UNIVERSITY OF TURKU

# Cluster analysis for outlier detection

A case study of applying unsupervised machine learning on diesel engine data


Data Analytics

Master's Degree Programme in Computer Science

Department of Computing, Faculty of Technology


Master of Science Thesis


Author:

Otto Westerlund

Supervisors:

Prof. Jukka Heikkonen (University of Turku)

CEO Kim Westerlund (Edupower Oy Ab)


January 2023

**Master of Science Thesis**
**Department of Computing, Faculty of Technology**
**University of Turku**

**Subject**: Data Analytics
**Programme:** Master's Degree Programme in Computer Science
**Author**: Otto Westerlund
**Title**: Cluster analysis for outlier detection; A case study of applying unsupervised machine learning on diesel engine data
**Number of pages**: 53 pages, 1 appendix page
**Date**: January 2023

With the advent of modern data driven methods, engine manufacturers and maintainers are attempting to pivot from corrective to predictive maintenance. One way to achieve this goal is to install sensors on the engine and look for anomalies in the data patterns it produces. Companies such as Wärtsilä that provide condition monitoring services use the Fast Fourier Transform to manually look for anomalies in the data.

The Edge-project is an industrial research project involving institutions such as universities and private companies, with the goal of developing technical solutions and edge analytics for autonomous devices and vessels. Several papers and theses have been written as a result of the project, using techniques such as autoencoders to perform anomaly detection on data produced by sensors on a diesel engine.

This thesis explores the use of cluster analysis for anomaly detection on diesel engine data from the Edge-project. Finding clusters could potentially represent different states of the running engine, with anomalies being represented e.g. by data points far away from cluster centroids, or data points not belonging to any particular cluster. The techniques of K-means, DBSCAN and spectral clustering are used for assigning clusters, with silhouette coefficient and eigengap used as hyperparameter tuning heuristics. Distance from cluster centroids and reduced kernel density estimation are used to flag anomalies. T-SNE and Self-Organizing Maps are used as dimensionality reduction techniques to visualize the data into a 3-dimensional and 2-dimensional space, respectively.

Results show that what data are flagged as anomalies is highly sensitive to the choice of algorithm and chosen hyperparameters. The different results suggest different data as anomaly candidates. Therefore, further evaluation is needed from subject matter experts to determine which one of the models provides the most interesting results. Further work could include building an ensemble model that combines the used approaches, which could flag certain areas of the data space as a high risk for being anomalous.

**Keywords**: Cluster analysis, algorithm, engine, unsupervised learning, anomaly detection, edge computing

**Pro gradu -tutkielma**
**Tietotekniikan laitos, Teknillinen tiedekunta**
**Turun yliopisto**

**Pääaine**: Data-analytiikka
**Tutkinto-ohjelma:** Tietojenkäsittelytieteen maisteriohjelma
**Kirjoittaja**: Otto Westerlund
**Otsikko**: Cluster analysis for outlier detection; A case study of applying unsupervised machine learning on diesel engine data
**Sivut**: 53 sivua, 1 liitesivu
**Päivämäärä**: Tammikuu 2023


Moottorien valmistajat ja ylläpitäjät pyrkivät siirtymään korjaavasta huollosta ennakoivaan huoltoon modernien datavetoisten menetelmien avulla. Tämä voidaan saavuttaa esimerkiksi asentamalla antureita moottoriin ja etsimällä poikkeavuuksia anturien tuottamasta datasta. Yritykset kuten Wärtsilä, jotka tarjoavat kunnonvalvontapalveluita etsivät datasta poikkeavuuksia manuaalisesti Fourier-muunnosten avulla.

Edge-projekti on teollinen tutkimushanke, johon osallistuu mm. yliopistoja ja yksityisen sektorin yrityksiä, ja jonka tavoitteena on tuottaa teknisiä ratkaisuja ja reunalaskenta-analytiikkaa itseohjautuville laitteille, ajoneuvoille ja aluksille. Hankkeesta on kirjoitettu monia tutkimusartikkeleita ja opinnäytetöitä, joissa käytetään tekniikoita kuten syviä neuroverkkoja poikkeavuuksien havaitsemiseen dieselmoottoriin asennettujen anturien tuottamasta datasta.

Tämä opinnäytetyö tutkii klusterianalyysiä menetelmänä poikkeavuuksien havaitsemiseen Edge-projektissa ajetun dieselmoottorin datasta. Klusterit voisivat mahdollisesti edustaa ajettavan moottorin eri tiloja, ja poikkeavuudet voisivat olla esim. kaukana klusterien keskipisteistä olevia datapisteitä, tai datapisteitä, jotka eivät kuulu mihinkään tiettyyn klusteriin. Työssä käytetään algoritmeja K-means, DBSCAN ja spektraaliklusterointia klusterien määrittämiseen, ja siluettikerrointa sekä ominaisväliä käytetään hyperparametrioptimoinnin heuristiikkoina. Poikkeavuuksien merkintään käytetään etäisyyttä klusterien keskipisteisiin sekä alennettua ydintiheysestimaattoria. T-SNE:tä ja itseorganisoituvaa karttaa käytetään datan ulottuvuuksien vähentämisen tekniikoina, jotta data voidaan visualisoida 3- ja 2-ulotteiseen avaruuteen.

Tulokset osoittavat, että mikä data tulkitaan poikkeavana, riippuu vahvasti algoritmin ja sen hyperparametrien valinnasta. Menetelmien merkitsemät poikkeavuudet eroavat huomattavasti toisistaan. Tämän vuoksi vaaditaan aihealueen ammattilaisilta lisätutkimuksia, jotta voidaan päättää mikä malli luo mielenkiintoisimmat tulokset. Jatkokehitysideana voisi olla mallikokoelma, jossa yhdistyy tässä työssä käytetyt menetelmät, ja jonka tehtävänä olisi kartoittaa data-avaruuden eri alueiden riskit poikkeavuuksien sisältämiseen.



**Avainsanoja**: Klusterianalyysi, algoritmit, moottori, ohjaamaton oppiminen, poikkeavuuksien havaitseminen, reunalaskenta

# Table of contents

# 1 Introduction

From the 18th to the late 20th century, humanity has undergone three industrial revolutions, starting with steam power and early machines, continuing with electricity and the assembly line, and finally with programmable computers and partial automation. Despite all the progress these revolutions have brought, industry today still faces many challenges. One of the remaining challenges is the breakdown of engines and machinery. If a machine breaks, production or service providing may stand still until the machine is repaired, causing the service provider to lose money, and causing inconvenience to customers. To mitigate this, maintainers have moved from corrective maintenance to preventive maintenance, where maintenance is given at specific times to the machine before it stops working. However, preventive maintenance is still not fully optimal, as maintenance may be given long before it is needed. With the advent of the fourth industrial revolution, industry is attempting to move from preventive maintenance to predictive maintenance, where machinery is maintained only when really needed, before it malfunctions.

Machine learning is a popular and often effective tool for prediction and data analysis tasks. The field can broadly be separated into the categories of supervised learning, unsupervised learning, reinforcement learning, and combinations of these. One type of data analysis task that machine learning can perform is *anomaly detection*, which in turn can be a step towards achieving the goal of predictive maintenance. If available data contains pre-labelled examples of both normal and anomalous data, supervised learning methods can be used to train an outlier detection model. However, in many cases the "ground truth" of which parts of the data consist of anomalies and which parts are normal is unknown. In such cases unsupervised learning is a more appropriate approach for the task.

Mathematics and statistics provide classical methods for outlier detection and analysis. One popular method is Fourier analysis, which converts a signal from the time domain into the frequency domain, possibly exposing high-frequency peaks in the signal. While powerful, this method still requires a lot of manual analysis from human experts. Some methods may be precise in the mathematical sense but can often make simplistic assumptions about the underlying data models, and often also suffer from computational inefficiency. Methods from computer science and data mining, such as unsupervised machine learning, can be used with

the goal of preserving computational efficiency while obtaining a reasonable mathematical accuracy.

In this thesis unsupervised learning is used as a method for anomaly detection on diesel engine data. The work falls under the broader category of machine condition monitoring and fault diagnostics. The used data is from an industrial research project called the Edge-project. Because pre-existing information about anomalies in the data set is not available, a reasonable approach is to model the normal behaviour of the engine based on the available data. Then, deviations from the normal behaviour may indicate the presence of anomalies. This is the approach taken in this thesis. As for the model of the normal behaviour, unsupervised clustering methods are chosen. The hypothesis is that the engine will exhibit certain groups of behaviours, corresponding to different states, which may translate to clusters in the data set. The central research question is how robust is cluster analysis regarding the anomalies they find – do different clustering algorithms flag the same data as outliers? K-means, DBSCAN and spectral clustering algorithms are compared, as they represent different paradigms of clustering methods. The goal is to provide insights and complement other analytical work done on the data, serving as building blocks towards a more automated anomaly detection framework.

The rest of the thesis is organized in the following way. Chapter 2 provides the necessary theoretical background of the chosen methods and the Edge-project. Chapter 3 explains the methods of the conducted research and experiments. Chapter 4 presents the experimental results. Chapter 5 provides conclusions, limitations, and a discussion.

# 2 Background

## 2.1 Edge-project

This work is related to the Edge-project that is an industrial research project with the goal of developing technical solutions and edge analytics for autonomous devices and vessels. The project has involved several institutions, such as universities and private companies.
So far, there have been previous works done as a result of the Edge-project. In (Mtesigwa, 2020), Mtesigwa describes the setup and configuration of an edge computing framework for gathering engine data from a marine vessel. Kekäläinen describes in (Kekäläinen, 2021) a data processing and analysis pipeline and the challenges that the data quality has posed. Siganos (Siganos, 2019) presents a framework based on neural network autoencoders to detect anomalies from engine vibration data in an unsupervised manner. In their work they mention that companies such as Wärtsilä currently provide a condition monitoring service. This service converts signals from the time domain to the frequency domain with the Fast Fourier Transform, which analysts then manually inspect to find anomalies. Siganos' framework aims to provide additional automation of this process through unsupervised deep learning.

This thesis explores the possibility of using cluster analysis as an alternative for anomaly detection of engine sensor data from the Edge-project. Deep learning methods are highly successful in many domains; however, they are usually considered "black boxes" as the reason for their results are difficult to explain. Clustering methods are often simpler and have more intuitive inner workings, making it worth it to explore them as a possible solution for the anomaly detection problem. The intent is not to replace, but rather complement the proposed neural network methods.

## 2.2 Anomaly detection

Anomaly detection refers to finding abnormalities in how data is expected to behave. This is useful in many domains, such as fraud detection, cyber security, and fault detection in mechanical units such as engines. When the normal behaviour of a system is known, unexpected patterns may indicate that the system is about to fail or more generally that something unwanted is happening.

Chandola et.al. (Chandola;Banerjee;& Kumar, 2007) describe different types of anomalies. The first category, *point anomalies*, refers to data points that are outliers compared to the rest of the data set, i.e., visually they may lie far away from the rest of the data points, as shown in Figure 1.

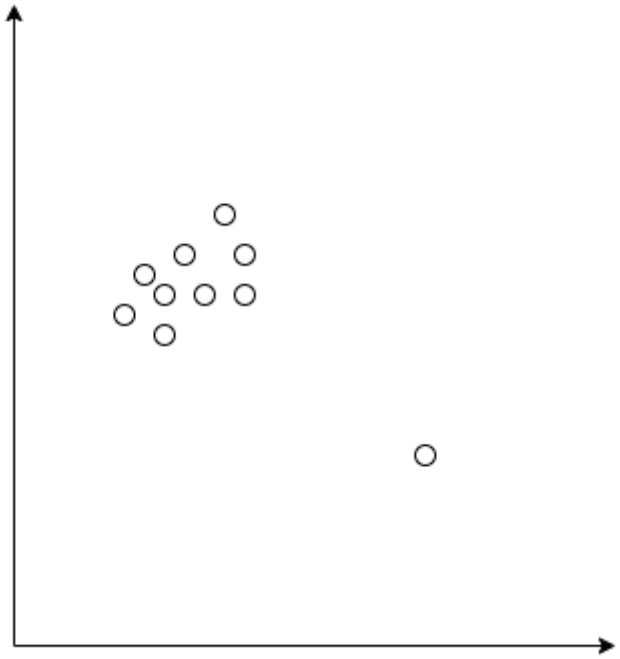

*Figure 1. The data point most to the right is far away from the rest of the points and can as such be considered a point anomaly.*

*Contextual anomalies* refer to data points that are within normal ranges when considering the rest of the data set but are abnormal given their context. As an example, the temperature of negative 20°C is not a point anomaly for the temperatures in Nordic countries, however it is a contextual anomaly during an afternoon in the middle of July. Contextual anomalies often have a temporal aspect to them, i.e., they usually appear in time-series data.

The final category is *collective anomalies*. This refers to data points that together as a collection are anomalous when compared to the rest of the data.

It should be noted that some authors make a distinction between outliers, noise, and anomalies. In (Aggarwal, 2017), Aggarwal defines noise and anomalies as being types of, or rather degrees of outliers. Specifically, anomalies are outliers that are of particular interest to an analyst, whereas noise are outliers of a lesser degree or with no interesting properties. In

this thesis, the term outlier and anomaly are used interchangeably; whether flagged outliers are of interest comes down to domain knowledge.

Furthermore, outliers can be identified in different manners, either with **a)** a binary label, identifying each data point as either normal or anomalous (hard categorization) or **b)** a score that indicates how much a data point deviates from the expected normal pattern (soft categorization). While type **b** contains more information, type **a** is often necessary for decision making processes. The types need not be mutually exclusive, rather a type **b** outlier can after a certain threshold be converted into a type **a** "hard" outlier. Setting that threshold also comes down to subject matter expertise.

## 2.3 Supervised vs unsupervised machine learning

Machine learning is the science of making computers do things without each instruction being explicitly programmed. This can be achieved when there is a suitable amount of *training data* available. A data set is fed to a *learning algorithm*, which analyses the structure of the data and tries to learn an unknown function f(X) = Y, also known in the literature as a *hypothesis*, such that exemplars from the input data set X are mapped to a desired output Y. This is an instance of predictive modelling, which means that the goal is not to learn a hypothesis f(X) = Y that only maps instances of the input data set to their corresponding outputs of Y, but rather the goal is with the help of the data set X learn a hypothesis such that it can generalize and map *unseen data* to the output set Y with a reasonable accuracy. Since what constitutes as a "reasonable" accuracy (or other desired quality metric of the hypothesis f), what is known or assumed about the data generating process, and the characteristics of the inputs and outputs are all factors that can vary drastically, many different approaches have been suggested to tackle the predictive modelling problem.

Table 1 shows a toy data set containing the features height, weight, and species of different animals. As each row of the data set contains three description attributes, or features, it is said that the data set has three *dimensions*.

In data mining, a typical task would be to predict the species of an animal given the height and weight as input. This is called classification and is an example of *supervised learning*. The procedure would typically be to feed in rows 1 through 6 of the table to a learning algorithm, with the Species column specified as the *target variable*. The algorithm would then

be able to infer that the 7th row where the target variable information is missing most likely belongs to the Human class. To formalize the approach, supervised learning is a method of finding a hypothesis f(X) = Y by having a selection of (X, Y) pairs as input for the learning algorithm. In addition to classification, supervised learning is often used for regression tasks.

| Height (cm) | Weight (kg) | Species |
|---|---|---|
| 0,3 | 0,003 | Ant |
| 173 | 74 | Human |
| 370 | 4018 | Elephant |
| 329 | 3692 | Elephant |
| 190 | 93 | Human |
| 0,22 | 0,002 | Ant |
| 185 | 87 | |

*Table 1. Sample data set of animal species. A typical supervised learning task would be to predict the missing species label, classifying the data point.*

Table 2 shows the same data set, but this time there is no species information. Another typical data mining task would be to find out whether the data points can be assigned to different groups in some logical way. This is task is called cluster analysis and is an example of *unsupervised learning*. A good algorithm would identify three distinct groups, corresponding to the different species previously present in the data set. To again formalize the approach, unsupervised learning attempts again to find a suitable hypothesis f(X) = Y, however only exemplars of X are available. In addition to cluster analysis typical tasks of unsupervised learning include association analysis and most dimensionality reduction tasks.

| Height (cm) | Weight (kg) |
|---|---|
| 0,3 | 0,003 |
| 173 | 74 |
| 370 | 4018 |
| 329 | 3692 |
| 190 | 93 |
| 0,22 | 0,002 |
| 185 | 87 |

*Table 2. Data set with unknown class labels and no target variable. A typical unsupervised learning task would be to find out if the data points can be grouped in some optimal way.*

## 2.4 Cluster analysis

Cluster analysis is an unsupervised machine learning technique, where data is split into groups based on some similarity metric. The goal is to have similar data within the same group, and dissimilar data in different groups (clusters). Contrary to supervised learning,

where training data contains pre-determined labels which act as the target variable for a prediction, in unsupervised learning the data is unlabelled and with no specific target variable. Rather, the goal of the learning algorithm is to find a suitable labelling for the data points. Figure 2 illustrates the different starting points in supervised and unsupervised machine learning, with labelled data for supervised classification in Figure 2a and unlabelled data for an unsupervised clustering task in Figure 2b.

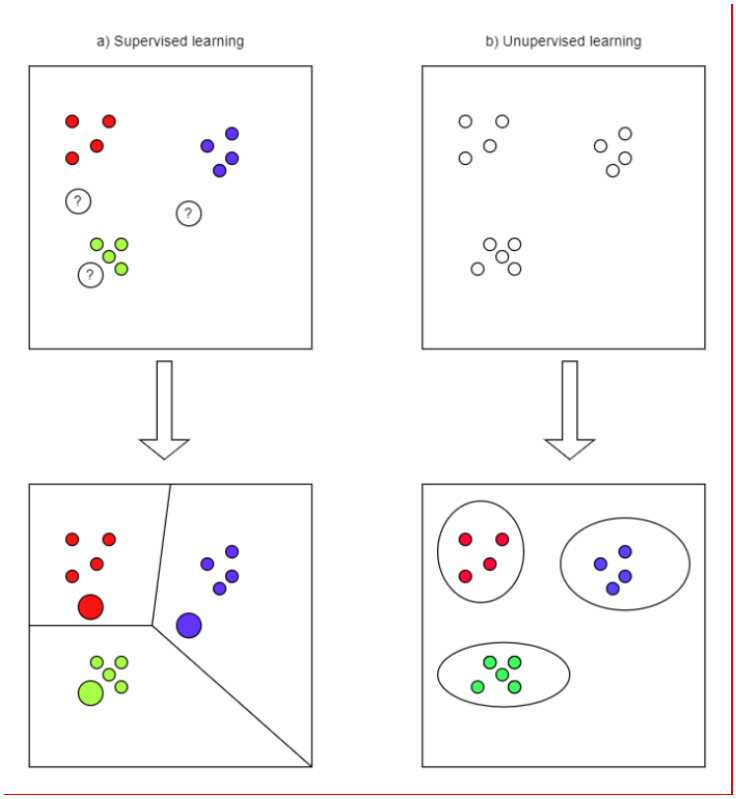


*Figure 2.* ***a)*** *The data points have labels to begin with, indicated by their colour, and the algorithm learns to classify unseen data according to these labels.* ***b)*** *All data points are unlabelled in the start, and the algorithm assigns labels to them based on some notion of similarity.*

Determining the criterion by which data are considered similar or dissimilar is no trivial task. Consider the dotted lines in Figure 3a. The dots could be considered data points on a 2D plane. A sample task could be to assign the dots into groups in some logical way. A first intuitive idea is to divide the points into two groups as in figure 3b – each line forming its own group or cluster. However, a different algorithm might consider data points that are as far away from each other as possible as most dissimilar, resulting in a clustering like in Figure 3c.

Often in unsupervised machine learning tasks the “ground truth” is unknown, and assessing whether scenario 3b or 3c makes more sense comes down to domain expertise.

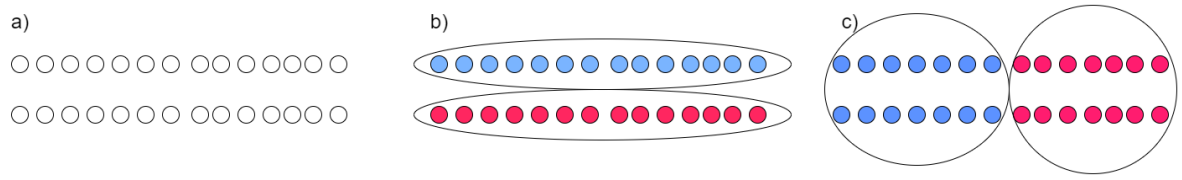


*Figure 3.* ***a)*** *The dots can be considered data points on a two-dimensional plane.* ***b)*** *a sample cluster partitioning obtained by an algorithm.* ***c)*** *the same data clustered in a different way by some other algorithm.*

Cluster analysis can be used as a tool for anomaly detection. Chandola et.al. (Chandola;Banerjee;& Kumar, 2007) describe three possible assumptions that are made when cluster analysis is applied for anomaly detection. The assumptions are:

1. Normal data instances belong to a cluster, while anomalies do not belong to any cluster.
2. Normal data instances are close to the closest cluster centroid, while anomalies are far away from the closest cluster centroid.
3. Normal data instances belong to large and dense clusters, while anomalies belong to small or sparse clusters.

Which assumption is made depends on the clustering algorithm of choice. From the algorithms explored in this thesis, K-means and Self-Organizing Maps operate under the second assumption, DBSCAN operates under the first assumption, and spectral clustering can operate under the third assumption or second assumption.

Clustering algorithms can be grouped into different categories depending on their mode of operation. Popular types of clustering algorithms include:

- Centroid-based clustering
- Connectivity-based clustering
- Density-based clustering
- Neural networks

In centroid-based clustering, data objects are grouped based on their distances to the centres of the current clusters. Typical distance metrics include squared Euclidean distance. Centroid-based clustering algorithms include the popular K-means algorithm.

Connectivity-based clustering is also known as *hierarchical clustering*. It is an umbrella term for methods that rather than placing each element in a single cluster as a hard categorization, they create a hierarchy of clusters based on a similarity score and linkage criterion. An example algorithm of this family is agglomerative clustering.

In density-based clustering, clusters are formed based on how densely the data points lie in the data space. Areas of high density form clusters, whereas sparse areas form borders between clusters or are considered noise. A popular density-based clustering algorithm is DBSCAN.

From unsupervised neural networks, the Self-Organizing Map (SOM) is among the most well-known ones.

## 2.5 K-means

K-means is one of the most well-known and simplest clustering algorithms. The algorithm works by choosing *k* data points as centroids based on some initialization criteria (most simple is to choose the initial centroids randomly) and calculating the Euclidean distances from each other data point to each of the centroids. The data points are assigned to a cluster belonging to the closest centroid. Next, the centroids of the clusters are updated by calculating the mean coordinates of each cluster. After this, a new iteration of assigning data points to centroids and recalculating centroids is started, until convergence or a stopping criterion is met. Formally: Let $X = \{x_1, x_2, \dots x_n\}$ be a set of *n* data points d-dimensional data points, i.e. $x_i = \{a_1, a_2, \dots a_d\} \in R^d$. Let $M = \{m_1, m_2, \dots m_k\}$ be a collection of *k* cluster centers (centroids). Then, the K-means objective function is defined as (Likas, Vlassis, & Verbeek, 2003):

$$E(M) = \sum_{i=1}^{n} \sum_{c=1}^{k} B(x_i \in C_c) ||x_i - m_c||^2$$

Here, B is a binary variable with the value 1 if $x_i$ belongs to the c-th cluster denoted $C_c$, and 0 otherwise. In other words, the objective is to find such a collection of cluster centroids that the sum in the equation (the error measure) is minimized.

| **ALGORITHM 1: K-MEANS** | |
|---|---|
| Variables:<br>• k = the number of clusters to search for<br>• s = current iteration counter<br>• l = iteration limit | |
| **1** | **begin** |
| **2** | Choose k data points as centroids based on initialization criterion (e.g. randomly) |
| **3** | For each data point x: |
| **4** | Calculate the Euclidean distance from x to each centroid |
| **5** | Assign x to the cluster belonging to the closest centroid |
| **6** | For each cluster: |
| **7** | Assign a new centroid = the mean coordinates of all data points in the cluster |
| **8** | Increase s |
| **9** | If cluster assignments changed and s < l, repeat from step 3 |
| **10** | **end** |

K-means has been widely used in many different domains and applications, ranging from biology (Botia, et al., 2017) to fraud detection (Anamika;Mayuri;Reddy;Iyengar;& Caytiles, 2018). However, the algorithm has some drawbacks. Firstly, the final result is sensitive to the initial selection of cluster centroids. Secondly, the algorithm can only find clusters with a regular spherical shape. And thirdly, one must specify the parameter *k*, the number of clusters to search for, a priori. In real applications, the third drawback is significant, as the number of clusters to search for is often unknown. There is research around mitigating this drawback however (Shao;Lee;Liu;& Shen, 2017). Sometimes the K-means algorithm is executed several times with different values for k, and then the resulting cluster partitioning is evaluated with metrics such as Calinski-Harabasz index or silhouette coefficient.

## 2.6 Self-Organizing Maps

The Self-Organizing Map (SOM) (Kohonen, 1990), also known as the Kohonen map or Kohonen network, is a type of artificial neural network used for data visualization and dimensionality reduction in an unsupervised manner. For a k-dimensional data set with n observations, the SOM produces a two-dimensional visualization, a sort of heat map, for each

of the k features. From these maps the user can recognize the possible existence of clusters within the values of a variable, and how the different variables correlate with each other: if two features have a similar looking heat map, the features are highly correlated.
The algorithm attempts to preserve the topology of the data – that is, points that are near each other in the low-dimensional projection are likely as such also in the original data space. The opposite, however, is not necessarily true, meaning that points that are far from each other in the low-dimensional projection are not necessarily as such in the original data space.

In the visualization produced by the SOM, the map usually consists of a two-dimensional grid of hexagons. The number of hexagons (which correspond to the neurons in the neural network) is a hyperparameter that needs to be determined before running the algorithm. Each of these neurons has a weight vector, which represents the position of the neuron in the input space. The algorithm attempts to "move" the weight vectors of the map space towards the data vectors of the input space by minimizing a distance metric such as the Euclidean distance.

**ALGORITHM 2: SELF-ORGANIZING MAP**

Variables:

- s = current iteration step
- λ = the iteration limit
- t = the index of the target input data vector of the input data set D
- D(t) = target input data vector
- v = the index of the node in the map
- $W_v$ = the current weight of node with index *v* in the map
- u = the index of the best matching unit (BMU) in the map
- α(s) = monotonically decreasing sequence of scalar coefficients, also known as neighborhood function

**begin**
Randomize the node vectors in a map
Randomly pick an input vector D(t)
For each node *v* in the map:
Calculate Euclidean distance between input vector D(t) and weight vector $W_v$
Assign u = index of the node with the smallest Euclidean distance = BMU
Update the weight vectors of the nodes in the neighborhood of the BMU (including the BMU itself) by pulling them closer to the input vector with the formula:

$$\boldsymbol{W_v(s+1) = W_v(s) + \alpha(s) \cdot (D(t) - W_v(s))}$$

Increase s and repeat from step 3 while s < λ
**end**

(Kohonen, 1990)

Another thing that can be extracted from the SOM is a Unified Distance Matrix (U-MAT). The U-MAT gives a higher-level view of the entire data set and how the data vectors are grouped in relation to each other. As the SOM "moves" around the weight vectors towards the input data vectors, the U-MAT is given by the Euclidean distances between the weight vectors in the resulting map. Usually this is depicted in an image of hexagons (representing the neurons in the SOM), where the distances between the vectors are represented by the colour of the neuron, e.g. a dark neuron can mean a low distance between vectors and a light neuron a higher distance between vectors. Concrete examples of SOM outputs can be seen in chapter 3.

## 2.7 DBSCAN

Density-Based Spatial Clustering of Applications with Noise, or DBSCAN, is another widely used clustering algorithm. It operates by identifying regions that are densely packed with data points, making these into clusters, and considering sparse regions as outliers.
When clustering data, the algorithm classifies data points into three categories:

- Core points
- Border points
- Noise points

Core points are data inside a densely packed region, forming a cluster. A data point is considered a core point if it is within a certain kind of neighbourhood around the point. This neighbourhood is given by the user-defined distance parameter ε (epsilon) and the also user-defined threshold parameter *minPts*, the minimum number of points within the distance ε of the current point for it to be considered a core point.

Border points are points that are in the region of a core point but are not core points themselves (they don't have ≥ minPts points within distance ε). A data point can be a border point for multiple core points.

Noise points are data points which are neither core points nor border points.

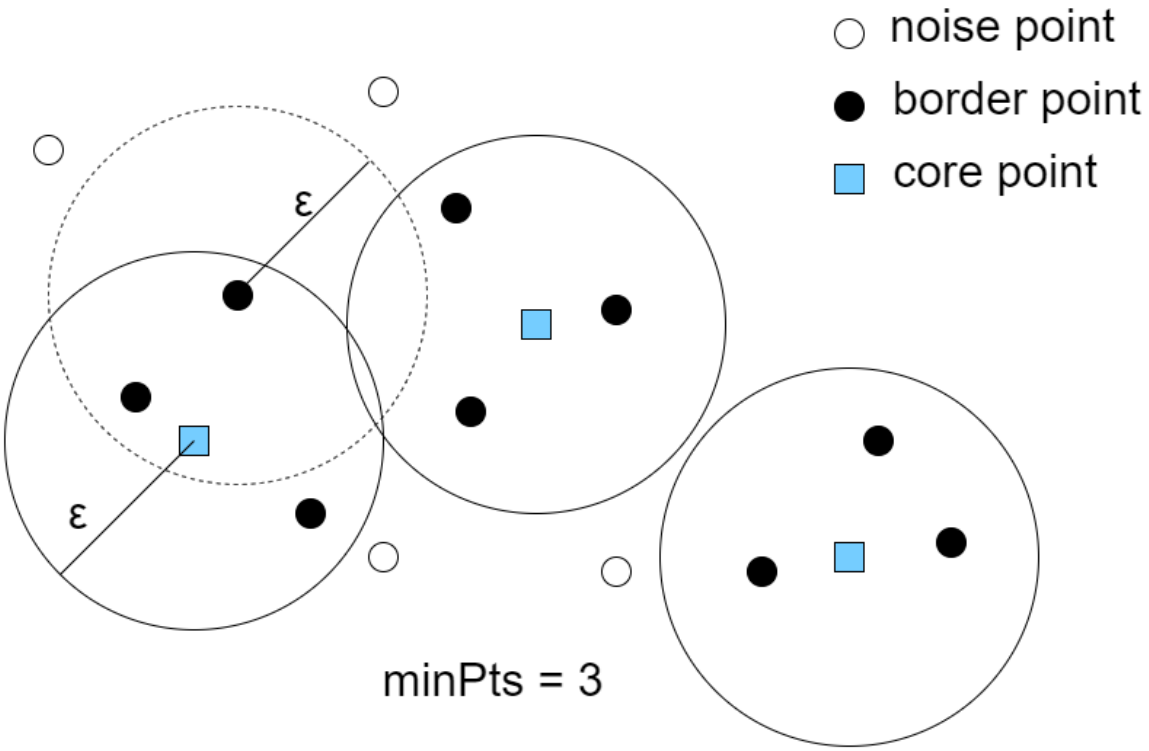


*Figure 4. Point assignment of DBSCAN algorithm. Because the point with the dotted perimeter has < minPts points within distance ε it is a border point instead of another core point.*

| **ALGORITHM 3: DBSCAN** | |
|---|---|
| Variables:<br>• minPts = the minimum number of points within distance ε of the current point for it to be considered a core point<br>• ε = the distance threshold for the core/border point calculation | |
| **1** | **begin** |
| **2** | Label all points as core, border, or noise points |
| **3** | Eliminate noise points |
| **4** | Put an edge between all core points that are within ε of each other |
| **5** | Make each group of connected core points into a separate cluster |
| **6** | Assign each border point to one of the clusters of its associated core points |
| **10** | **end** |

(Schubert;Sander;Ester;Kriegel;& Xu, 2017)

One key advantage of DBSCAN is that it can find clusters of arbitrary shapes. Also, while it has a worst-case time complexity of $O(n^2)$, the original DBSCAN paper

(Ester;Kriegel;Sander;& Xu, 1996) claims it often runs at O(n log n) time complexity in practice by certain data structure usage. However, (Schubert;Sander;Ester;Kriegel;& Xu, 2017) claims that the conclusion of an average time-complexity of O(n log n) is false.

While DBSCAN is powerful and effective in many situations, its usefulness is limited in data sets where there are few regions of varying density. In such cases, the algorithm may easily tend to assign all points into a single cluster or classify them all as noise.

## 2.8 Hierarchical clustering

Hierarchical clustering is different from partitioning algorithms such as K-means and DBSCAN in that while the previous methods assign each data point to exactly 1 cluster (K-means) or to 0 or 1 clusters (DBSCAN), hierarchical clustering allows clusters to consist of subclusters. This way, a tree-like hierarchy of clusters is formed, with the root node/cluster containing all data elements, and the leaf nodes being the data points themselves in their own 1 element clusters. This hierarchy can be visualized with the help of dendrograms, such as in Figure 5.

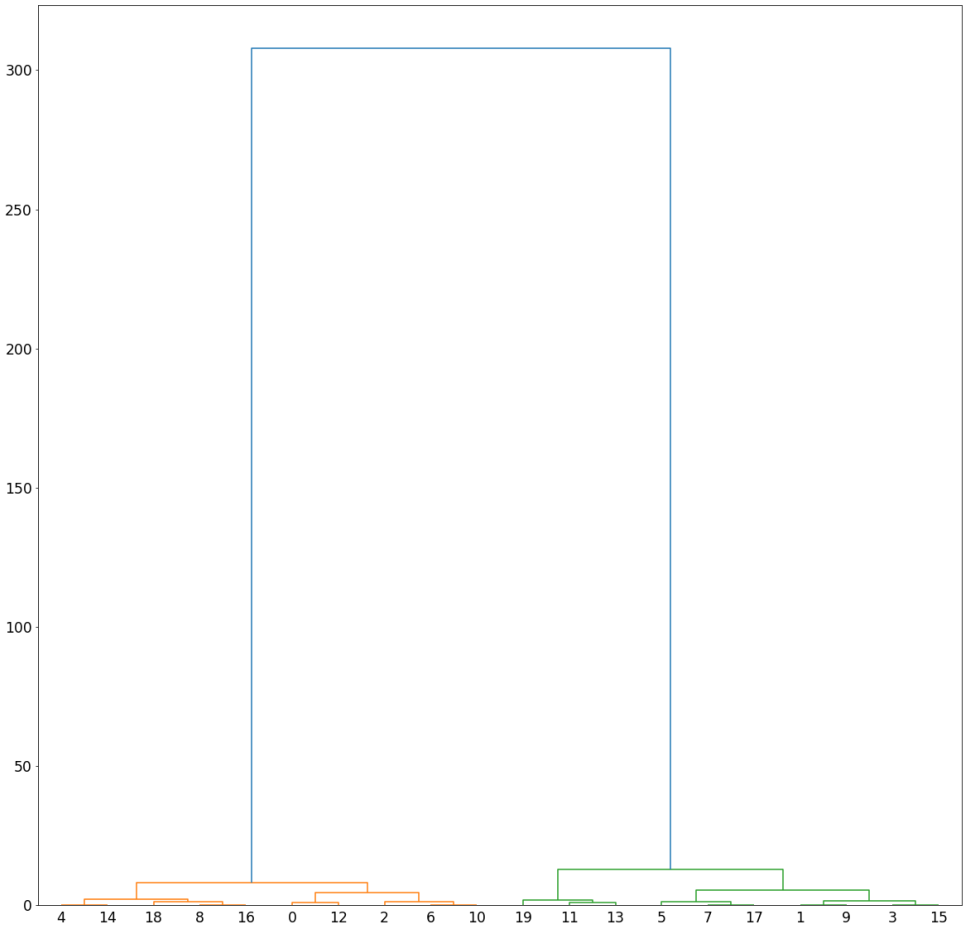


*Figure 5. Dendrogram illustrating Agglomerative clustering on a toy data set (shown in Figure 6). The dendrogram shows that there are two main clusters (the blue "legs"), which are relatively far away from each other (the height of the blue legs), and the samples within each cluster are relatively close to one another (the height of the lower levels).*

The dendrogram in Figure 5 represents the hierarchy of a toy data set consisting of 2 clusters, shown in Figure 6.

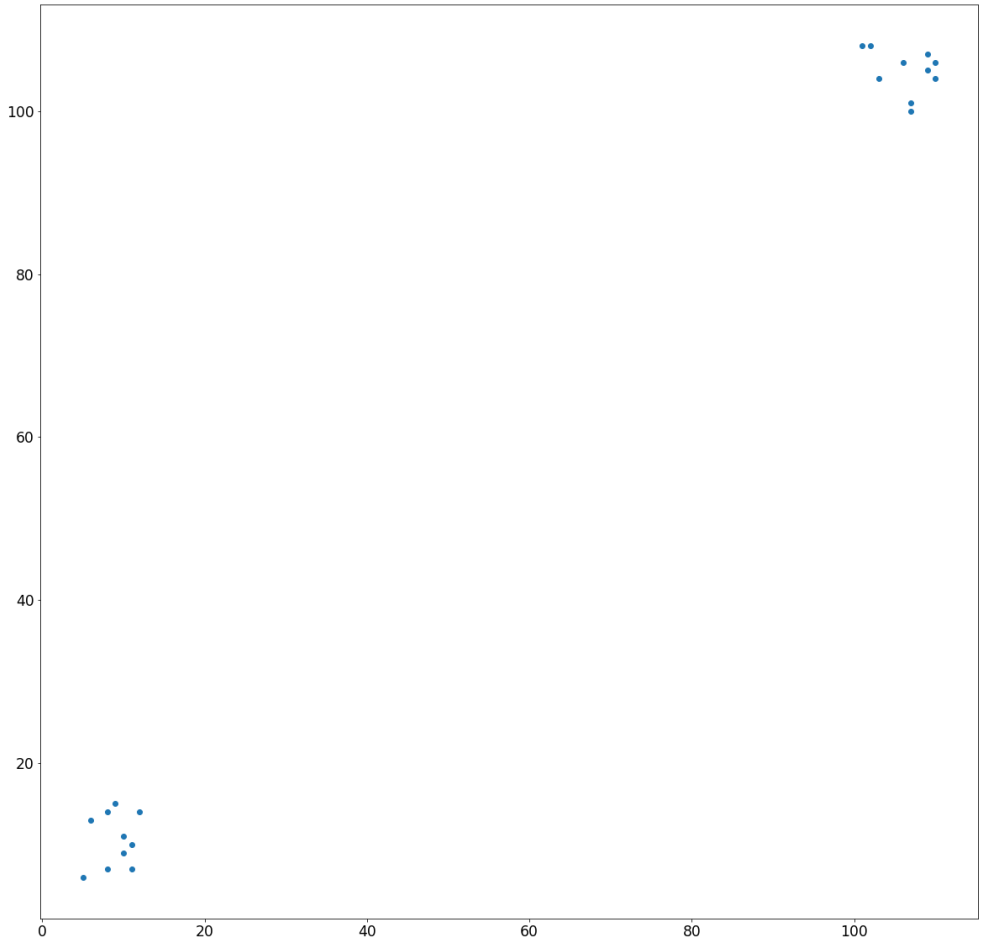


*Figure 6. Toy data set with 2 clusters. As shown in the dendrogram in Figure 5, the clusters are far away from each other, compared to the distances between the samples within each cluster.*

Commonly hierarchical clustering methods are split into two categories:

- Agglomerative hierarchical clustering
- Divisive hierarchical clustering

Agglomerative clustering works with a “top-down” approach, starting with the individual data points, and merging them into bigger and bigger clusters, until the single cluster with all items

is reached. Divisive methods work in reverse, starting with the single cluster, splitting it into smaller and smaller ones until the single-element clusters are reached.

As an example, agglomerative clustering partitions the toy data set in Figure 6 into clusters as shown in Figure 7.

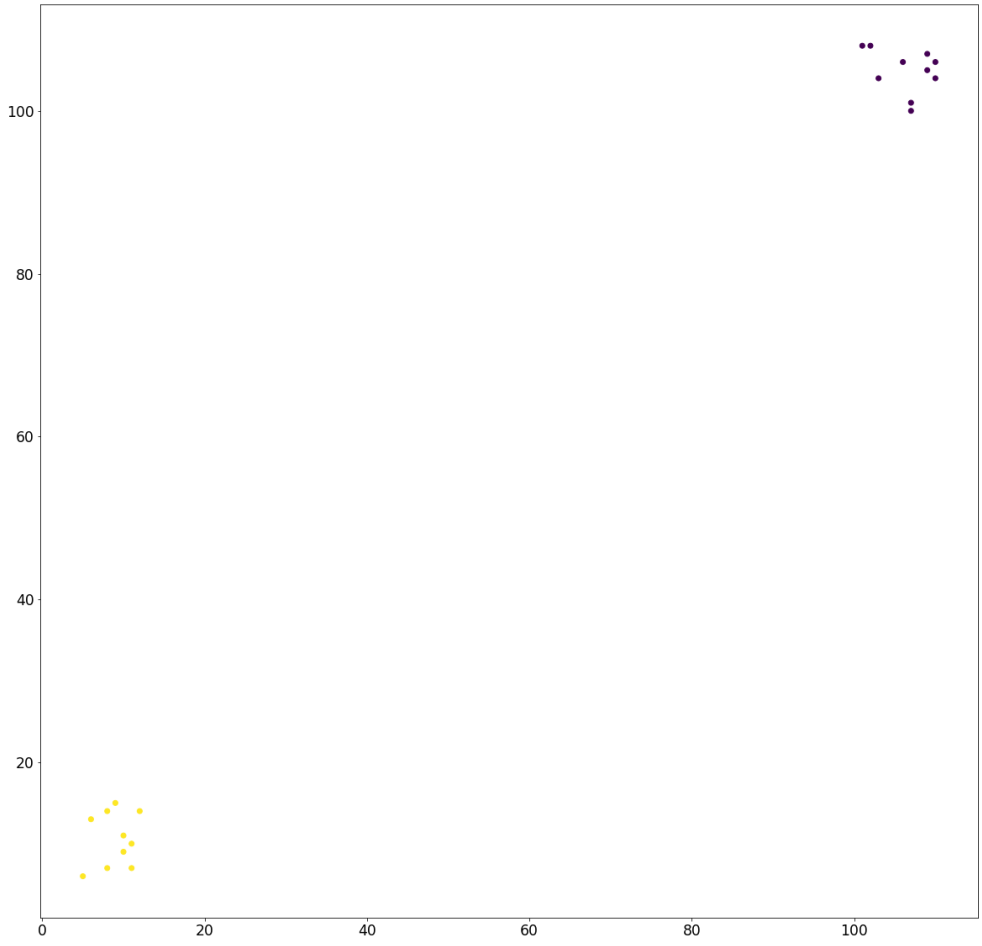


*Figure 7. Same toy data set with the data points being coloured according to which cluster it belongs to according to the agglomerative clustering algorithm.*

## 2.9 Spectral clustering

A technique that has recently received a lot of attention is spectral clustering. It works by performing a dimensionality reduction on the high-dimensional data through usage of the eigenvalues of a similarity matrix of the data, then performs clustering on the lower-dimensional representation of the data. Consider a matrix A, a non-zero vector x and a scalar λ. If these entities fulfil the property $Ax = \lambda x$, then x is an *eigenvector* of A, and λ is its *eigenvalue*.

Specifically, the matrix that the algorithm operates on is usually a graph Laplacian matrix. Let A be a symmetric similarity matrix, where $A_{ij} \geq 0$ is the similarity between data points i and j. Let D be the diagonal matrix of A, such that $D_{ii} = \sum_j A_{ij}$. Then, the graph Laplacian matrix L is defined as $L = D - A$.
The basic spectral clustering algorithm uses the following steps:

| **ALGORITHM 4: SPECTRAL CLUSTERING** | |
|---|---|
| Variables:<br>• k = the number of clusters to search for<br>• A = similarity matrix | |
| **1** | **begin** |
| **2** | Calculate the diagonal matrix D such that $D_{ii} = \sum A_{ij}$ |
| **3** | Calculate the Laplacian matrix L = D – A |
| **4** | Calculate the first k eigenvectors, corresponding to the k smallest eigenvalues of L |
| **5** | Form a matrix from the first k eigenvectors. The l-th row defines the features of graph node l |
| **6** | Cluster the graph nodes based on these features with some clustering algorithm such as K-means |
| **7** | **end** |

(Shalev-Shwartz & Ben-David, 2014)
Depending on the clustering algorithm used in step 6 of Algorithm 4, one may need to provide the number of clusters as an input parameter. Linear algebra methods provide a heuristic for estimating a good number of clusters. When inspecting the eigenvalues of the graph Laplacian matrix L, the first non-zero eigenvalue or the second smallest eigenvalue is called the Fiedler value. In a graph G, the number of connected components is same as the number of times the value 0 appears as an eigenvalue in the graph Laplacian matrix. This can be used as a heuristic to approximate the number of clusters for the spectral clustering algorithm. A similar heuristic is the “eigengap” heuristic. The idea is that when inspecting the

eigenvalues, if the k+1:th eigenvalue is significantly larger than the k:th eigenvalue, and the difference between the k first eigenvalues is relatively small, then k is probably a good number of clusters, as seen in Figure 8.

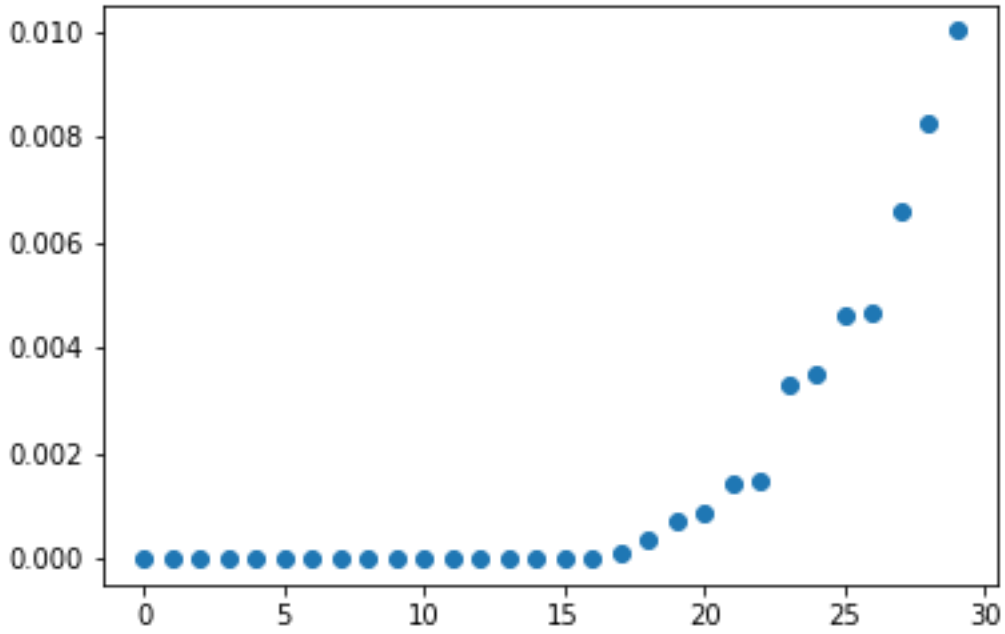


*Figure 8. Eigenvalues of a graph Laplacian. First non-zero eigenvalue at index 17, and a clear gap is at index 23. Thus, 16 and 22 are good estimates for the number of clusters to be provided as an input parameter for spectral clustering.*

## 2.10 t-SNE

t-distributed Stochastic Neighbourhood Embedding (t-SNE) is a dimensionality reduction method, most well suited for visualizing high-dimensional data with a two- or three-dimensional embedding. It works by minimizing the Kullback-Leibler divergence (a distance measure between probability distributions) between probability distributions modelling the high- and low-dimensional spaces. In the original data space, a Gaussian distribution is used, and in the low-dimensional embedding a Student's t-distribution is used.

Formally: given an input data set $X = \{x_1, x_2, \dots, x_n\} \subset R^d$, the joint probability or similarity $p_{ij}$ between two data points $x_i$ and $x_j$ is given in the original high-dimensional space by

$$p_{i|j} = \frac{\exp\left(-||x_i - x_j||^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} exp(-||x_i - x_k||^2 / 2\sigma_i^2)} \text{ , then } p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n}$$

The data in the low-dimensional embedding is denoted as $Y = \{y_1, y_2, \dots, y_n\} \subset R^s$, such that $s \ll r$ (typically s = 2 or s = 3). The similarity $q_{ij}$ between points $y_i$ and $y_j$ is given by

$$q_{ij} = \frac{(1 + \left|\left|y_i - y_j\right|\right|^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^{2^{-1}}}$$

The t-SNE cost function is defined as

$$C(Y) = KL(P||Q) = \sum_{i \neq j} p_{ij} log \frac{p_{ij}}{q_{ij}}$$

(Linderman & Steinerberger, 2019)

In simpler terms, the algorithm calculates pairwise similarities between each data point in the high-dimensional space. Then, it "drops" the data points randomly to a low-dimensional embedding. There, the algorithm moves around the points in a systematic manner, until the cost function above is minimized through gradient descent.

## 2.11 Silhouette coefficient

It can be a difficult task to assess the quality of a cluster partitioning obtained by a clustering algorithm. For this purpose, several metrics have been proposed, one of which is the silhouette coefficient.

The silhouette metric measures cluster cohesion and separation. In other words, it attempts to assess how similar a data point is to other data points within its own cluster, and how dissimilar it is to data points in other clusters. The similarity/dissimilarity score can be calculated with any metric of choice, such as Euclidean distance. The silhouette coefficient is calculated in the following way:

Let A be a cluster obtained by an algorithm like K-means. Let a(i) be the average dissimilarity of a data point *i* to all other points in cluster A. Let B be the cluster closest to point *i* that is not A, then b(i) = d(i, C) = average dissimilarity of point *i* to all points in cluster B. The silhouette s(i) of data point *i* is defined as (Rousseeuw, 1987):

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i),\ b(i)\}}$$

Commonly, an average silhouette score is calculated for the entire clustering obtained by the clustering algorithm to give an approximation of the quality of the entire cluster partitioning.

## 2.12 K-fold cross-validation

When applying machine learning algorithms on data, the outputs depend heavily on the values of so-called *hyperparameters* of the algorithm. The hyperparameters of the algorithm can be thought of as initiation settings that can be tuned to different values, depending on what kind of output is desired from the algorithm. An example is the value of k for the K-means algorithm, or epsilon and minPts for DBSCAN. Sometimes it is difficult to know in advance what the best hyperparameters are for a given task, but there are methods to estimate this. An example of this is a k-fold cross-validation.

K-fold cross-validation is a procedure where the data set is split in *k* distinct ways, where each of the k variants consists of the entire data set divided into a training and test set (and optionally a validation set), as shown in Figure 9. The goal of this is that by training and testing the algorithm and hyperparameter settings on several subset variants of the data, one can better estimate the impact of the chosen settings on unseen data.

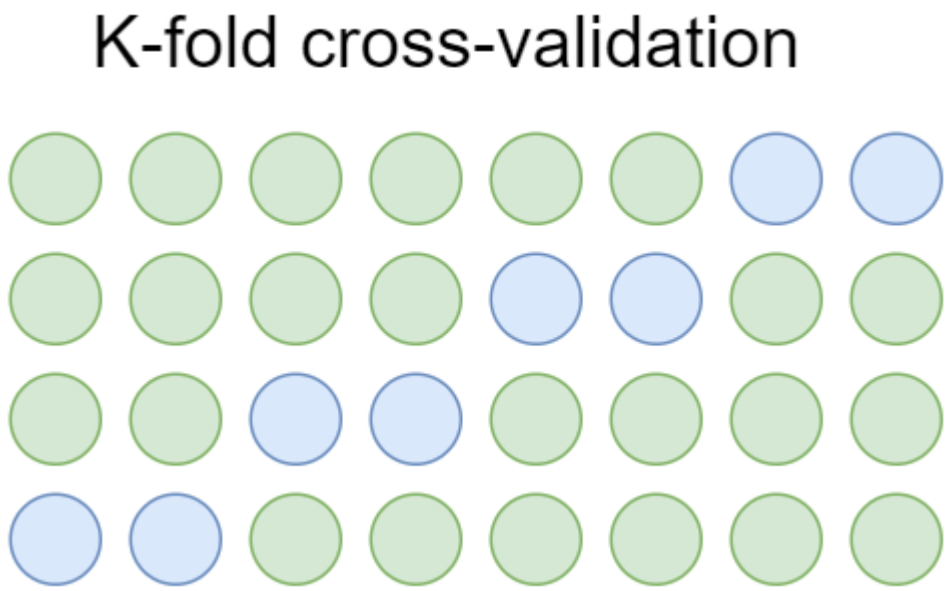


*Figure 9. Illustration of a data set split into a 4-fold cross-validation procedure. Each row is an instance of the entire data set, where the green circles represent training data and the blue circles test data.*

## 2.13 GridSearch cross-validation

GridSearch cross-validation is an extension of k-fold cross-validation. In a GridSearch cross-validation, each of the hyperparameters are given a range of possible values to test with.

Then, the algorithm of choice is run by exhaustively testing each combination of hyperparameter tuples against some scoring criterion; an example that will be used in this thesis is running an exhaustive search on epsilon-minPts pairs on the DBSCAN algorithm with silhouette coefficient as the scoring criterion. To give the best results, each hyperparameter tuple can be run through a k-fold cross-validation, meaning that the data set is split into a subset in *k* different ways, and the average score of these splits can be used to give a total assessment of the quality of the current iteration of hyperparameter combinations.

GridSearch cross-validation is computationally quite expensive but can give great results in a reasonable time if the data set is small enough. The time-complexity of a GridSearch cross-validation is O(mnk) for DBSCAN, where m is the range of epsilon values, n is the range of minPts values, and k is the number of splits in the k-fold cross-validation.

# 3 Method

This chapter lays out the experimental methods used in this thesis.

## 3.1 Data preliminaries

For the experiments conducted in this thesis, data from a diesel engine run in a laboratory setting was available. The engine is for a large marine vessel and was run in the laboratory detached from the vessel. There were two primary data sets: accelerometer data representing vibrations in spatial x-, y- and z- dimensions, and data from internal engine sensor variables, representing features like temperatures, pressures, humidity and more. The data set containing the vibrations had three primary features (the spatial dimensions) along with a fourth time-stamp feature, and about 8 700 000 observations. The data set containing the internal engine variables contained 98 features and about 50 500 observations.

Before starting to process the data, a decision had to be made whether to, and if so how, to combine the engine variables and the vibration data sets. The vibration data has observations sampled every 2-3 milliseconds, while the engine variables had observations sampled every few hundred milliseconds. The alternatives were to under-sample the vibration data into the engine variable data set, or to over-sample the engine variable data into the vibration data set by interpolating the missing observations. It was decided that under-sampling the vibration data is more likely to produce a reliable model in contrast to interpolating the engine variables, as it is difficult to know what interpolation method to choose. Also, some of the chosen methods are computationally expensive, making the smaller sized data set more convenient. The resulting data set contained 50512 observations with 104 features, with some of the extra features being related to synchronizing the timestamps between the two data sets.

## 3.2 Data cleaning and feature selection

Early analysis of the data showed that there were no missing/empty values in the data set. However, some of the features turned out to contain no variation, in other words all the observations contained the same value for a feature in question. This presented an opportunity to split the data into two sub-models: sub-model **A** containing all the features with no variation, and sub-model **B** containing all the features with $\geq 2$ unique values. This way, sub-model B has reduced dimensionality, making the modelling process slightly more convenient,

and sub-model A can flag for anomalies whenever any of its selected features observes a new value. The idea of sub-model A falls directly under the main assumption of this thesis, namely that the data represents the normal condition of the engine, and deviations possibly implicate anomalies. After this cleaning step, and after removing the timestamp synchronization data, and also normalizing the data using the z-score metric the resulting data set for sub-model B had 50512 observations with 75 features. As sub-model A is trivial, the rest of this thesis will focus on building sub-model B.

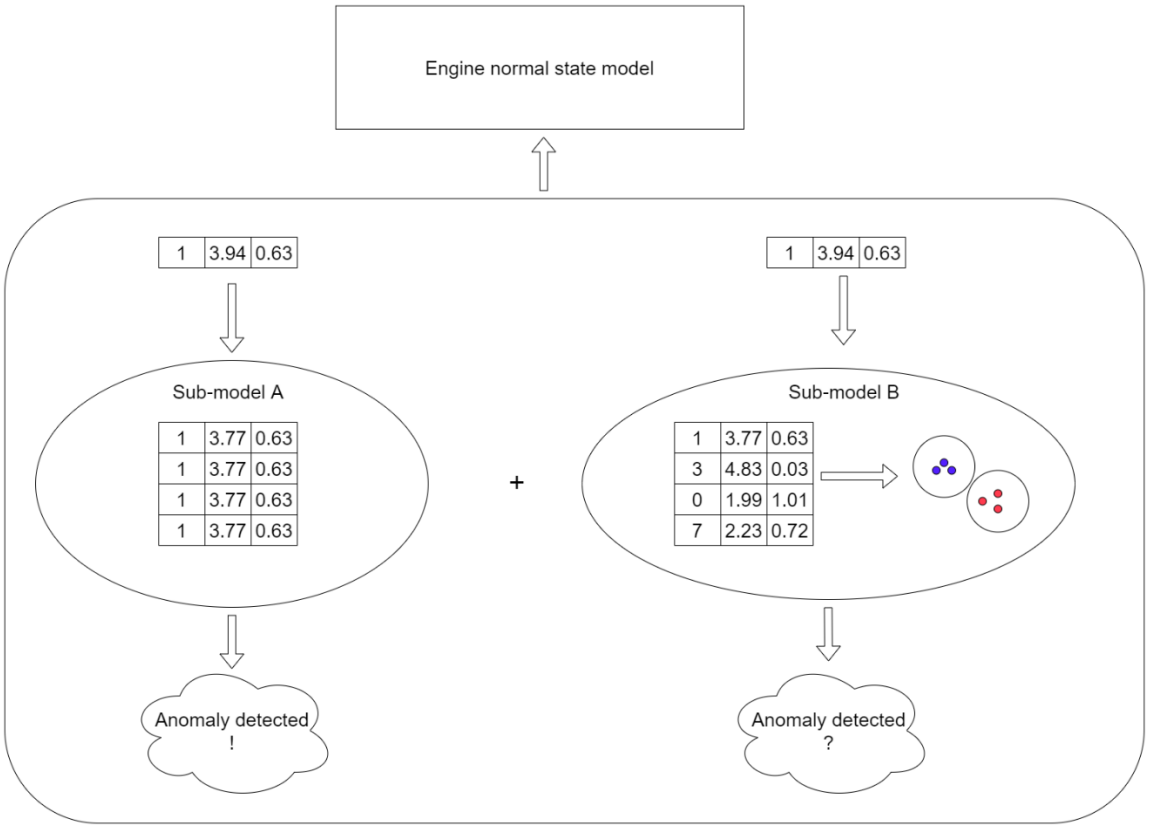


*Figure 10. Engine normal state modelling approach.*

## 3.3 Exploratory data analysis with SOM

To get a general view of the data, a Self-Organizing Map (SOM) was used. This reduces the dimensionality of the data into two-dimensional heat maps, allowing for a visual representation of the relationships between each variable. Figure 11 shows the collection of SOM images for the data set. It can be seen that several features are highly correlated or inversely correlated. As an example, the heat maps on the marked with red circles show very similar patterns in their heat maps, indicating that values in these features are highly correlated. Similarly, the values marked with blue circles are highly correlated, and inversely

correlated with the two previously mentioned features on the same row that were marked with red circles, as they have more or less inverted heat maps. These correlations have significant implications on the interpretation of the results in this work; these implications will be discussed in chapter 5.

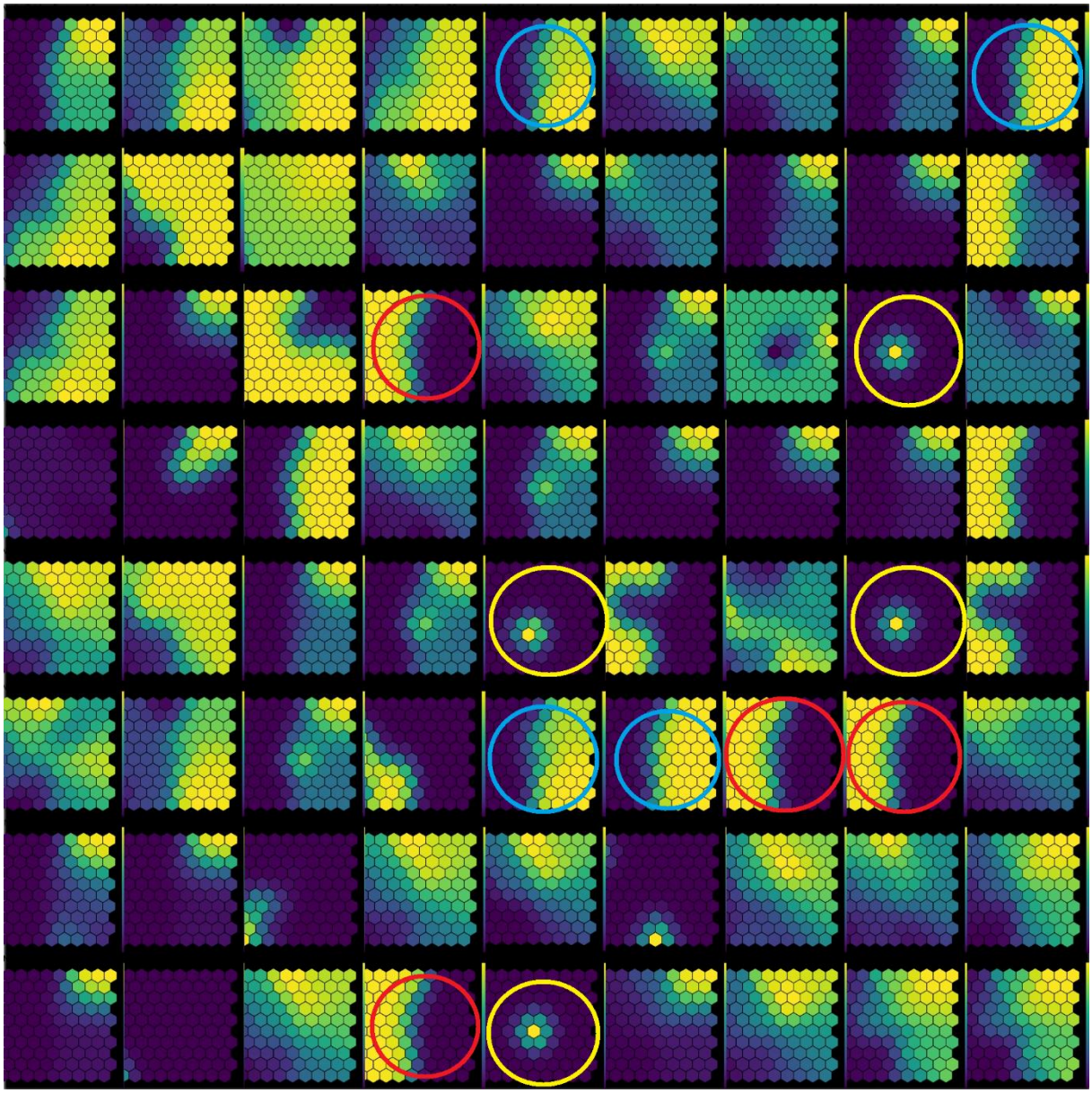

*Figure 11. Self-Organizing Maps of 72 of the attributes of the engine data set. The accelerometer vibrations are omitted from this collection. Features circled with the same colour are examples of highly correlated features. The red and blue circles are roughly inversely correlated. SOM and image generated with SimpSOM Python library (Comitani, 2022)*

The SOM also produces a Unified Distance Matrix (U-MAT). Figure 12 shows the resulting U-MAT for the working data set. The colours of the cell represent the distance between adjacent neurons of the SOM. A small distance between neurons corresponds to a small distance between vectors in the input space, and a large distance between neurons *sometimes* corresponds to a large distance between input vectors (it is important to remember that a

dimensionality reduction always loses some information, and the results are approximations). This means that the U-MAT gives a higher-level view of the entire data set and relationships between values there. In Figure 12 there can be found roughly 4 or 5 very dark parts, separated by lighter cells. This could indicate that there are 4 or 5 clusters, or states that the engine takes when it is running. These states could be things like warming up, running at full capacity, off, etc.

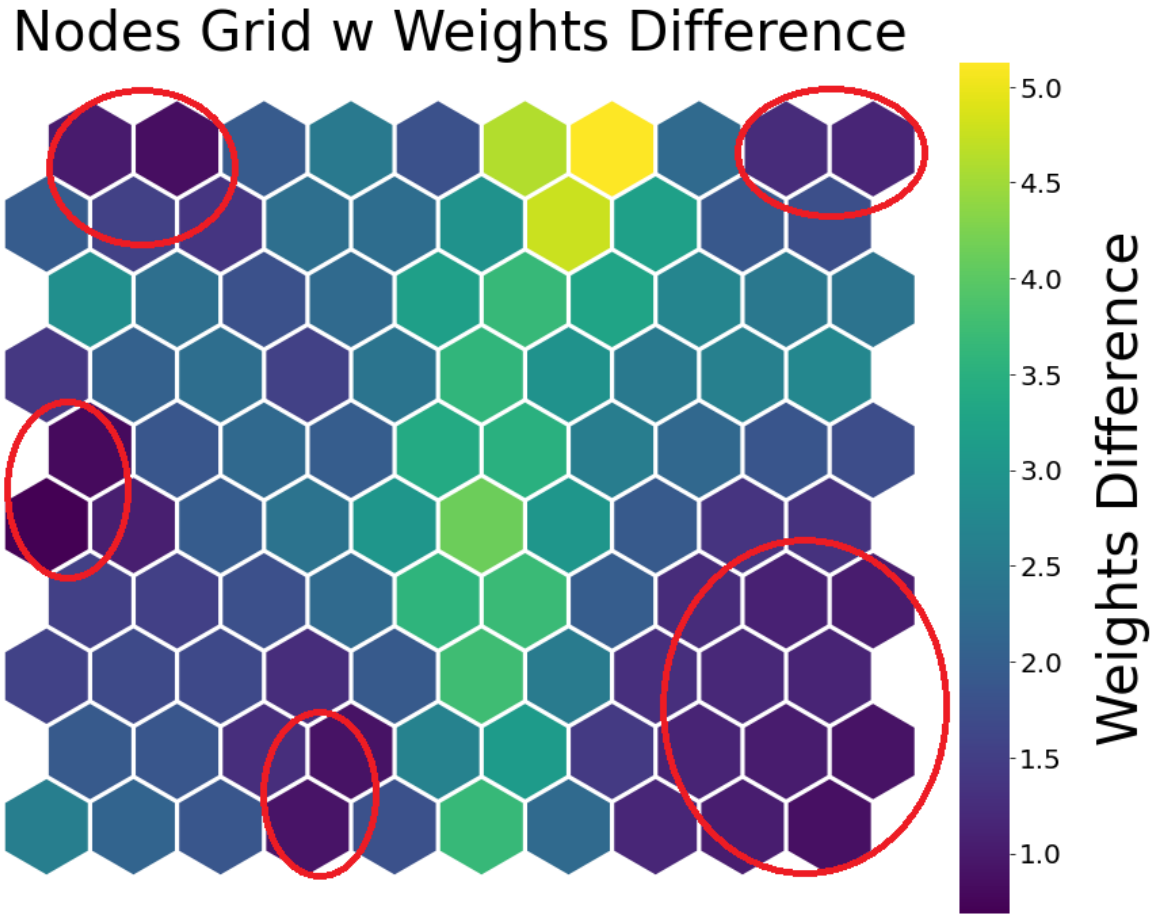


*Figure 12. Unified Distance Matrix of the SOM features. Groups of dark cells separated by borders consisting of lighter cells can be considered as clusters. Image produced with the SimpSOM Python library (Comitani, 2022)*

### 3.4 Model selection

For the experiments the K-means, DBSCAN and spectral clustering algorithms were compared. For K-means and spectral clustering, the number of clusters needed to be determined as an input hyperparameter. The methods of silhouette coefficient and the eigengap heuristic were chosen as methods to determine the optimal number of clusters for K-means and spectral clustering respectively. For DBSCAN, silhouette coefficient was used to determine the hyperparameters epsilon and minPts. The Python library scikit-learn (Pedgregosa, et al., 2011) was used as the implementation of choice for the experiments.

For further visualizations, t-distributed Stochastic Neighbourhood Embedding (t-SNE) was chosen as a dimensionality reduction method to project the data into three dimensions. Visualizing the data is useful to get an understanding of the results. As data spaces beyond 3 dimensions cannot be properly visualized, methods are needed to reduce the dimensionality of the data in such a way that it can be projected into a lower (commonly 2- or 3-) dimensional space, while preserving as much of the original information as possible. There are several popular methods for dimensionality reduction, including Principal Component Analysis (PCA) or Uniform Manifold Approximation and Projection (UMAP). What makes t-SNE particularly interesting is that it is known to approximate a spectral clustering scheme (Linderman & Steinerberger, 2019). Something to keep in mind is that the t-SNE projection can produce "visual clusters" that do not actually exist in the input data. However, (Linderman & Steinerberger, 2019) contains a proof that if the original data is well-clustered (as defined in the paper), then the t-SNE output will be well-clustered also.

To test the validity of the chosen methods, the clustering algorithms were run on a synthetic data set before using them on the actual data that the thesis explores. This benchmark shows how the methods work under favourable conditions.

## 3.5 Synthetic data set characteristics

For testing purposes, a data set from the *clustering basic benchmark* (Fränti & Sieranoja, K-means properties on six clustering benchmark datasets, 2018) by University of Eastern Finland was used. The data sets in the benchmark are designed such that typical heuristics will fail, but the sum-of-squares-error (SSE) objective function can solve the clusters. There are different sets of data in the benchmark with varying characteristics, e.g., a varying number of clusters, varying overlap of the clusters, varying dimensionality, and so on.

For the algorithm tests, data set $S_1$ was chosen from the clustering basic benchmark. The S data sets contain a fixed number (15) of Gaussian clusters with varying overlap. Since the clusters are generated by a Gaussian process, most of them are spherical, but a few have been truncated. The data set and its ground truth is available at (Fränti & Sieranoja, K-means properties on six clustering benchmark datasets, 2018), but the general characteristics of the data will be described next.

Table 3 shows the characteristics of some of the different data sets in the benchmark, including for $S_1$. The data distributions are defined by Overlap, Contrast, Intrinsic dimensionality, and H-index.

Overlap occurs when data points belonging to a cluster **A** are closer to the centroid of a different cluster **B** than to the centroid of **A**. The Overlap score is calculated in the following way:
Let the distance from a data point to its centroid be $d_1$, and the distance to the nearest point in another cluster be $d_2$. An overlap is defined as

$$ov(d_1, d_2) = \begin{cases} 1, d_1 > d_2 \\ 0, d_1 \leq d_2 \end{cases}$$

Then

$$Overlap = \frac{1}{N} \sum ov(d_1, d_2)$$

Contrast measures variation in distances. To measure the contrast of a point, the distances to its nearest and furthest neighbour are needed ($d_{min}$ and $d_{max}$). With these quantities the contrast is defined as

$$Contrast = median(\frac{(d_{max} - d_{min})}{d_{min}})$$

Intrinsic dimensionality is a sort of estimation of the complexity of the data set beyond just the number of features. This feature is defined with the following formula:

$$IntrinsicDim = \frac{\hat{d}^2}{2\sigma^2}$$

where $\hat{d}$ is the mean distance between all points and $\sigma^2$ is the variance. The idea of intrinsic dimensionality is that, as an example, if a data set has 10 features/dimensions, but all of them contain exactly the same values, then it can be thought that the data set has only 1 “true” or intrinsic dimension.

Finally, to define the H-index, an attribute called *the hubness score* is needed. The hubness score for a point x has been defined as the number of points that consider x as its k-nearest neighbour in a kNN graph. Then, to calculate the H-index, the data points are ranked according to their hubness scores from highest to lowest, and the H-index is **the highest rank**

**for which the hubness score is greater than or equal to its rank**. Formally, if *i* is the rank of the data point, and *f(i)* its hubness score, then:

$$h - index = \max\{i \in \mathbb{N} : f(i) \geq i\}$$

| Data set name | Overlap | Contrast | Intrinsic dim. | H-index |
|---|---|---|---|---|
| $A_1$ | 20 % | 227 | 1,5 | 2 |
| $A_2$ | 20 % | 261 | 2,0 | 3 |
| $S_1$ | 9 % | 320 | 2,2 | 2 |
| $S_2$ | 22 % | 257 | 2,2 | 3 |
| $S_3$ | 41 % | 210 | 2,0 | 3 |
| Birch2 | 4 % | 8308 | 2,6 | 3 |

*Table 3. Characteristics of some of the data sets of the clustering basic benchmark. Set $S_1$ is used for the tests in this thesis.*

## 3.6 Outlier detection

Outlier detection is a complex and heavily studied field that contains many interesting opportunities and methods to test on the data sets. To limit the scope of the thesis, a few methods were selected for comparison.

DBSCAN marks some data points as outliers automatically, so the algorithm's results themselves were chosen as the first method for marking outliers. Moreover, the number of outliers found by DBSCAN was chosen as *n* for the other methods (how many outliers to highlight); this allows for a direct comparison of the marked outliers with the other methods.

Recall the three assumptions for anomaly detection by cluster analysis from chapter 2.4. For a centroid-based clustering algorithm like K-means, one can work with *assumption 2*: *Normal data instances are closer to the cluster centroids, while anomalies are far away from the cluster centroids.* Thus, the second of the used methods was to calculate the distances of all data points within each cluster obtained by K-means to its corresponding centroid, and mark *n* data points that were furthest from their centroid as outliers.

When it comes to the spectral clustering algorithm, the clusters do not necessarily have specific centroids, as the clusters can have arbitrary shapes. For this reason, a simplification was made in that "pseudo-centroids" were calculated by taking the average of each data point within a cluster, like in K-means. Notably this is sometimes a very crude approximation of the ground truth, but it allows for applying the next outlier scoring method on the results of the spectral clustering algorithm.

As the second method for marking outliers a *reduced kernel density estimation* was chosen. Kernel density estimation (KDE) is a nonparametric way of estimating the probability density function (PDF) of a distribution. To get an idea of how it works: let $\{x_1, x_2, \ldots, x_n\}$ be an *i.i.d.* (independent and identically distributed) sample from a distribution with an unknown density *f*. The kernel density estimation of the PDF f is defined as

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K(\frac{x - x_i}{h})$$

where K(.) is the *kernel function*, and h > 0 is known as the *bandwidth*.

In this work, the Gaussian function is chosen as the kernel function. The kernels are "placed" on the cluster centroids obtained by spectral clustering (in other words, the set of pseudo-centroids is the sample for the KDE formula), and the resulting model is used to calculate an *outlier score* to each data point in the data set. Then, *n* points with the largest outlier score are marked as outliers. The KDE is referred to as *reduced*, since it performs the density estimation based on the cluster centroids rather than the entire clusters or data set; here, the assumption is that the centroids provide a representative sample for how the data is distributed.

It should be noted that a key difference between the two previously mentioned outlier marking methods is that the method used on K-means flags outliers locally with regards to each individual cluster, whereas the kernel density estimation calculates outliers with regards to the entire data set.

# 4 Experimental results

## 4.1 Synthetic data benchmarking

To benchmark the chosen clustering algorithms, they were applied on the synthetic data set $S_1$ from the clustering basic benchmark described in the previous chapter. The synthetic data is two-dimensional, consisting of 15 Gaussian clusters, some of which are truncated, and with varying overlap.

### 4.1.1 DBSCAN benchmark

The DBSCAN hyperparameter settings were tuned by using the silhouette score-based optimization. The best score was obtained with epsilon value 0.1 and minPts value 9; Figure 13 shows a plot of the optimization results. When DBSCAN assigned all points to a single cluster the silhouette coefficient couldn't be calculated, in these cases it was set to 0. With these settings, DBSCAN identified 15 clusters as in the ground truth, and marked some of the points that are furthest from their centroids as outliers. The DBSCAN partitioning and outliers are shown in Figure 14.

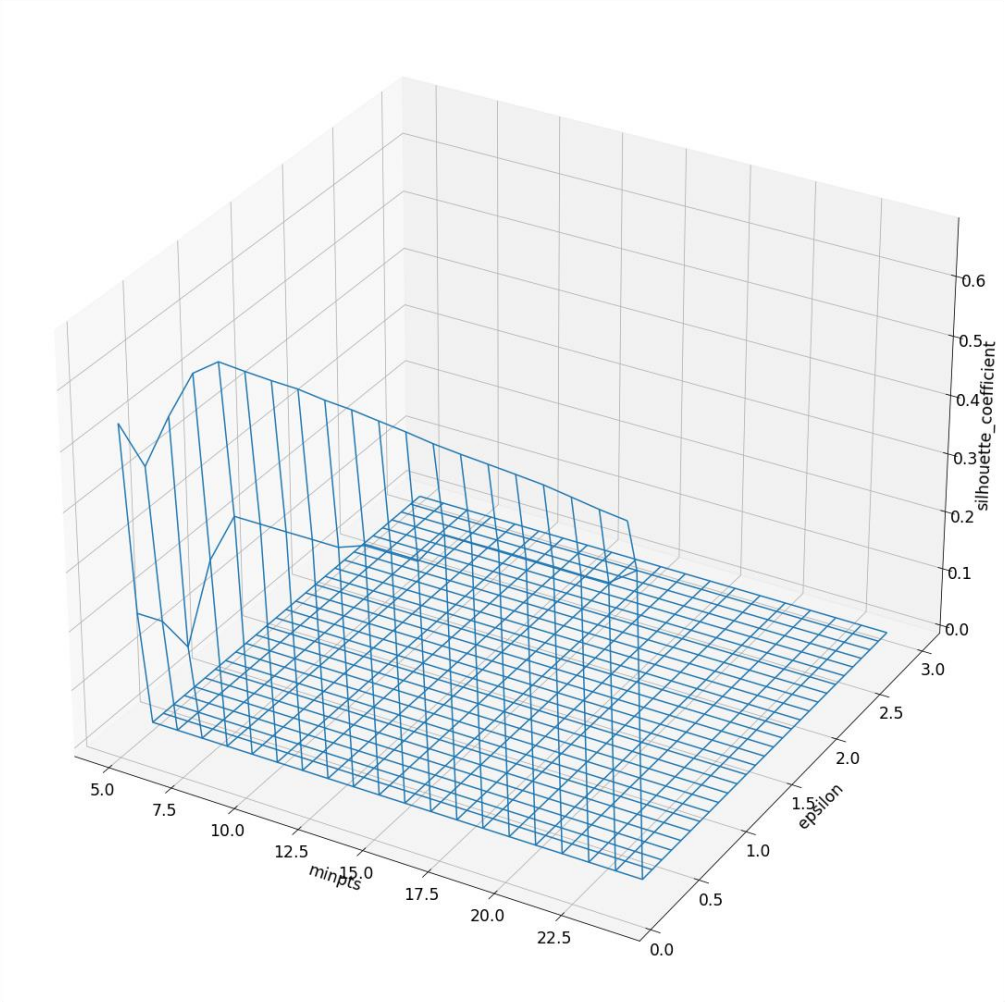


*Figure 13. Hyperparameter optimization result for DBSCAN on the clustering basic benchmark. The higher up the grid value the better.*

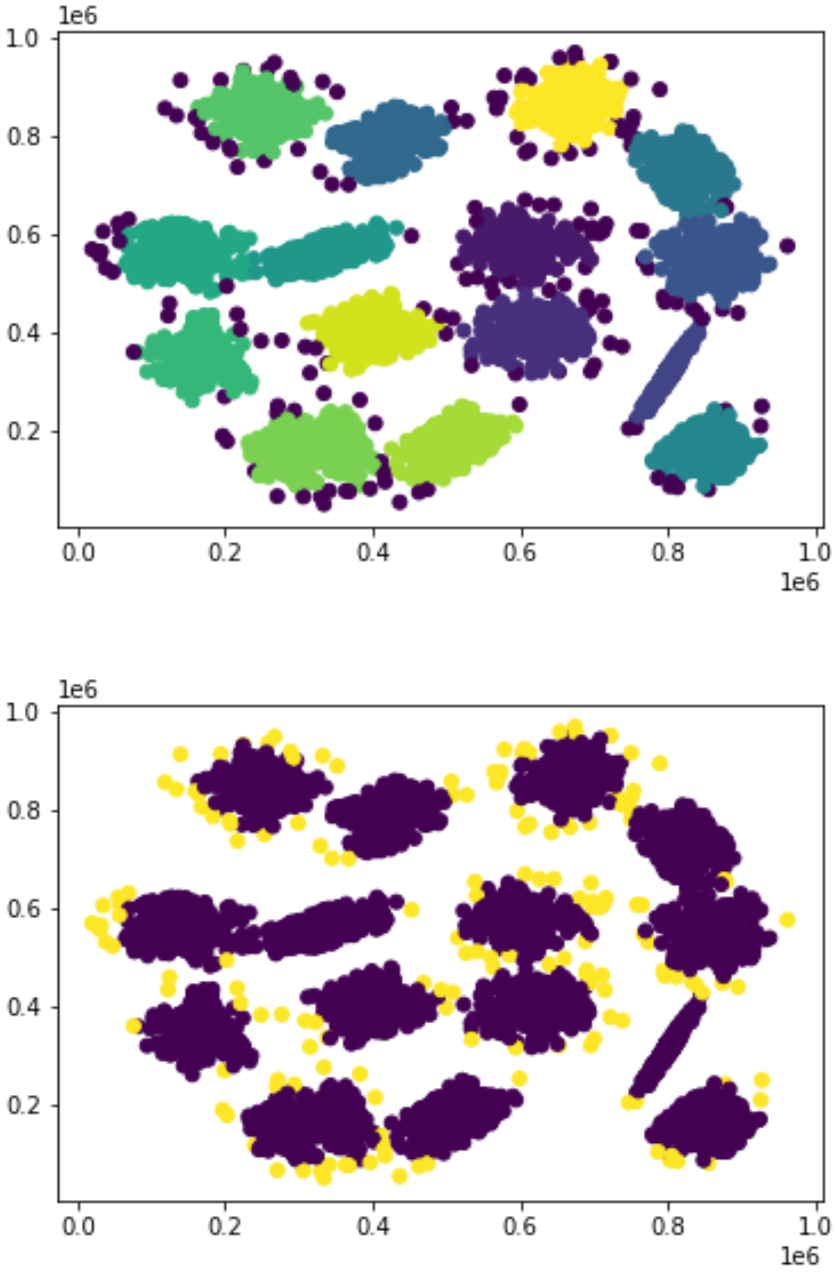


*Figure 14. DBSCAN on synthetic data. The second figure illustrates only the outliers, marked with yellow colour.*

### 4.1.2 K-means benchmark

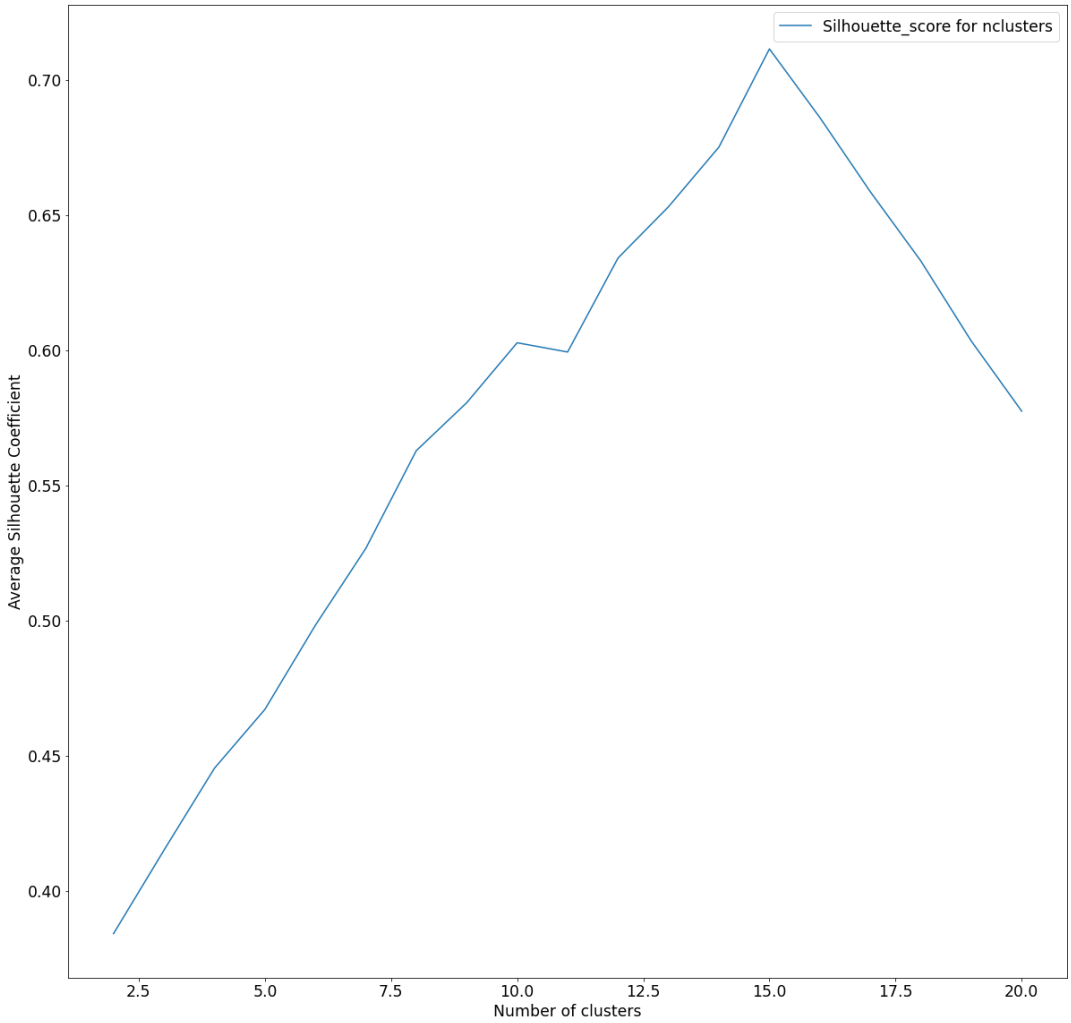


*Figure 15. K-means silhouette coefficient hyperparameter optimization.*

Figure 15 shows a plot of optimizing the number of clusters k for K-means on the clustering basic benchmark. The best silhouette score is obtained with 15 clusters, as should be expected

with knowledge of the data ground truth, and since this benchmark is explicitly designed to test K-means. Figure 16 shows the cluster partitioning identified by the algorithm.

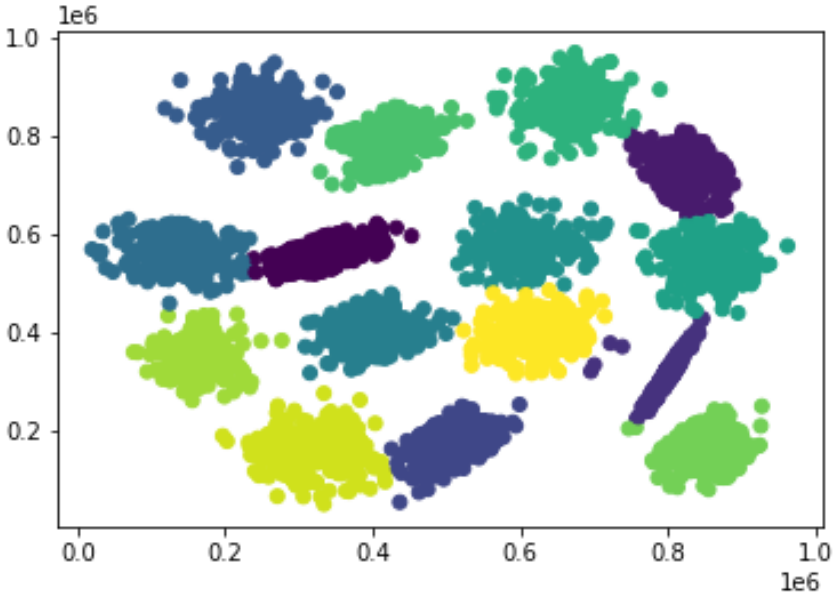


*Figure 16. K-means on synthetic data.*

With the outlier marking method assigned to K-means in chapter 3, namely assigning *n* data points that are the furthest from their centroid as outliers, we get the outliers as seen in Figure 17. The results are very similar to that of DBSCAN.

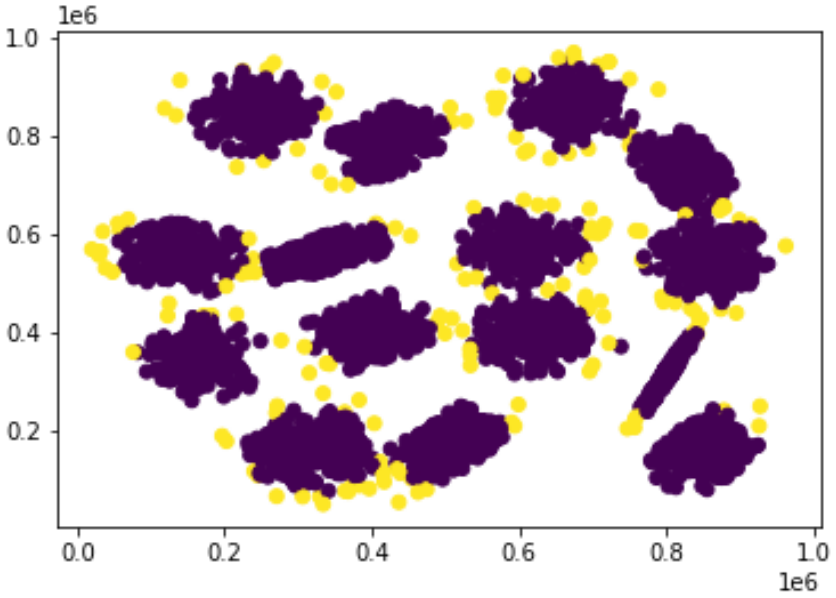


*Figure 17. Outliers marked for K-means in yellow.*

### 4.1.3 Spectral clustering benchmark

The eigengap heuristic was tested to see if one can find the correct number of clusters with its help. Figure 18 shows the resulting eigenvalues. One can note a gap at eigenvalue 16, and a larger one at 17, suggesting that 15 or 16 clusters would be suitable for the data set.

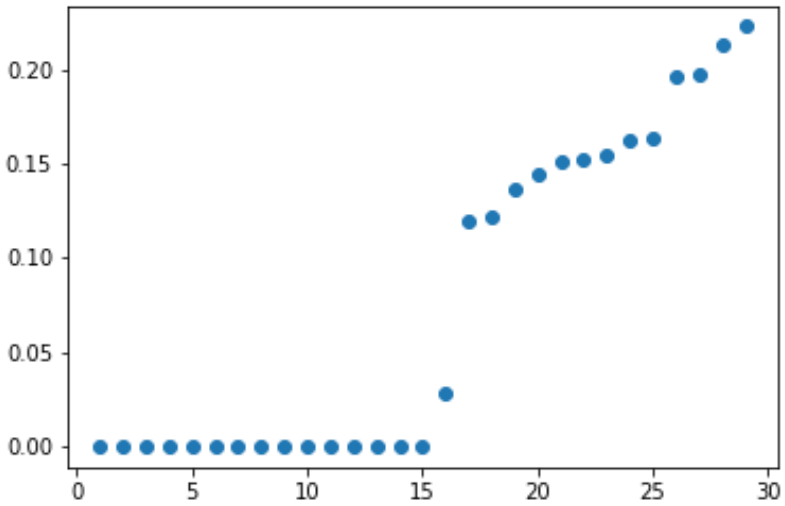


*Figure 18. Eigenvalues for synthetic data.*

The choice of number of clusters is not immediately obvious between 15 and 16; on one hand, the gap between 16 and 17 is larger, but on the other hand, 16 is at the Fiedler value, i.e., the first non-zero value, which is often considered to be a good estimation for the number of clusters by itself (von Luxburg, 2007). With this in mind, 15 was chosen as the number of clusters for the spectral clustering algorithm. Figure 19 shows the partitioning obtained by the algorithm.

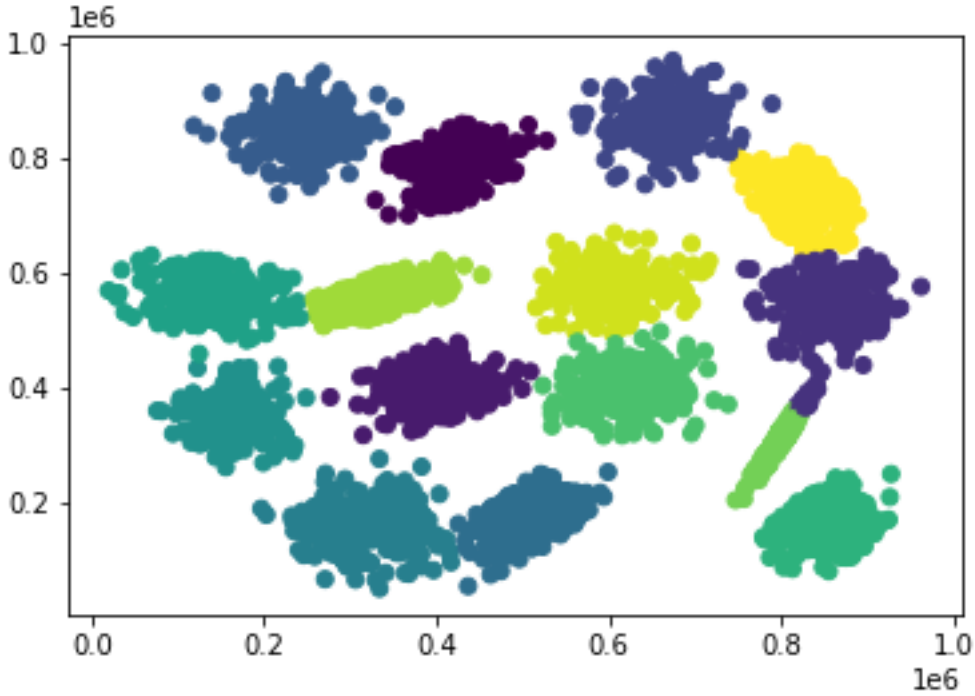


*Figure 19. Spectral clustering synthetic data.*

For the spectral clustering method, the resulting clusters do not necessarily have specific obvious centroids, as they can have arbitrary shapes. To test an outlier detection method on data set $S_1$ and spectral clustering, an approximation for centroids was calculated by taking the average of the data points for each cluster. In data set $S_1$, this obviously results in a good approximation of centroids, as the identified clusters are close to the ground truth and have regular shapes. However, in a real setting, the approximation may not be as good.
Based on the calculated centroids, a reduced kernel density estimation (KDE) was applied on the data, with the kernels placed on each centroid. The KDE hyperparameters were chosen with a GridSearch cross-validation.

With the KDE model, each sample was scored for their log-likelihood under the model. Then, based on the log-likelihood, the samples were assigned an outlier score based on the formula:

$$outlier\_score = \frac{-model\_scores}{norm(-model\_scores)}$$

where model_scores is the vector of log-likelihood scores assigned by KDE, and norm is a vector 2-norm defined as

$$||x||_2 = \sqrt{\sum_{i=0}^{m-1} |x_i|^2}$$

for an m-dimensional vector. This allows for a normalization of the model scores. The outliers identified by this method interestingly differ from the ones found by K-means and DBSCAN and can be seen in Figure 20.

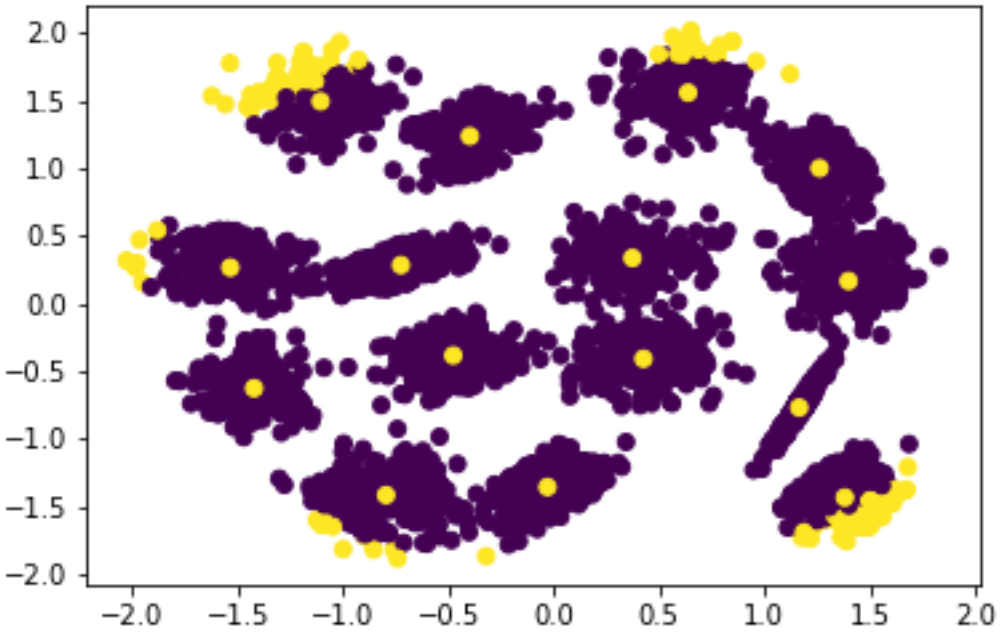


*Figure 20. Centroids and outliers found for spectral clustering and KDE marked with yellow.*

## 4.2 DBSCAN results on real data

To start the experiments on the real engine data set, a GridSearch cross-validation was run to find optimal hyperparameters for the DBSCAN algorithm. Silhouette coefficient with Euclidean distance was used as the scoring function. Because of computational constraints, the GridSearch was limited to a 2-fold cross-validation.

What stands out is that with the "optimal" hyperparameters DBSCAN flags almost half of the data (n=22804) as outliers. This is obviously unreasonable, as pre-existing knowledge of the running engine suggests that everything was normal during the run. The reason for this behaviour is probably in that silhouette coefficient with Euclidean distance as the metric is not a suitable scoring function for validating DBSCAN on the data set in question, considering its high dimensionality (the reasons will be discussed in chapter 5). Figure 21 illustrates the data

set as a timeline, with normal data marked with dark colour and outliers marked with yellow colour.

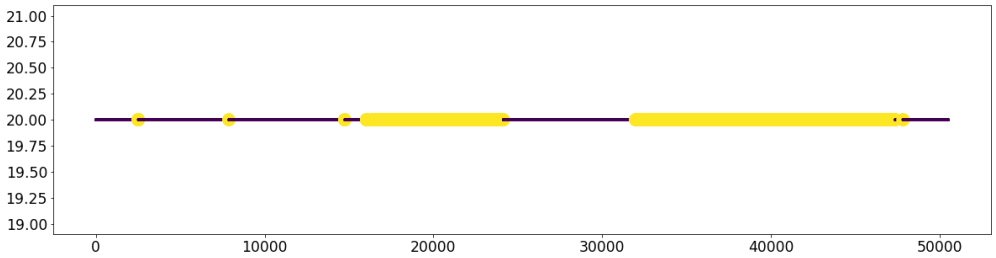


*Figure 21. DBSCAN with silhouette coefficient cross-validated outliers, as a timeline. Yellow markers are outliers. This method marked 22804 data points out of 50512 as outliers, which is not a reasonable result. Hyperparameters were epsilon = 0.4, minPts = 20.*

Considering the following facts: 1) The engine run is presumed to consist of mostly normal data, and 2) silhouette coefficient seems unfit to evaluate the clustering results, a new question is what criterion to use as a scoring function to find optimal hyperparameters? One option could be to run a GridSearch that minimizes the number of outliers. While it can't be guaranteed in advance that the method flags the correct data as outliers, it is still interesting to see what hyperparameters are found and what DBSCAN finds with these hyperparameters.

Figure 22 shows a plot of minPts-Epsilon pairs and corresponding error in the hyperparameter search; the search was conducted as a 5-fold cross-validation GridSearch with minPts ranging between 5 and 20 in increments of 1, and epsilon ranging from 0.1 to 3.0 in steps of 0.1. The smallest number of outliers found by DBSCAN with these hyperparameter combinations was

177. It should not come as a surprise that increasing the value of epsilon and minPts results in fewer outliers, as this will result in larger clusters being considered.

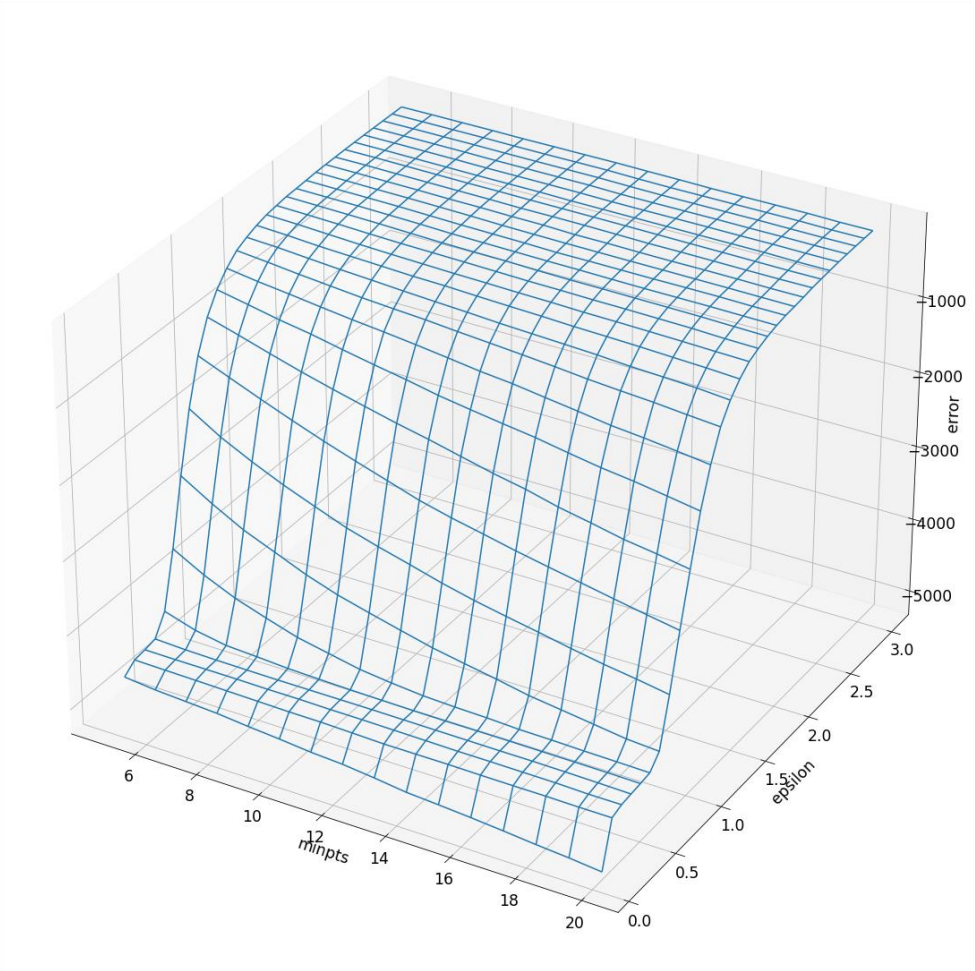


*Figure 22. Hyperparameter pairs that minimize the number of outliers that DBSCAN finds. As the grid-search is a maximizing function, and we seek to minimize the outliers, the error score is a negative value of the number of found outliers; higher value is better.*

Figure 23 shows where the outliers marked by DBSCAN lie on the original time-series. The timeline with the fewest outliers was chosen for this picture.

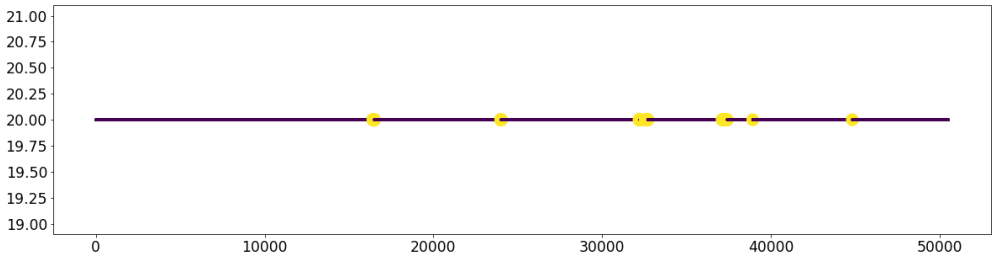


*Figure 23. Outliers marked by DBSCAN on a timeline. Number of outliers is 177, the smallest in the tests and number of clusters found is 27.*

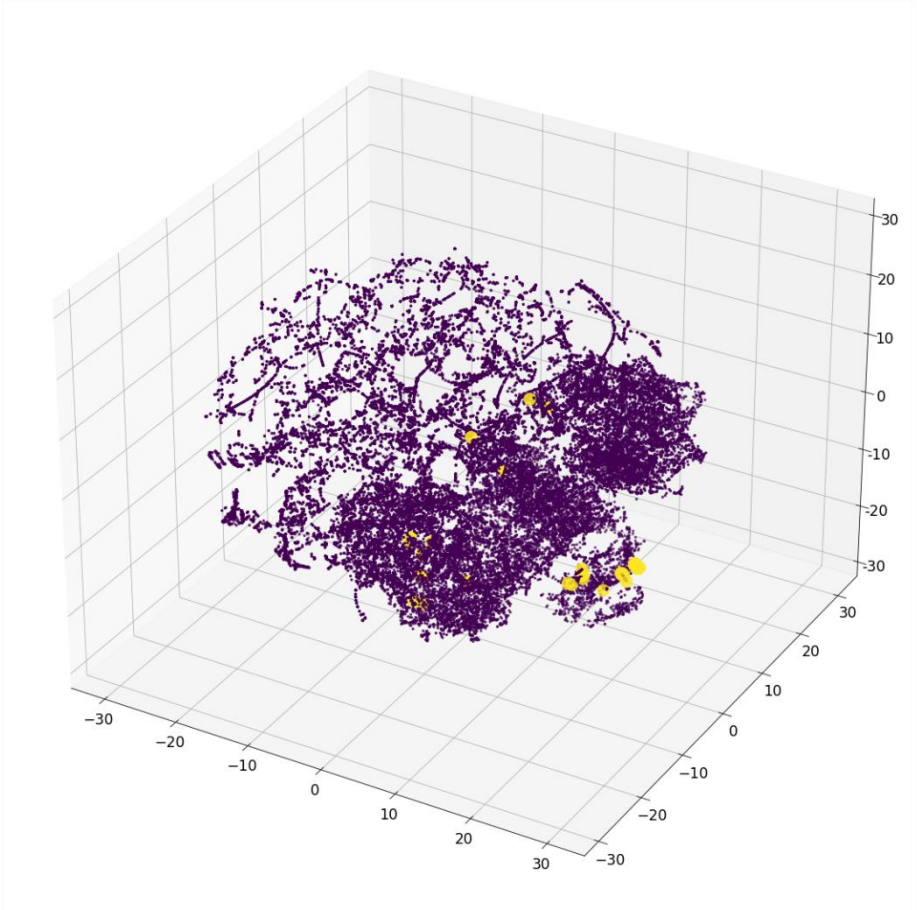


*Figure 24. DBSCAN outliers in a t-SNE projection. Outliers are marked with yellow colour.*

Figure 24 shows the same outliers, projected into a 3D-data space with a t-SNE dimensionality reduction.

## 4.3 Number of clusters for K-means and spectral clustering

As mentioned in the previous chapters, the number of clusters must be determined beforehand for K-means and spectral clustering methods.

For spectral clustering, the eigengap heuristic was used. As the method is computationally expensive, the computation was run on several subsets of the data. Figure 25 shows the results of the computations. Notably, on several of the subsets a suitable gap seems to appear between 20 and 25, indicating that 20-25 clusters might be an appropriate choice for the method. On the other hand, the Fiedler value sometimes appears at eigenvalues around 16 to 20. In these conditions, the choice of number of clusters was difficult even with the heuristic. The choice was made to run spectral clustering with the number of clusters set to 17, 20, and 24.

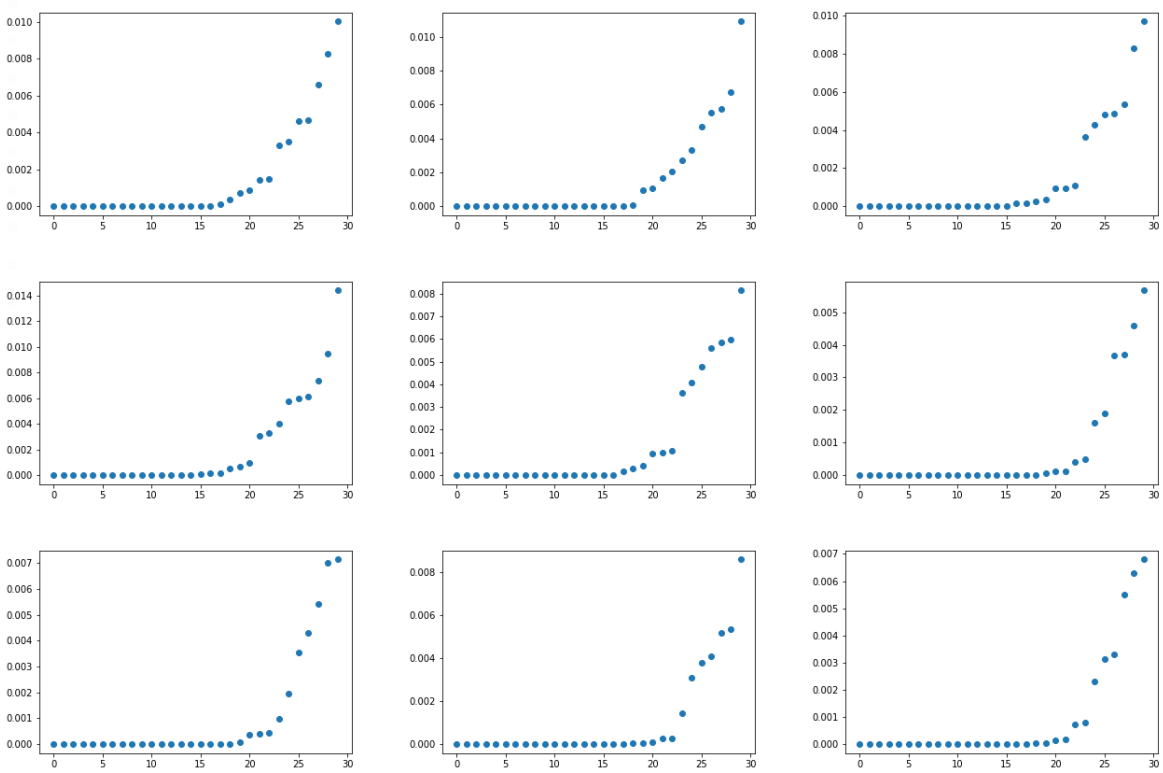

*Figure 25. Eigenvalues for subsets of the engine data.*

For K-means, an initial estimation for the number of clusters was obtained with a GridSearch cross-validation. Some of the results are shown in Figure 26.

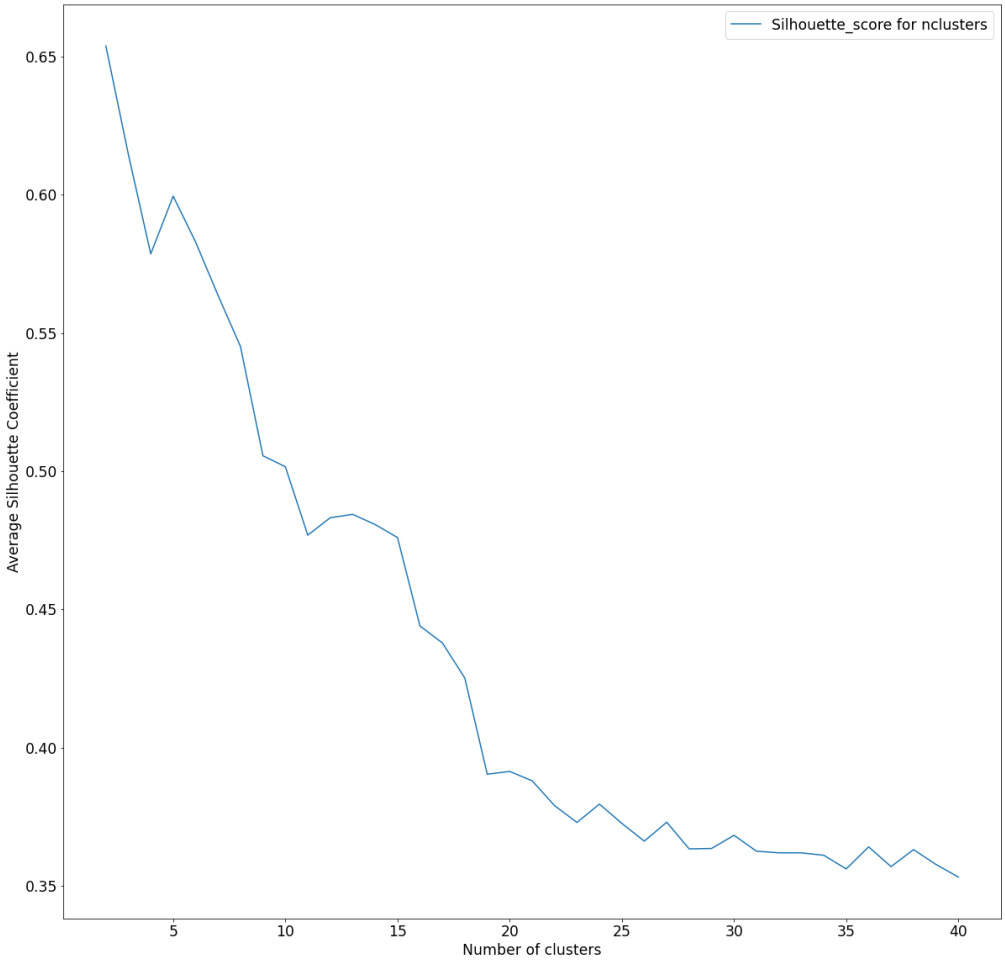


*Figure 26. Silhouette coefficient for different values of k in K-means run on the engine data.*

Just like for DBSCAN, silhouette coefficient with Euclidean distance was used as the scoring function. Surprisingly, the best score was obtained with k = 2. However, as silhouette coefficient was deemed unfit for DBSCAN, other methods should probably be considered to determine the number of clusters for K-means as well. Taking this into consideration, K-means was run with different values for k:

- K = best estimator given by the silhouette coefficient scored grid search
- K = number of clusters estimated by eigengap heuristic

- K = number of clusters found by DBSCAN grid search scored by minimization of outliers

As such, the final set of candidates for k in K-means was {2,5,17,20,24,28}.

## 4.4 K-means outliers

The outliers for K-means were again tagged according to *assumption 2* from chapter 2. Figure 27 shows the distribution of outliers when the data is projected into a three-dimensional space, and Figure 28 shows the distribution of outliers when the points are arranged as a timeline.

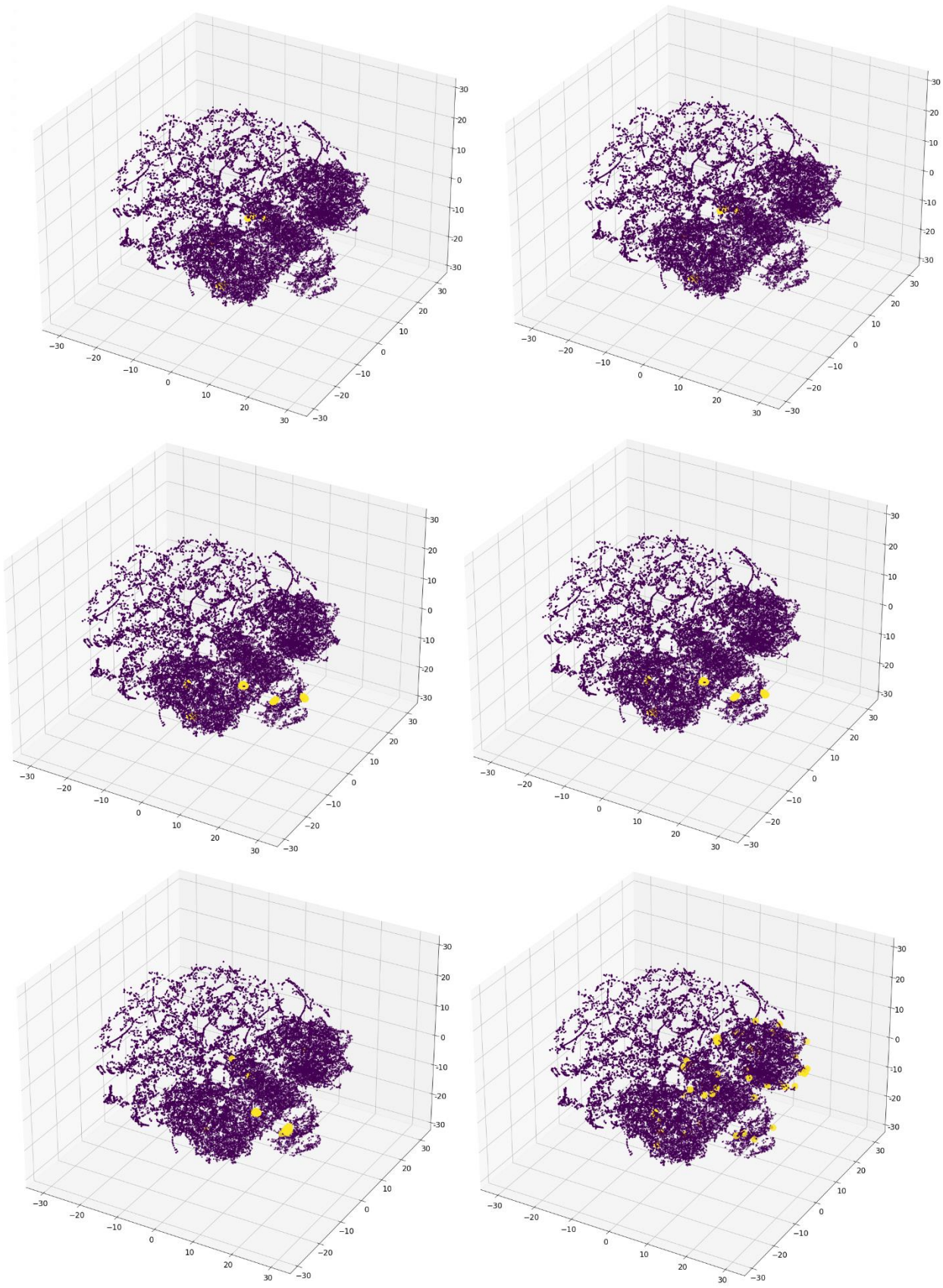

*Figure 27. K-means outliers found under assumption 2, for k = 2,5,17,20, and 28.*

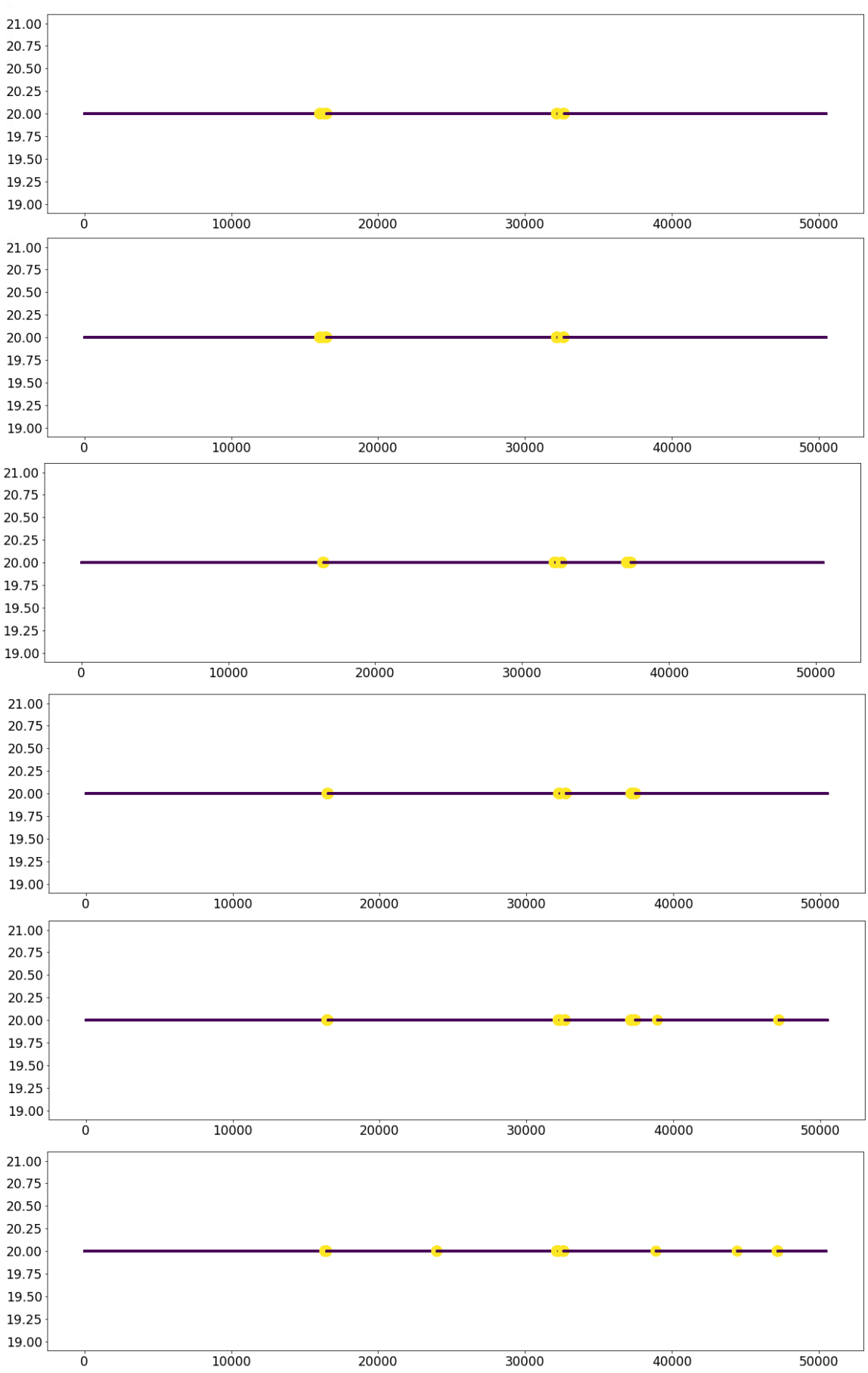
21.00
20.75
20.50
20.25
20.00
19.75
19.50
19.25
19.00
0
10000
20000
30000
40000
50000

*Figure 28. Same K-means outliers shown where they appear in the time-series.*

## 4.5 Spectral clustering outliers

For spectral clustering, outliers were assigned with the methods discussed in chapter 3: pseudo-centroids were calculated for each cluster as the average of each point, and a reduced kernel density estimation was performed based on these centroids. Figure 29 shows the timeline in the same fashion as for the previous algorithms, with normal data marked as dark coloured dots and outliers marked as yellow-coloured dots. Figure 30 shows the outliers in the 3D-projected data space.


*Figure 29. Spectral clustering with kernel density estimation tagged outliers as a time series.*

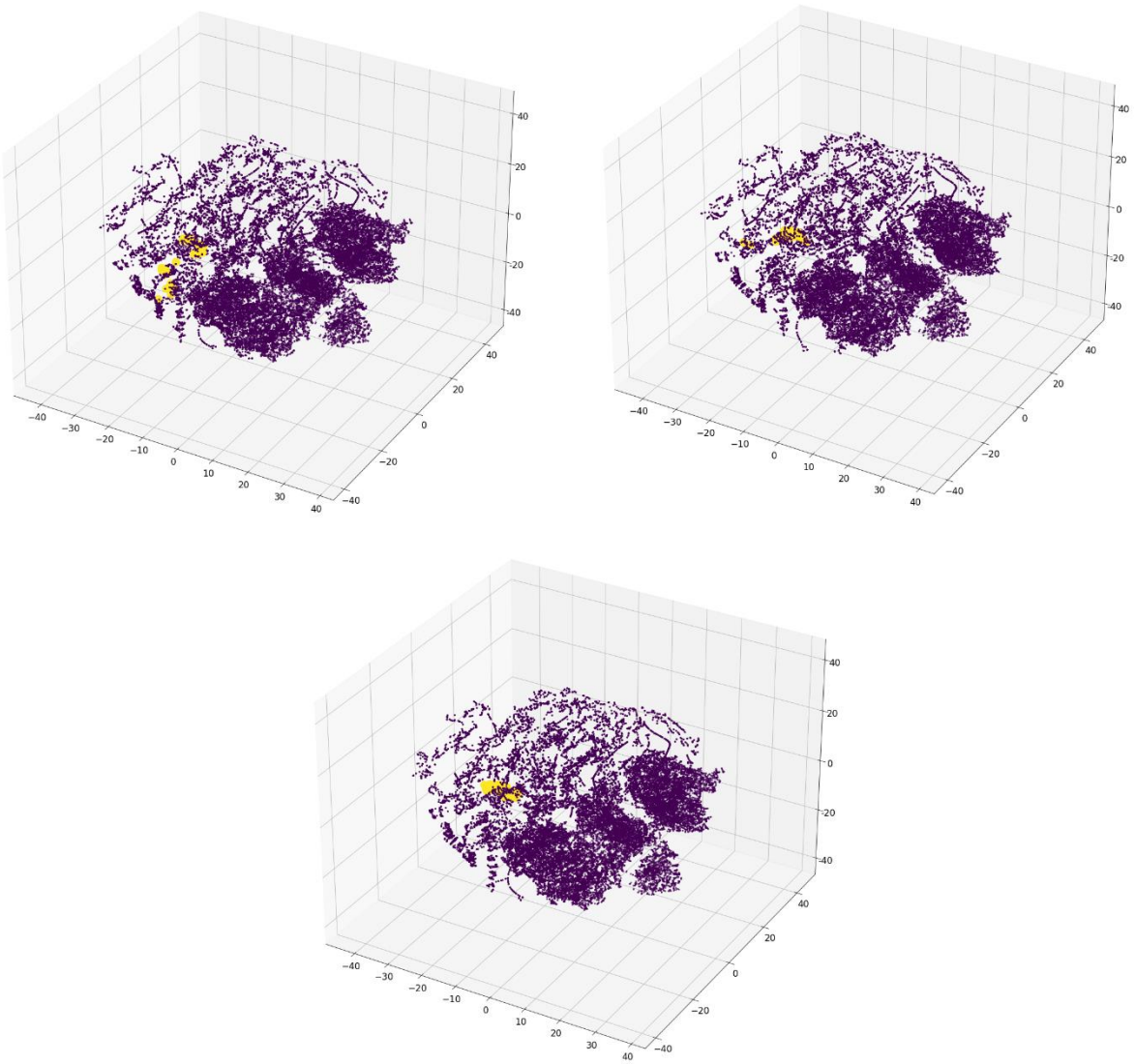

*Figure 30. Spectral clustering with kernel density estimation tagged outliers as a 3D data space projection.*

Notably, the marked outliers are in similar regions in each of the tests.

# 5 Conclusion and limitations

In this thesis three different clustering methods from different paradigms were used to perform anomaly detection on a marine vessel diesel engine, run in a laboratory setting. The clustering algorithms K-means, DBSCAN, and spectral clustering were chosen for comparison. Each clustering algorithm was paired with an outlier scoring method, and the resulting flagged outliers were compared with each other. All methods were also run against a basic clustering benchmark data set, to show how they perform under favorable conditions. Exploratory data analysis in the form of dimensionality reduced visualizations were produced with Self-Organizing Maps and t-Distributed Stochastic Neighborhood Embedding.

As a first observation, none of the clustering methods flagged the exact same subset of data as outliers. Notably, DBSCAN and K-means-based methods marked outliers in similar regions to each other and differed remarkably from the results obtained by spectral clustering with the reduced kernel density estimation. Most likely this is due to the fact that the first two algorithms make locally based decisions on how to mark outliers, whereas the KDE looks at the outliers with regards to the entire data set. As such, the differing results are more likely dependent on the chosen method of KDE, rather than the spectral clustering method alone.

From all the figures, one can see that where data points lie in the 3D-projection is not completely arbitrary; rather, it seems to be closely related to each datum's position in the original time-series. Outliers marked by spectral clustering/KDE are on the left side in the 3D-projection, and on the timeline, they are in the first quarter. Outliers marked by K-means/distance-flagging and DBSCAN are more on the right side of the 3D-projection, and sure enough they are distributed more on the later parts of the timeline. From this one can conclude that in this case, t-SNE produced a projection that is useful in exploring the data and how it is distributed. One has to remember the pitfalls and limitations of t-SNE, but taking that into consideration the produced projections are still informative.

As there was no pre-existing information on whether the data contained anomalies or not, the most reasonable approach was to model the normal behavior of the engine, which is the reason why clustering methods were chosen. All in all, it is not completely clear after these experiments which algorithm produced the most reliable model. Determining which, if any, of the marked outliers most realistically are actual anomalies would require a review by an

expert. What can be concluded is that what data are marked as outliers is highly sensitive to the procedure, from the selection of algorithm to the hyperparameter tuning and feature engineering. Further work would benefit from cooperation with subject matter experts to review these current results, and better more robust models could then be built.

To limit the scope of this work, several choices were made regarding the selected methods. This of course also limits the scope of the provided insights. Some of these choices and their implications are discussed in the following.

K-means and other methods commonly use Euclidean distance as the metric of choice for measuring distances between data points. While this is often sufficient, in the context of this work it may have been too simplistic of a model. The main reason for this is the so-called *curse of dimensionality*. This phenomenon has several implications for this work.

When it comes to Euclidean distance, as the number of dimensions grow, the data points start appearing to be more and more sparse or dissimilar to each other in many dimensions, and as such the Euclidean distance starts losing its meaning as a similarity metric. Berthold describes this phenomenon in (Berthold;Borgelt;Höppner;& Klawonn, 2010). If one considers a circle embedded in a square, such that the side of the square d = 2r, where r is the radius of the circle, the corners of the square will not be covered by the circle. The area of the square not covered is around 22 %. If this setting is extended to three dimensions, such that it is now a sphere embedded into a cube, now the volume not covered by the sphere is over 50 %. As the number of the dimensions increases, the volume covered by the (hyper)sphere quickly approaches 0 %. This means that if the data points are evenly distributed over the entire hypercube, then most of the points will reside in the corners, and the probability of a point being within r distance of a reference point becomes very small.

Euclidean distance is also problematic when there are correlated features. Euclidean distance considers distance between individual points only, without regard to how they are distributed. This can lead to a phenomenon where two points are equally far away from a cluster centroid, but one of the points is actually closer to the rest of the points in the cluster than the other one, as shown in Figure 31. Remember that the Self-Organizing Map from chapter 3 showed that there were several highly correlated features. This means that using Euclidean distance to measure anomalies based on distances to cluster centroids does not yield as straightforward a

result as one might expect. A way to mitigate this could be to use alternative metrics such as the Mahalanobis distance instead of Euclidean distance. Mahalanobis distance is the distance between a point and a distribution, and it can be thought of as a multivariate version of Euclidean distance.

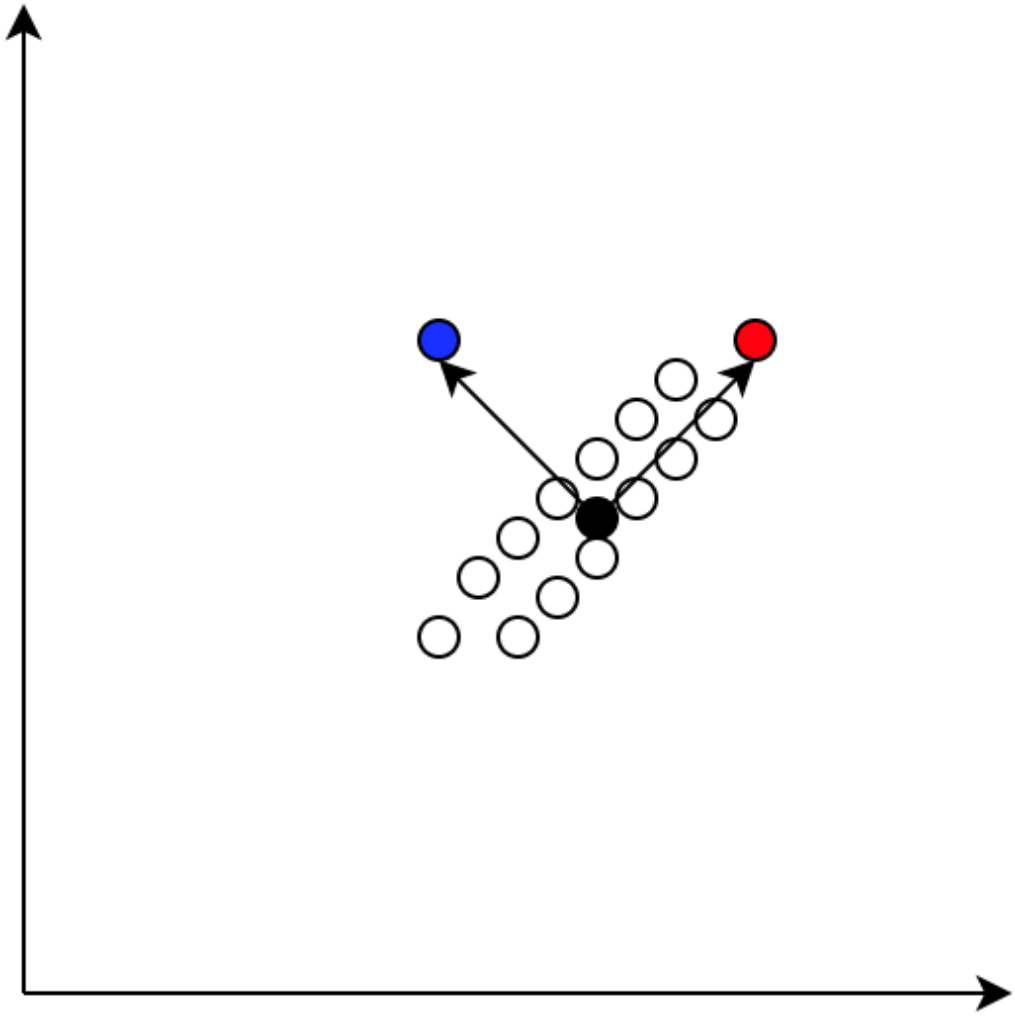

*Figure 31. The red and blue dot are at equal distance from the black centroid, but the red dot is closer to the cluster distribution.*

Another problem caused by the curse of dimensionality is the inclusion of potentially irrelevant features. This problem was partially mitigated by splitting the features into sub models A and B (described in chapter 3); however, sub-model B includes almost all of the features that contain some form of variation. Some feature selection techniques, such as random forests or PCA, could be a way to extract the most informative features into a lower-

dimensional model. Other ways to tackle data sets with too many features include *subspace clustering*, a family of methods that try to automatically identify a set of features that would provide a good clustering result.

Partially related to the curse of dimensionality, one limitation of this work was the choice between the engine variable data and vibration data mentioned in chapter 3. Recall that the engine variable data contained around 50 000 observations of 98 features, while the accelerometer vibration data had 3 primary features (spatial x-, y- and z-dimensions), but almost 9 million observations. It was decided to under-sample the vibration data into the engine variable data set. A possible alternative could have been using interpolation techniques to artificially create samples with the same frequency from the engine data, however a data set of (eventually) 75 dimensions and almost 9 million observations would have been posed a set of new challenges regarding computational complexity. Of course, the vibration data could have been analyzed completely separately from the engine data. However, a 3D visualization of the vibration data shows a somewhat uniformly densely packed blob of data points; it is questionable (but not impossible) whether clustering techniques could have provided meaningful insights from this alone.

The usage of Euclidean distance continues to be a potential source of problems in other tasks as well in this work. A task that remained difficult was choosing the “goodness” metrics for tuning the hyperparameters of K-means and DBSCAN. In the tests on the clustering basic benchmark, cross-validation with silhouette coefficient worked well and gave great results. For the engine data it is however questionable whether silhouette coefficient was a sufficient metric. Especially in the case of DBSCAN it was shown that for this particular data set silhouette coefficient did not play well with the algorithm. For K-means silhouette coefficient gave better results for as low value of k as possible: $k = 2$ gave the best results, $k = 3$ the next best, $k = 4$ next after that and so on. As mentioned in chapter 2, the silhouette coefficient can use any chosen dissimilarity metric of choice, and Euclidean distance was used in this work, and as mentioned in previous examples in this chapter, Euclidean distance is not optimal for high-dimensional data with correlated features. Future tests could include using the Mahalanobis distance as a dissimilarity metric for the silhouette coefficient calculation to see how the results compare to those in this work.

This work has been exploratory in its nature, looking at what happens when different clustering methods and outlier analysis techniques are applied on the engine data set.

When it comes to the research questions and hypotheses posed in the beginning of the thesis, some of them remain unanswered. One such question was the hypothesis that different states of the running engine, such as warming up, running at full capacity, slowing down, etc., may translate into distinct clusters in the data set. While there seems to be some indication that the data presents clustered behavior, it is not completely clear whether these directly translate into the engine states, and it is even less clear how many such states or clusters would be the appropriate amount. Results from the SOM and K-means silhouette coefficient optimization suggest that there is a low number of clusters (2 to 5), however the eigengap heuristic and DBSCAN clustering suggested the number of clusters to be somewhere in the range of 17 to 28. Intuition suggests that the lower estimates given by the SOM and K-means optimization are probably more reasonable guesses, a sort of "big picture" of what states the engine takes, whereas the higher numbers of clusters suggested by the eigengap heuristic and DBSCAN may indicate something else.

Another research question was: Do the different methods flag the same data as anomalous? This question did get an answer: the different algorithms flagged different parts of the data as anomalous, however displaying certain interesting patterns such as flagging anomalies in similar parts of the data for K-means and DBSCAN. It became clear that the algorithms are highly sensitive to initialization settings and choices of metrics.

In this work each clustering algorithm was paired with a single outlier analysis method: DBSCAN by itself, K-means with distance to cluster centroids, and spectral clustering with a reduced kernel density estimation. For more robust results, further work could include cross-checking results from K-means and spectral clustering with each other's outlier analysis methods, i.e., pairing K-means with the KDE and spectral clustering with the distance to cluster centroids method. Dissimilarity metrics other than Euclidean distance should definitely be tried; the already mentioned Mahalanobis distance seems like a solid candidate. Should it give good results, one could experiment with building some sort of clustering ensemble model, which could take the results of each of these methods into account and tag certain areas of the data space as high risk for containing anomalies.

## References


Aggarwal, C. C. (2017). *Outlier Analysis, Second Edition.* Springer.

Anamika, G., Mayuri, K., Reddy, B. K., Iyengar, N. C., & Caytiles, R. D. (2018). Analyzing the performance of Various Fraud Detection Techniques. *International Journal of Security and Its Applications*, 21-36.

Berthold, M. R., Borgelt, C., Höppner, F., & Klawonn, F. (2010). *Guide to Intelligent Data Analysis.* Springer.

Botia, J. A., Vandrovcova, J., Forabosco, P., Guelfi, S., D'Sa, K., Hardy, J., . . . Weale, M. E. (2017). An additional k-means clustering step improves the biological features of WGCNA gene co-expression networks. *BMC Systems Biology*.

Chandola, V., Banerjee, A., & Kumar, V. (2007). *Anomaly Detection: A Survey.* Minneapolis: University of Minnesota.

Comitani, F. (2022, oct). *fcomitani/simpsom: v2.0.2.* Retrieved from https://doi.org/10.5281/zenodo.7187332

Ester, M., Kriegel, H.-P., Sander, J., & Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. In E. Simoudis, J. Han, & U. Fayyad, *Proceedings of The Second International Conference on Knowledge Discovery and Data Mining.* Association for the Advancement of Artificial Intelligence (AAAI).

Fränti, P., & Sieranoja, S. (2018). K-means properties on six clustering benchmark datasets. *Applied Intelligence*, 4743-4759.

Fränti, P., & Sieranoja, S. (2018). *K-means properties on six clustering benchmark datasets*. Retrieved from http://cs.uef.fi/sipu/datasets/

Gunay, H. B., & Shi, Z. (2020). Cluster analysis-based anomaly detection in building automation systems. *Energy & Buildings*.

Kekäläinen, S. (2021, June 15). Data Pipelines and Edge Analysis in Engine Installations. Vasa.

Kohonen, T. (1990). The Self-Organizing Map. *Proceedings of the IEEE, Vol 78, No. 9*.

Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern Recognition*, 451-461.

Linderman, G. C., & Steinerberger, S. (2019). Dimensionality Reduction via Dynamical Systems: The case of t-SNE. *SIAM Journal on Mathematics of Data Science*, 313-332.

Mtesigwa, M. M. (2020). Vibration Sensing for Engine Condition Monitoring and Predictive Maintenance. Finland: Åbo Akademi University.

Pedgregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., . . . Duchesnay, E. (2011). Scikit-learn: Machine Learning in {P}ython. *Journal of Machine Learning Research*, 2825--2830.

Rousseeuw, P. J. (1987). Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 53-65.

Schubert, E., Sander, J., Ester, M., Kriegel, H.-P., & Xu, X. (2017). DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN. *ACM Trans. Database Syst. 42, 3, Article 19*, 21.

Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding Machine Learning: From Theory to Algorithms.* Cambridge University Press.

Shao, X., Lee, H., Liu, Y., & Shen, B. (2017). Automatic K Selection Method for the K-means Algorithm. *The 2017 4th International Conference on Systems and Informatics (ICSAI 2017).* IEEE.

Siganos, A. (2019, September). Anomaly detection on vibration data. Eindhoven.

Valko, M., Kveton, B., Valizadegan, H., Cooper, G. F., & Hauskrecht, M. (n.d.). *Conditional Anomaly Detection with Soft Harmonic Functions.*

von Luxburg, U. (2007). A tutorial on spectral clustering. *Stat Comput*, 395-416.

# Appendices

## Appendix 1: List of symbols and terms

- Embedding: an instance of a mathematical structure contained within some other instance. As examples, the natural numbers can be thought of as an embedding in the integers, and a 2D Euclidean space as an embedding in some higher-dimensional Euclidean space.
- Euclidean distance: for two n-dimensional points, $(p_1, p_2, \dots p_n)$ and $(q_1, q_2, \dots q_n)$ the Euclidean distance is $\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2}$
- i.i.d.: independent and identically distributed
- Log-likelihood: a transformation which takes the logarithm of a likelihood, which transforms calculations from a product to a sum. The result is equivalent but has a more convenient representation.
- Vector 2-norm: Also known as Euclidean norm, it is the "magnitude" of a vector in Euclidean space, i.e. the length of the vector.
- $x_i$: element *x* with index number *i* in some collection, e.g. in a set
- $\{x_1, x_2, \dots x_n\}$: a set with *n* elements
- $x \in R^d$: *x* is in *d*-dimensional Euclidean space. In terms of this thesis, the element *x* is a vector of *d* real values
- $\sum_{i=0}^{n} x_i$: sum of elements $x_0$ through $x_n$ from some collection
- $\subset$: subset of (in the context of this thesis the meaning is similar to $\in$)