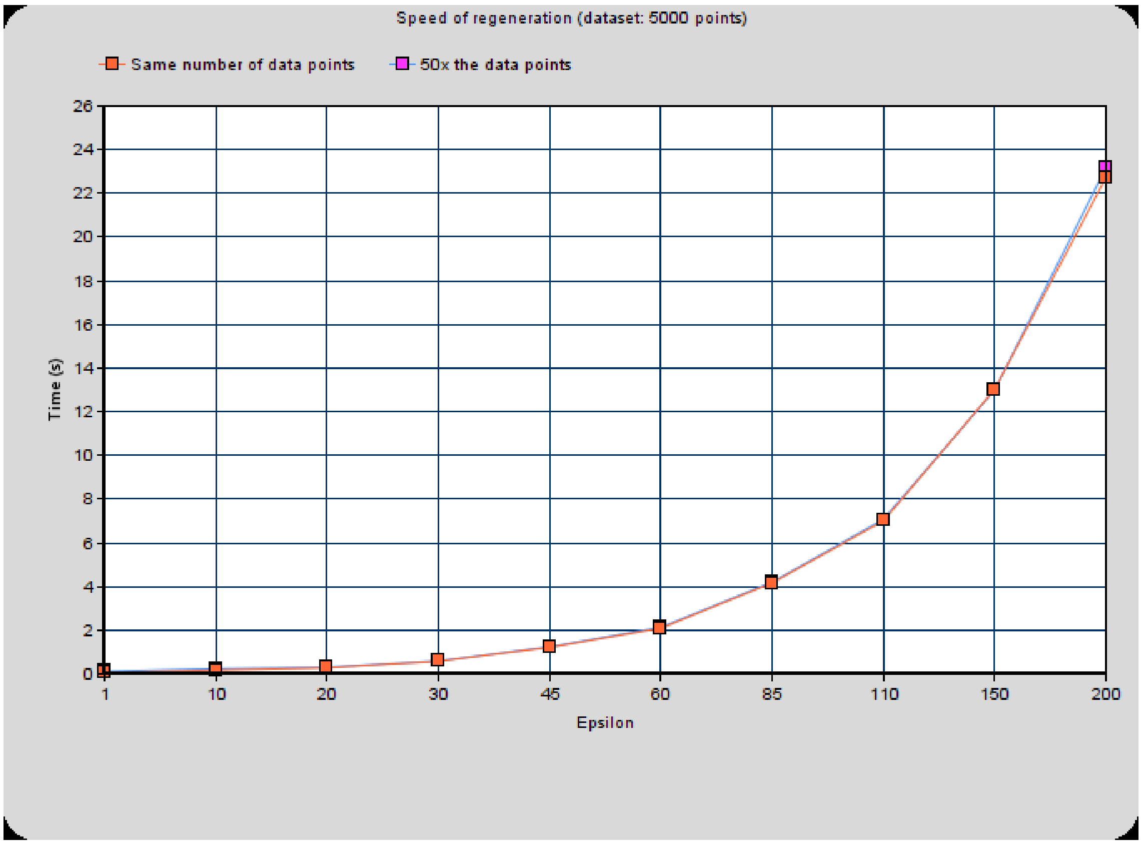
Even though some of the results are satisfactory, there are several parameters of the algorithm which should be further improved.

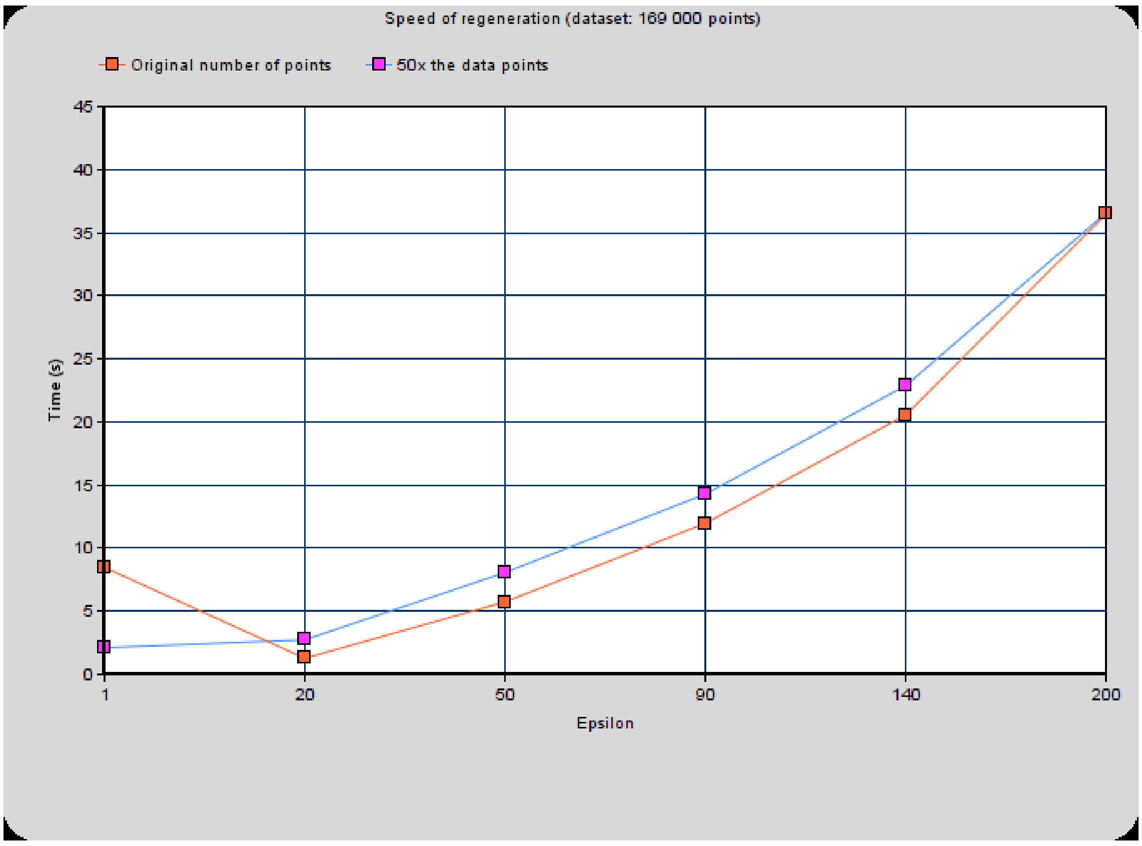
Further research should be made on how to find a correct ϵ and m minimal number of points for the DBSCAN algorithm. For the k–means algorithm, the *elbow method* should be used to find much optimal number of sub–clusters in its phase, instead of finding an already given number of clusters.

Currently, the algorithm works for 2–dimensional, numerical (floating point) data, in the magnitude range of $10^0 - 10^4$. Further data types should be considered, and hence augmenting the hybrid algorithm to higher, n number of dimensions.

6.2 Speed of data regeneration

The table below shows the time (in seconds) elapsed when regenerating a set of data under different parameters.




Speed of regeneration (dataset: 169 000 points)
Original number of points
50x the data points
Time (s)
0
5
10
15
20
25
30
35
40
45
1
20
50
90
140
200
Epsilon

References

[1] Standard deviation diagram. Illustration from: http://www.statisticshowto.com/average-deviation/.

[2] Voronoid cell image. Illustration from: https://en.wikipedia.org/wiki/Voronoi_diagram.

[3] A step of the k–means initialization process. Illustration from: https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html.

[4] Illustration from: https://stats.stackexchange.com/questions/133656/how-to-understand-the-drawbacks-of-k-means.

[5] Non-convex sets with k-means and hierarchical clustering. Illustration from: https://pafnuty.wordpress.com/2013/08/14/non-convex-sets-with-k-means-and-hierarchical-clustering/.

[6] Matlab kmeans clustering for non linearly separable data. Illustration from: https://stackoverflow.com/questions/40455334/matlab-kmeans-clustering-for-non-linearly-separable-data.

[7] Dbscan. Illustration from: https://en.wikipedia.org/wiki/DBSCAN.

[8] Clustering algorithms: From start to state of the art. Illustration from: https://www.toptal.com/machine-learning/clustering-algorithms.

[9] Mihály Szalay. Linear algebra. University Lecture, 2017.

[10] Atsuyuki Okabe. *Spatial tessellations.* Wiley Online Library, 1992.

[11] K Krishna and M Narasimha Murty. Genetic k-means algorithm. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 29(3):433–439, 1999.

[12] Gavin Hackeling. *Mastering Machine Learning with scikit-learn.* Packt Publishing Ltd, 2014.

[13] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data & Knowledge Engineering*, 60(1):208–221, 2007.