UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

BACHELOR THESIS No. 6985

IDENTIFICATION OF RECURRING PATTERNS FOR TEST BED MEASUREMENTS

Jan Čapek

Zagreb,

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

BACHELOR THESIS No. 6985

IDENTIFICATION OF RECURRING PATTERNS FOR TEST BED MEASUREMENTS

Jan Čapek

Zagreb,

UNIVERSITY OF ZAGREB
FACULTY OF ELECTRICAL ENGINEERING AND COMPUTING

Zagreb, 13 March 2020

BACHELOR THESIS ASSIGNMENT No. 6985

Student:	**Jan Čapek (0036509045)**
Study:	Computing
Module:	Computer Science
Mentor:	asst. prof. Marko Đurasević

Title: **Identification of recurring patterns for test bed measurements**

Description:

Describe testbeds and the problem of detecting faulty behavior that may occur while performing measurements. Study and describe the methods that can be used to detect patterns by which incorrect measurements can be determined. Implement selected pattern detection methods over a given set of measurement data and evaluate their effectiveness. Source codes, the results obtained with the necessary explanations and the used literature should be attached to the thesis.

Submission date: 12 June 2020

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

ZAVRŠNI ZADATAK br. 6985

Pristupnik:	**Jan Čapek (0036509045)**
Studij:	Računarstvo
Modul:	Računarska znanost
Mentor:	doc. dr. sc. Marko Đurasević

Zadatak: **Identifikacija ponavljajućih uzoraka kod mjerenja u testnim platformama**

Opis zadatka:

Opisati testne platforme te problem detekcije neispravnog ponašanja koje se može desiti tijekom obavljanja mjerenja. Proučiti i opisati metode koje se mogu koristiti za detekciju uzoraka pomoću kojih je moguće odrediti neispravna mjerenja. Ostvariti odabrane metode detekcije uzoraka nad zadanim skupom mjernih podataka te ocjeniti njihovu učinkovitost. Radu priložiti izvorne tekstove programa, dobivene rezultate uz potrebna objašnjenja i korištenu literaturu.

Rok za predaju rada: 12. lipnja 2020.

CONTENTS

1. Introduction

Identification of recurring patterns is useful in many cases for example in advertising where it helps to understand which group a person belongs to in order to target them with ads or in identification of fraudulent activities where frauds are outliers. Although many applications exist, this thesis will focus more closely on its application in car engine testing.

Testing of car engines is a very long process and it needs to be done in multiple stages. First, there is a calibration campaign where signals are measured for different starting states, and afterwards, engine's stability must also be tested which can take a few weeks. What would be useful is if engine's stability could be determined *a priori* from calibration campaign's tests. That is where recurring pattern detection comes in.

During tests, multiple signals are recorded at the same time. If patterns can be found in tuples of chosen signals, engine's stability might be determined by whether or not its state, defined by other signals not used as clustering parameters, is similar at every occurrence or most occurrences of some pattern.

This thesis will describe some clustering techniques, such as K-Means, DBSCAN and OPTICS which can be used to achieve detection of recurring patterns, and will compare them.

2. Data

As mentioned in an earlier chapter, multiple signals (temperature, rotation speed, cooling etc.) are recorded at the same time during testing. Those signals might not be recorded at the same frequency therefore all of them will need to be resampled to the common frequency in order to create time windows with the same number of samples for each of them.

Since a signal can contain a myriad of samples in a given time window it is not practical to use raw samples as point's coordinates for clustering. It would be more practical if each time-windowed signal can be described by some attribute e.g. mean of all the samples in a window. That would significantly decrease cluster computation time since the number of coordinates would be decreased by several magnitudes. That is not to say that a signal cannot be described by multiple characteristics like average derivation, mean, median etc. but in this thesis, for simplicity, a windowed signal will only be described by its mean.

That being said, data points will be in the form

$$p_i = (m_1, m_2, ..., m_N)$$

where m_n is a mean value of the n-th signal for an i-th time window and N is the number of time windows.

For the purposes of comparing and displaying results only two signals will be used because data points will then be 2-dimensional and easy to plot. Since the data is confidential, real names of the two signals cannot be disclosed and will instead be referred to as signals A and B, respectively.

The example data set has 7374 samples and its graph can be seen in **Figure 2.1**.

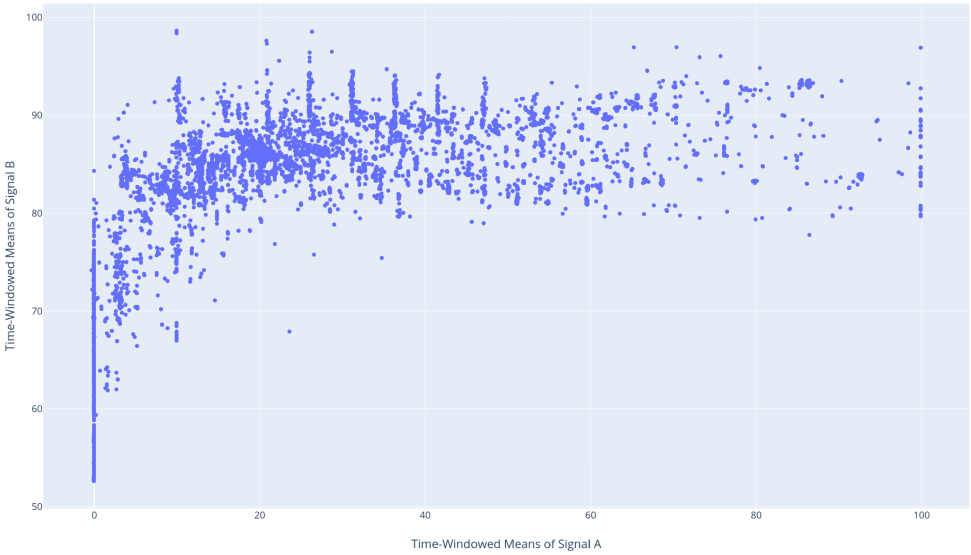

Figure 2.1: Example data set.

3. Clustering Techniques

3.1. K-Means

K-Means is the most widely used algorithm for data clustering. Like its name suggests, it groups data into K clusters by assigning the data to a cluster with the nearest mean.

Its simplicity and computational efficiency might be the exact reason for its popularity.

3.1.1. Algorithm

To not complicate the algorithm with more words than necessary, the pseudo code of the algorithm is outlined in **Algorithm 1**.

Algorithm 1 K-Means algorithm.

function KMEANS($K, data$)
  $centroids \leftarrow pickInitialCentroidsFromData(K, data)$
  $closest \leftarrow \{\}$ ▷ Dictionary of points associated to centroids
  loop
    for $point$ **in** $data$ **do**
      $closest[point] \leftarrow findClosestCentroid(point, centroids)$
    end for
    $nextCentroids \leftarrow calculateCentroidsBasedOnClusterMeans(closest)$
    if $nextCentroids = centroids$ **then** ▷ Convergence point reached
      return $closest$
    end if
    $centroids \leftarrow nextCentroids$
  end loop
  return $closest$
end function

Now that the pseudo code for the algorithm is laid down, let us describe what it does. The above function returns a dictionary $closest$ which assigns the closest centroid to each key (data point) .

The algorithm starts by picking initial centroids and starting a loop. In the loop, to each data point the nearest centroid, which defines a cluster, is assigned. Then, the cluster centroids are recalculated based on the means of the data in them. The loop is repeated until the convergence point is reached i.e. the centroids have not changed after recalculation.

pickInitialCentroidsFromData picks K random points from $data$ and decides that those points will be initial centroids.

findClosestCentroid finds the closest centroid to the current $point$. In order to determine which centroid is the closest, first, we must determine how the distance between the points is calculated. Most commonly, Euclidean distance is used. There are more ways, but that is out of the scope of this thesis.

Euclidean distance between points A and B is calculated by the following formula

$$d(A, B) = \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_N - b_N)^2}$$

where a_n and b_n are coordinates of A and B, respectively.

Of course, the goal of the algorithm is to minimize the distance d of clusters' members to their centroids (means) therefore, since for all the points p it is true that $p \in R^N$, minimization can also be done with d^2 so as to not unnecessarily compute the square root in order to save computational time. With that being said, the formula which will be used in the minimization process is

$$d^2(A, B) = (a_1 - b_1)^2 + (a_2 - b_2)^2 + ... + (a_N - b_N)^2$$

or more formally

$$d^2(A, B) = \sum_{i=1}^{N} (a_i - b_i)^2$$

calculateCentroidsBasedOnClusterMeans calculates, as function's name suggests, new centroids based on the current cluster means. To calculate a new centroid for a cluster, the function calculates the mean value of all the points currently in that cluster and that mean value becomes the new centroid of the cluster.

3.1.2. Challenges

Optimal Number of Clusters

The first challenge is choosing an optimal K for the data at hand. One of the methods to overcome that is running the algorithm many times but with an increased K each time and then choosing K after which the increase in quality of groups is insignificant. This method is called an elbow technique. Although there are more methods which can be used to solve this problem, here, only the elbow method will be described.

Elbow method is a heuristic method for finding an optimal number of clusters. The main principle of the method is to calculate the score of a clustering result for each $K \in \{2, 3, ..., k_{max}\}$ and find K for which a point of diminishing returns is reached. For all Ks after that point an increase in result quality will be insignificant. Most commonly, a score of a K-Means clustering result is represented by the ANOVA[1] F-Test. If the score is big that means that the variance between groups is much larger than the variance within groups which is a good thing because the goal of clustering is to group similar data, and if the score is low that means that the variance in the clusters is large which means that clusters are chosen poorly.

$$F = \frac{between\ group\ variance}{within\ group\ variance} = \frac{\frac{\sum_{k=1}^{K} n_k * (\mu_k - \overline{X})^2}{K-1}}{\frac{\sum_{k=1}^{K} \sum_{i=1}^{n_k} (X_{ki} - \mu_k)^2}{N-K}}$$

where K is number of clusters, n_k is number of samples in k-th cluster, μ_k is the mean value of k-th cluster, $\overline{X}$ is the mean of all data samples, X_{ki} is i-th sample in k-th cluster and N is overall sample size.

Non-Static Result

Since initial centroids are chosen randomly, the output of the algorithm for a fixed data set will be different each time which means the result sometimes might not be optimal. This can be solved by performing many runs of the algorithm and then choosing the best result. Results can also be compared by performing an F-Test on them.

Flat Geometry

The third problem for which there is no solution is that K-Means only works with flat geometry which means that if the data should be clustered by its shape rather than

[1] ANOVA is an acronym for "analysis of variance".

distance, K-Means is not suitable for that and another algorithm must be used. This would be the case if the plotted data looks like concentric circles and each circle should be one cluster.

3.2. DBSCAN

Density-Based Spatial Clustering for Application with Noise (DBSCAN) is a clustering algorithm which, as opposed to K-Means, calculates clusters based on the data density rather than closeness. That allows clusters to take any number of shapes instead of only circular ones, and solves the *flat geometry* issue of K-Means.

This algorithm's core attribute is that it deals with noise really well. The algorithm acknowledges that some samples are so different from others that they should not belong to any group so it labels them as outliers or noise.

The algorithm requires two parameters, the ϵ and N. ϵ parameter is a radius within which points are considered neighbors. The N parameter defines a minimum number of neighbors a point needs in order for it to be considered *core*.

DBSCAN algorithm defines three different types of points: *core*, *border* and *noise*.

Core point is a point which has at least $N-1$ neighbor points around. Not all points are considered neighbors, only the ones which are not further than ϵ are considered as such. If one core point is neighboring another, they belong to the same cluster.

Border point is a point which does not have at least $N-1$ neighbors but it has a neighbor which is a core point. That makes it a border point and it still belongs to a cluster. If a border point is neighboring with more than one core point its cluster is the same as the first core point found.

Noise point is just that, noise. It does not belong to any cluster since it does not satisfy the requirements to be a core point, nor a border point.

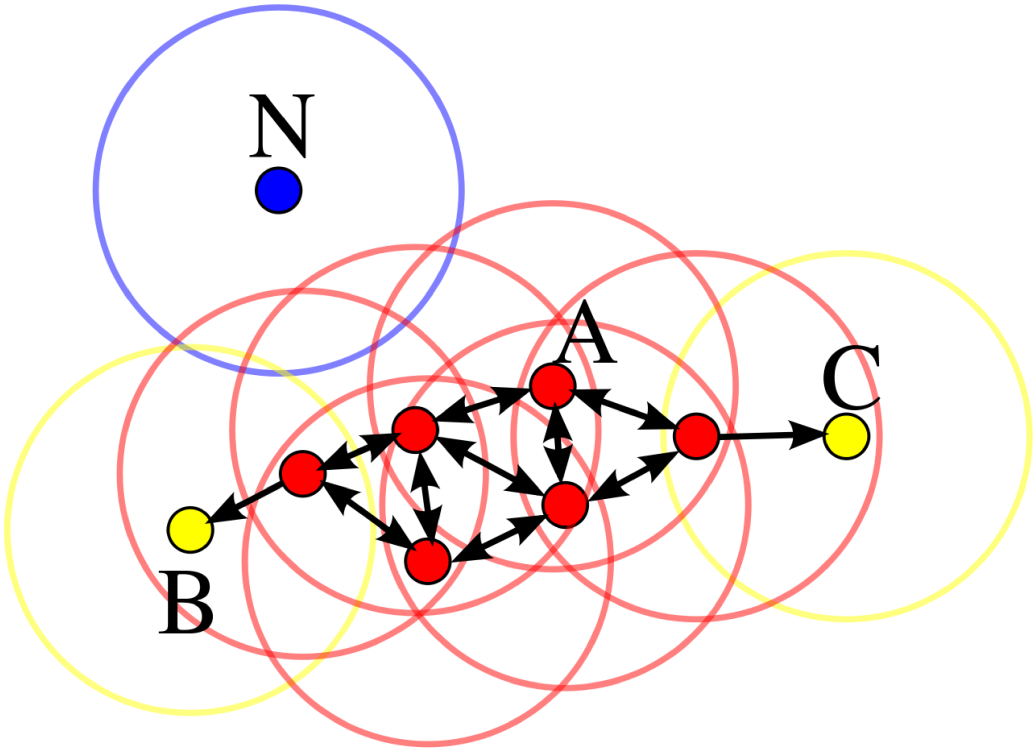

Figure 3.1: Illustrated core, border and noise points of DBSCAN for $N = 4$. By Chire - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=17045963

The points are very well illustrated in **Figure 3.1** for $N = 4$. There it can be seen that the A is a core point since it has 3 points within the radius ϵ, and all other red points are core points as well since all of them have at least 3 neighbors. Points B and C are border points because they have less than 3 neighbors but one them is a core point. The point N neither has at least 3 neighbors nor one of its neighbors is a core point therefore it is a noise point.

3.2.1. Algorithm

The algorithm is not conceptually complicated but its efficient implementation can be. The point of the thesis is not how to implement an algorithm efficiently but to explain how the algorithm works hence the pseudo code below will not be the most efficient one nor it reflects real implementations of DBSCAN.

Algorithm 2 DBSCAN algorithm.

function DBSCAN($\epsilon, N, data$)
    // Find all core points.
    for $point$ **in** $data$ **do**
        if $hasAtLeastKNeighbors(point, \epsilon, N-1)$ **then**
            $markPointAsCore(point)$
        end if
    end for

    // Assign clusters to core points.
    $currentClusterIndex \leftarrow -1$
    for $point$ **in** $getCorePoints(data)$ **do**
        if $isAlreadyAssignedToACluster(point)$ **then**
            $continue$
        end if
        $currentClusterIndex \leftarrow currentClusterIndex + 1$
        $assignClusterIndexToPoint(point, currentClusterIndex)$
        for $corePoint$ **in** $recursivelyReachableCorePoints(point, data, \epsilon)$ **do**
            $assignClusterIndexToPoint(corePoint, currentClusterIndex)$
        end for
    end for

    // Go through all non-core points and distinguish border points from noise.
    for $point$ **in** $getNonCorePoints(data)$ **do**
        $corePointNeighbor \leftarrow getCorePointNeighbor(point, data, \epsilon)$
        if $corePointNeighbor$ **is** $None$ **then**
            $markPointAsNoise(point)$
            $continue$
        end if
        $assignClusterIndexToPoint(point, clusterIndexOf(corePointNeighbor))$
    end for

    // Return map where to each point its cluster index is assigned (excluding noise).
    return $getPointToClusterIndexMap(data)$
end function

The pseudo code is split into three for loops. The first for loop marks points with at least $N - 1$ neighbors as core points. The second loop iterates through all the core points, and if the point is not yet assigned to a cluster, it will be assigned to a new one. Additionally, after a core point is assigned to a new cluster, all core points directly or indirectly (through other core points) reachable from it will also be assigned to the same cluster. A point is reachable from another if they are not further than ϵ. The third loop iterates through all non-core points and determines if they are border, and belong to some cluster, or noise points.

Now that the core of the algorithm is defined, some non-straightforward functions used in the pseudo code will be explained in the following paragraphs.

recursivelyReachableCorePoints returns a list of all the core points which are neighboring with the point itself or with its neighbors or their neighbors etc. This is used so each core point can be assigned to the same cluster as its core point neighbor.

getCorePointNeighbor returns the first core point neighbor it can find for the point given. If the point does not have a core neighbor then it returns $None$. If this method returned $None$ that means the point is a definitely a noise point and does not belong to any cluster.

getPointToClusterIndexMap returns a map of points with assigned cluster indices to which they belong to. Noise points can be included in the map or left out completely. That is entirely up to the implementation. Of course, if noise points are included in the map then they need to be assigned to a special value so that they are not confused with the non-noise ones.

3.2.2. Challenges

Data Sets With Drastic Differences in Density

DBSCAN clusters data by using fixed ϵ value. That works well when data has somewhat consistent density, but if the opposite is the case, then less dense regions could be considered as noise rather than valid clusters but less dense.

Optimal Parameters

Parameter N, as well as ϵ, depends mostly on the domain of the data. It often requires a person to be familiar with the domain in order to decide which values are acceptable.

In contrast to N, there have been some techniques proposed to find an optimal ϵ. Here, only one will be covered.

Finding an optimal ϵ could be achieved by plotting a K-distance plot and choosing as an epsilon value a distance at the maximum curvature. This method is entirely a rule of thumb method, and it might not work really well on all data sets. The method begins by calculating the minimal distance to k-th, k should not be confused with the N, neighbor for all points in the data set. Then, distances are sorted in an ascending order and plotted against their own index. So, distances will be on y-axis and their indices on x-axis.

3.3. OPTICS

Ordering points to identify clustering structure (OPTICS) is very different from K-Means and DBSCAN algorithms. It is not a clustering algorithm *per se* because its output is not really a clustering result right away but a **reachability distance graph**. After the distances are found it uses DBSCAN to actually cluster the points or the points can be manually selected by looking at the graph.

Actually, the algorithm is quite similar to DBSCAN. It also requires a minimum number of points N in order to identify core points, but the situation with ϵ is a bit different. ϵ is not really required but usually is asked for by the libraries to reduce algorithm's time complexity. Contrary to DBSCAN, OPTICS does not assume constant density of data. It acknowledges that some clusters are more dense than others.

The algorithm defines two terms: *core distance* and *reachability distance*.

Core distance is the minimal radius such that a point is considered a core point. Within this radius a point must have at least $N - 1$ neighbors. If the radius is greater than the ϵ or a point simply does not have $N - 1$ neighbors, core distance is undefined.

Reachability distance reachability distance of point Q to P is given as

$$max(CoreDistance_P, d(P, Q))$$

where $d(P, Q)$ is the distance between points Q and P. If $CoreDistance_P$ is undefined then the reachability distance from Q to P is also undefined.

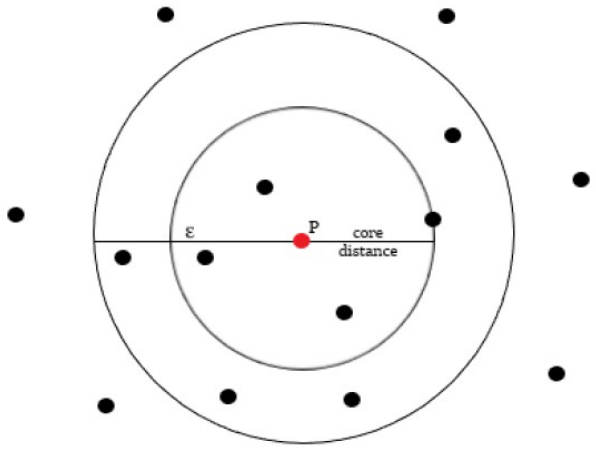

Figure 3.2: Core distance illustration for $N = 5$.

Figure 3.2 illustrates the core distance of a point P for $N = 5$. The point's core distance is the distance from P to its fourth nearest neighbor. If P's fourth neighbor was further than its ϵ neighborhood (illustrated by the outer circle) its core distance would be undefined, and therefore any reachability distance from any other point to P would also be undefined.

For any point inside the inner circle, the reachability distance to P is P's core distance. For all points outside the inner circle, the reachability distance is the actual distance to P (usually Euclidean).

3.3.1. Algorithm

As previously mentioned, OPTICS does not return clusters, instead, it returns points ordered by reachability which can then be plotted and afterwards clusters can be found by performing a DBSCAN on the data with ϵ chosen according to the reachability plot. On the plot clusters can be identified by the drops in the reachability.

It must be noted that points are not exactly ordered by the value of their reachability, instead they are ordered both by reachability as well as the order they are processed in.

The pseudo code of the algorithm is outlined in **Algorithm 3**.

Algorithm 3 OPTICS algorithm.

```
function OPTICS(ϵ, N, data)
    orderedPoints ← {}                          ▷ Ordered set of processed points.
    for point in data do
        if isProcessed(point) then
            continue
        end if
        markAsProcessed(point)
        addToOrderedSet(point, orderedPoints)
        if coreDistance(point, ϵ, N, data) is not None then
            minHeap ← {}          ▷ Min-heap used to order points by reachability.
            neighbors ← getNeighbors(point, ϵ, data)
            updateHeapWithEpsNeighborhood(point, minHeap, neighbors, ϵ, N)
            for other in minHeap do
                minHeap.remove(other)
                markAsProcessed(other)
                addToOrderedSet(other, orderedSet)
                if coreDistance(point, ϵ, N, data) is not None then
                    neighbors′ ← getNeighbors(point, ϵ, data)
                    updateHeapWithEpsNeighborhood(other, minHeap, neighbors′, ϵ, N)
                end if
            end for
        end if
    end for
    return orderedPoints
end function
```

Algorithm 4 Update min-heap function.

```
function UPDATEHEAPWITHEPSNEIGHBORHOOD(point, minHeap, neighbors, ε, N)
    coreDistance ← coreDistance(point, ε, N, data)
    for neigh in neighbors do
        if isProcessed(neigh) then
            continue
        end if
        newReachabilityDistance ← max(coreDistance, d(point, neigh))
        if reachabilityDistanceOf(neigh) is None then
            setReachabilityDistance(neigh, newReachabilityDistance)
            minHeap.insert(neigh)
        else if newReachabilityDistance < reachabilityDistanceOf(neigh)
then
            setReachabilityDistance(neigh, newReachabilityDistance)
            minHeap.moveUp(neigh)
        end if
    end for
end function
```

In the algorithm it is assumed that, at the start, reachability of all the points is $None$ or undefined. The algorithm consists of the two main nested loops.

In the outer loop we iterate through all unprocessed points and each point we mark as processed, add it to the ordered set which is the result of the algorithm and, if the point has a defined core distance, we get its neighbors, add them to an empty min-heap where neighbors are ordered by their smallest reachability distance to their already processed neighbor.

The inner loop iterates through the points in min-heap, and basically does the same thing as the outer one, but it does not construct a new min-heap, instead it uses the already constructed one. The most important thing to note here is that the inner loop iterates through points sorted by their reachability whereas the outer loop iterates through point in order as they were provided.

By far, the most important helper function of this algorithm is ***updateHeapWithEpsNeighborhood*** described in **Algorithm 4**, and it deserves a special explanation. The function iterates through the unprocessed neighbors of the point provided. The neighbors are checked if their old reachability, reachability distance to some other processed point, is greater than the distance to the point provided. If that is true, their

reachability is updated with the new one and their place in the min-heap is updated. Of course, if the old reachability was $None$, which would mean that the point has never even been reached until then, it would be updated with the new one and the point will be inserted in the heap.

The result of the algorithm is an ordered set of point sorted by their reachability and the point's reachability distance can be retrieved with $reachabilityDistanceOf(point)$ which can then be plotted and from the plot clusters can be determined.

3.3.2. Challenges

Although OPTICS solves same problems DBSCAN has, most prominently it does not need ϵ parameter, it still requires minimal neighbors parameter which still greatly depends on the data domain.

OPTICS is still not a miracle clustering algorithm which groups data perfectly, actually it does not group them, but outputs their ordering by reachability. It still requires manual intervention which points make up which clusters. That intervention can be performed manually either by handpicking points or with DBSCAN with appropriate ϵ parameter.

4. Results

This chapter will show actual results of the aforementioned algorithms on the sample data set.

4.1. K-Means

An F-Test score plotted against number of clusters can be seen in **Figure 4.1**. It is clear from the graph that the optimal number of clusters for presented data is 9 since the scores are not a lot higher for each additional cluster and is in fact much lower for $K = 13$ and after.

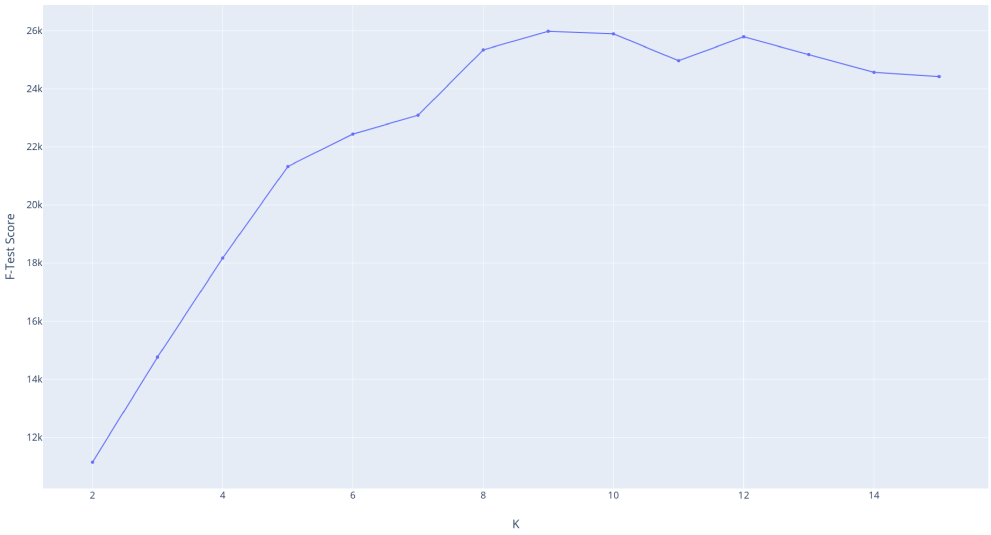

Figure 4.1: F-Test scores plotted against number of clusters.

For $K = 9$ a result can be seen in **Figure 4.2**. Some groups, especially the light green one, have a very small area whereas the dark blue cluster is visibly the largest. That is because the samples in the green cluster are very dense but in the blue one they are very spread out.

Figure 4.2: Result of K-Means clustering with 9 clusters.

4.2. DBSCAN

Following the method for finding an optimal ϵ, from the graph in **Figure 4.3**, the optimal value for ϵ is around 0.65. Distances in the graph are from points to their first neighbor because for any following neighbor, ϵ just was not really good for the data set used.

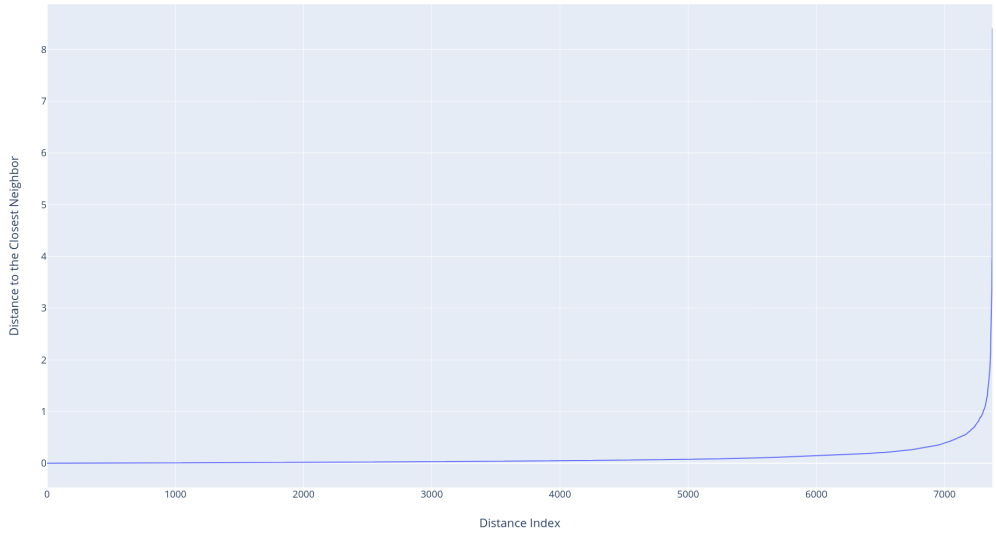

Figure 4.3: Distances to the nearest neighbor.

The clustering result for $N = 10, \epsilon = 0.65$ the graph can be seen in **Figure 4.4**. The result has 71 clusters. Since the color palette used to color clusters is smaller than the number of them, some colors are repeated. The points painted black are noise points.

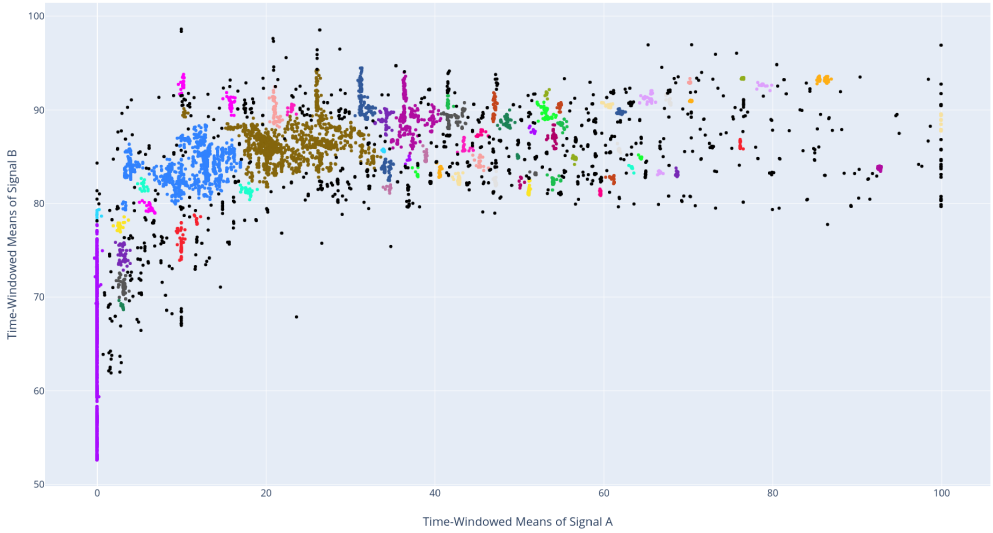

Figure 4.4: DBSCAN clustering result for $N = 10, \epsilon = 0.65$ with noise.

The exact graph can be seen in **Figure 4.5** but without noise. Here, it can really be seen how very well DBSCAN handles noise.

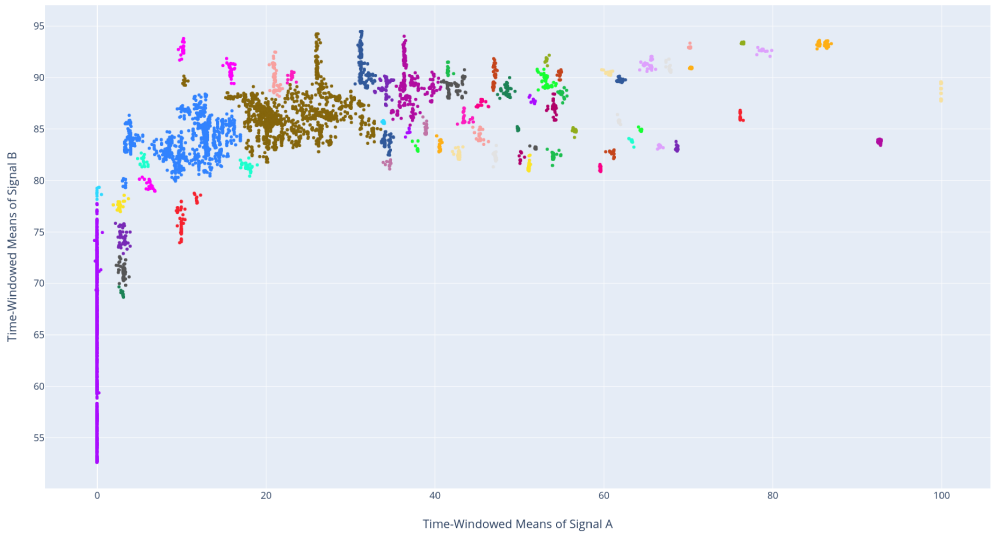

Figure 4.5: DBSCAN clustering result for $N = 10, \epsilon = 0.65$ without noise.

From the graph presented, it can also be seen how the groups are smaller than the groups of K-Means. Also, they take all kinds of shapes.

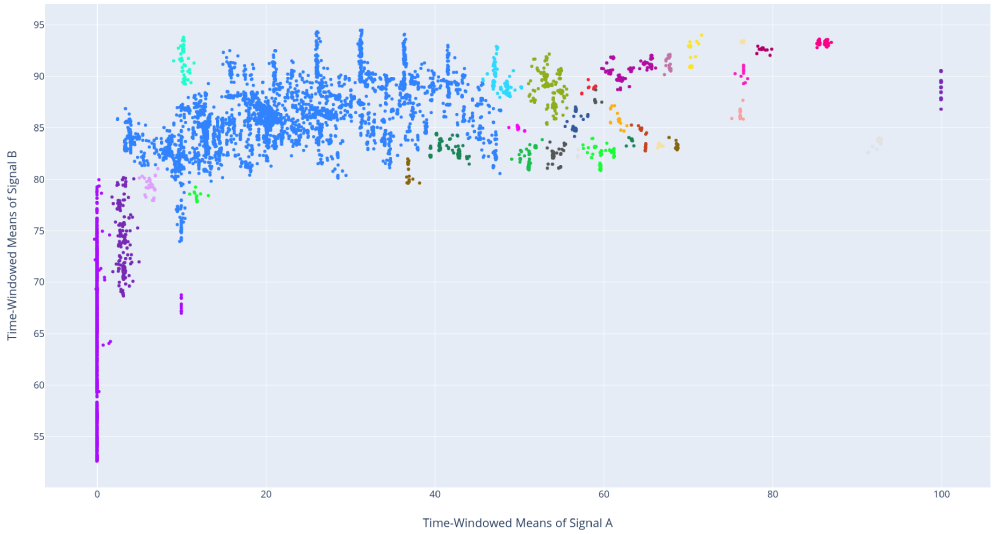

Figure 4.6: DBSCAN clustering result for $N = 10, \epsilon = 1.0$ without noise.

Figure 4.6 shows what happens when the ϵ value is too large. The blue cluster is much bigger than all the other ones.

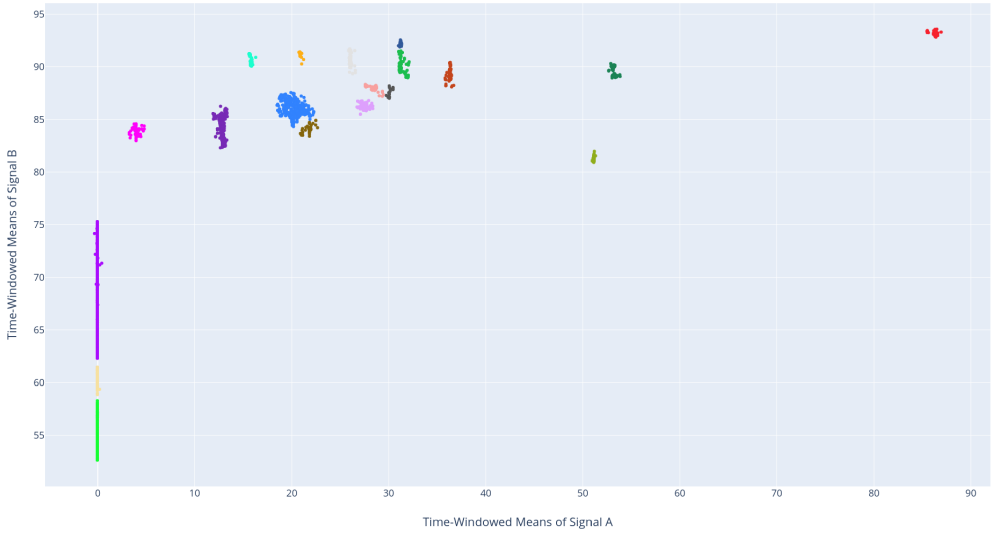

Figure 4.7: DBSCAN clustering result for $N = 50, \epsilon = 0.65$ without noise.

If the value for the parameter N is larger, with the ϵ value being fixed, found clusters will be more dense. The graph for $N = 50$ and $\epsilon = 0.65$ can be seen in **Figure**

4.7. Of course, not only that the clusters are more dense but there are also fewer of them since only denser areas will not be considered as noise. The mentioned graph only contains 19 clusters as opposed to 71 shown in **Figure 4.5**.

4.3. OPTICS

As already mentioned, the output of OPTICS are not really clusters but points ordered by reachability distances. When the result is retrieved, the distances can be plotted and clusters can then be determined as valleys in the graph.

In **Figure 4.8**, the exact graph can be seen for the mentioned data set with parameters $\epsilon = 2.5$ and $N = 10$. Since a lot of clusters can be found in the graph, as index gets higher, clusters are smaller and the valleys cannot so easily be seen but the first few valleys are very well identifiable and they represent bigger clusters. The infinity in the graph means that the distance is greater than ϵ.

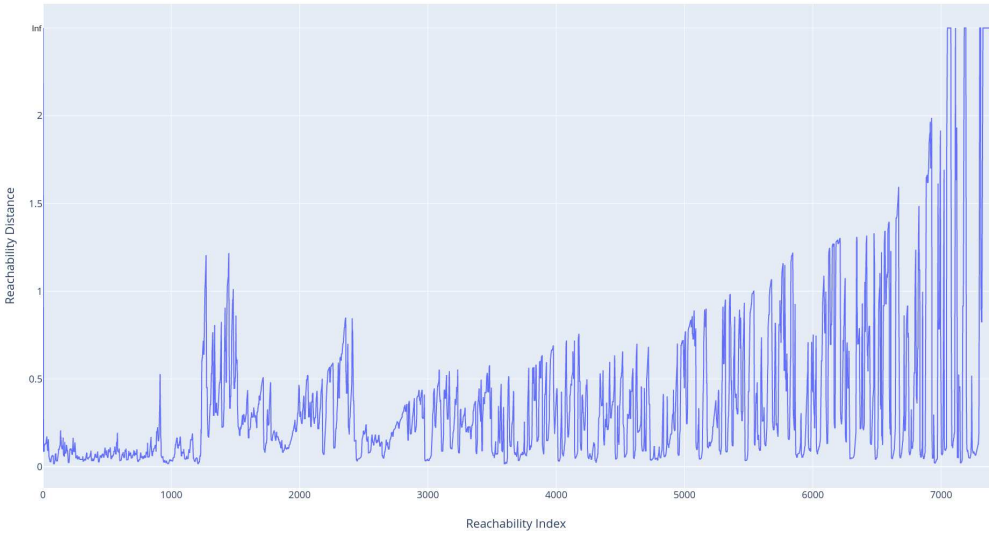

Figure 4.8: Reachability graph for the data set.

Points with reachability distances below the line in **Figure 4.9** are considered non-noise points. All the points with the distances continuously below the line belong to the same cluster. As soon as one point has a distance greater than the line, the next point with distance below the line will belong to the new cluster.

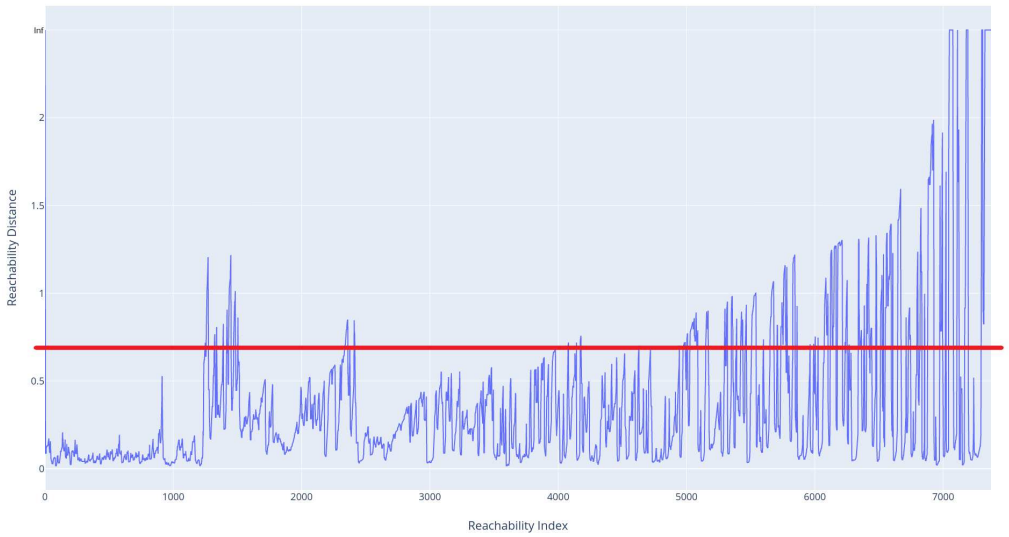

Figure 4.9: Reachability graph for the data set with line at $\epsilon \approx 0.65$.

The line in the above graph is set at $\epsilon \approx 0.65$ to show how the clusters in the DBSCAN's result would look like in the reachability graph.

If the line is lowered to $\epsilon \approx 0.45$, we can see that the first valley is split into two valleys which results in splitting that cluster into two. The mentioned cluster is the first one from the left colored purple in **Figure 4.5**. The line on the reachability graph can be seen in **Figure 4.10** and the clustering result using DBSCAN with parameters $\epsilon = 0.45$ and $N = 10$ in **Figure 4.11**. The new result has 100 clusters different clusters, and it can be seen how the mentioned purple cluster was split into two, one purple and another dark green cluster.

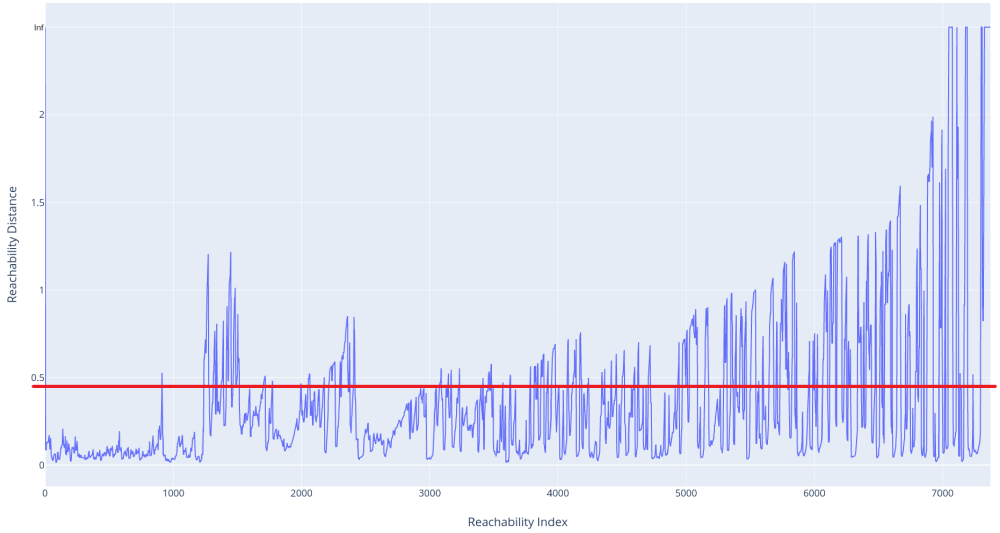

Figure 4.10: Reachability graph for the data set with line at $\epsilon \approx 0.45$.

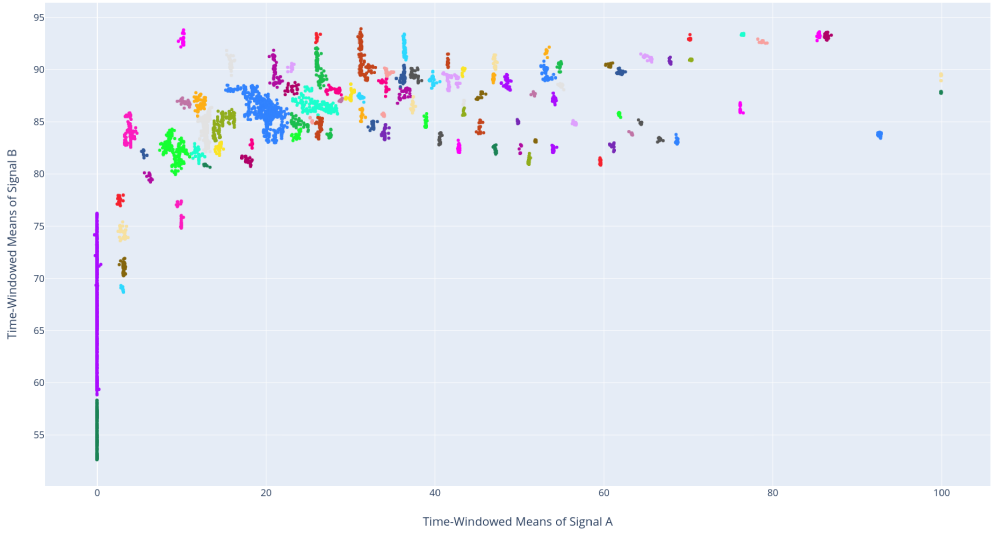

Figure 4.11: DBSCAN clustering result for $N = 10, \epsilon = 0.45$ without noise.

5. Summary

The majority of the thesis' content was aimed at explaining the algorithms which can be used to cluster data. That is because in order to find recurring patterns in the data, similar data samples must be found, actually, similar samples are patterns.

K-Means algorithm is the simplest and easiest to understand but it requires the knowledge of how many clusters are in the data which is not really suitable for finding patterns in the measurements which can be very large and N-dimensional so they cannot be plotted on the graph and clusters cannot be identifiable from looking at the graph.

Also, contrary to OPTICS and DBSCAN, if K-Means is used for the measurement data sets it will identify all points as some pattern which certainly cannot be true. Some points in data sets will be noise for sure.

That leaves us with determining which is better, OPTICS or DBSCAN. DBSCAN is really great if the density of a data set is consistent, but as it can be seen from sample data set presented here, points further on the x-axis are less dense therefore if we want to group those, OPTICS must be used instead of DBSCAN.

LITERATURE

[1] Dbscan. URL https://en.wikipedia.org/wiki/DBSCAN. [Online; accessed 11-June-2020].

[2] Degrees of freedom: What are they? URL https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/degrees-of-freedom/. [Online; accessed 11-June-2020].

[3] F-test. URL https://en.wikipedia.org/wiki/F-test. [Online; accessed 11-June-2020].

[4] Determining the number of clusters in a data set. URL https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set. [Online; accessed 11-June-2020].

[5] Optics algorithm. URL https://en.wikipedia.org/wiki/OPTICS_algorithm. [Online; accessed 11-June-2020].

[6] Clustering. URL https://scikit-learn.org/stable/modules/clustering.html. [Online; accessed 11-June-2020].

[7] Imad Dabbura. K-means clustering: Algorithm, applications, evaluation methods, and drawbacks, 2018. URL https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. [Online; accessed 11-June-2020].

[8] Mohammed Elbatta i Wesam Ashour. A dynamic method for discovering density varied clusters. *International Journal of Signal Processing, Image Processing and Pattern Recognition*, 6:123–134, 02 2013.

[9] Cory Maklin. Dbscan python example: The optimal value for epsilon (eps), 2019. URL https://towardsdatascience.com/machine-learning-clustering-dbscan-determine-the-optimal-value-for-epsilon-eps-python-example-3100091cfbc. [Online; accessed 11-June-2020].

[10] Victor Roman. Unsupervised machine learning: Clustering analysis, 2019. URL https://towardsdatascience.com/unsupervised-machine-learning-clustering-analysis-d40f2b34ae7e. [Online; accessed 11-June-2020].

Identifikacija ponavljajućih uzoraka kod mjerenja u testnim platformama

Sažetak

Mnogo informacija se može naći u uzorcima velikih setova podataka. Ovaj završni rad opisuje neke često korištene algoritme grupiranja podataka kao što su K-Means, DBSCAN i OPTICS koji se mogu koristiti za pronalazak grupa ili uzoraka. Nakon što su metode opisane, navesti će se rezultati nad testnim podatkovnim setom te će se odrediti koji je algoritam najbolji za grupiranje već navedenih testnih podataka.

Ključne riječi: grupiranje; podaci; znanost o podacima; uzorak; K-Means; OPTICS; DBSCAN

Identification of Recurring Patterns for Test Bed Measurements

Abstract

A lot of information can be found by looking at the patterns in big data sets. This thesis describes some common clustering techniques like K-Means, DBSCAN and OPTICS which can be used to identify clusters or patterns. After the methods are described, the results on a sample data set will be shown and the conclusion will be drawn which method is the best for the data at hand.

Keywords: clustering; data; data science; pattern; K-Means; OPTICS; DBSCAN