JOHANNES KEPLER
UNIVERSITY LINZ

Submitted by
Dipl.-Ing.
Vlad-Ilarie Precup, BSc.

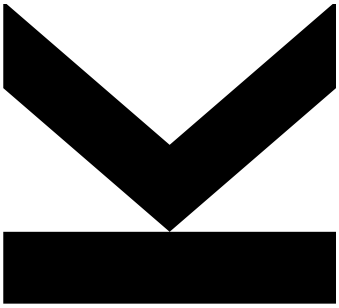
Submitted at
Institute for
Software System
Engineering

Supervisor
Prof. Dr.
Alexander Egyed,
Head of Institute

Co-Supervisor
Dipl.-Ing.
Hannes Thaller, BSc.

July 2018

Cluster Analysis for Multivariate Application Performance Management Issues



Master Thesis

to obtain the academic degree of

Master of Science

in the Master's Program

Internationaler Universitätslehrgang
Informatics: Engineering & Management

JOHANNES KEPLER
UNIVERSITY LINZ
Altenbergerstraße 69
4040 Linz, Österreich
www.jku.at
DVR 0093696

Acknowledgements

First of all, I would like to express my gratitude to Prof. Dr. Alexander Egyed and Dipl.-Ing. Hannes Thaller, for their continuous support, guidance and confidence in me for reaching the goals of the Master thesis in the given time frame. At the same time, I am extremely grateful to Dr. Herwig Moser for his guidance, dedication and professionalism in representing the company's objective. Furthermore, their mentorship and continuous feedback have been crucial to the delivery and success of the present project.

I would also like to thank Dr. Wolfgang Beer and Dipl.-Ing. Alexander Scheran for their friendliness and support as well as for enabling the collaboration framework between Dynatrace, and me as their collaborator.

My sincere appreciation and gratitude also goes to Prof. Dr.phil. Dr.h.c.mult. Bruno Buchberger for his effort of making everything happen. His dedication for the *Universal Computing and Business* Master program and his energy motivated me throughout the entire study period.

Last but not least, I am grateful for having the chance to meet and exchange ideas with Dr. Otmar Ertl and Hans Kohlreiter, BSc. Their advices have been very important to me and the insightful conversations that we had, sparked ideas that drove the project forward in a very smooth manner.

Overall, Dynatrace has been an ideal place for innovation and experimentation. The office climate has been pleasant and friendly at all times, so I am grateful for this wonderful experience. Thank you, Dynatrace, for having had me as part of your team!

The research reported in this monograph has been partly supported by the Austrian Ministry for Transport, Innovation and Technology, the Federal Ministry of Science, Research and Economy, and the Province of Upper Austria in the frame of the COMET center SCCH.

Abstract

With the growth in size and complexity of the software systems, their monitoring becomes more difficult to control. Generally, maintaining complex software systems imply complex reporting of the identified problems within them. However, this can also cause a more-than-necessary amount of detected problems. In this regard, a balance between frequent monitoring problem notification and coarse reporting is essential for running software systems effectively and efficiently. This thesis tackles the business problem of reducing the recurrent monitoring issue overload for users by systematically using unsupervised techniques to analyze and cluster previously-identified monitoring issues. We firstly propose a data selection and pre-processing method to prepare the monitoring data for clustering. Then we choose the most suitable clustering algorithm for the problem based on our comparison experiment and develop a lightly-customized version of the Wave-Hedges distance metric function to compute the distance matrix between the problem samples. We show the ground truth for a restricted data set and present the results of the parameter optimization technique based on external evaluation metrics. After evaluating the performance of the clustering model based on the obtained parameters, we show that our method can generalize for vast amounts of data and we conclude by comparing our method to the intuitive approach in the context of the business conditions imposed by our problem.

Contents

1. Introduction

1.1. Problem Statement

Complex software requires extensive monitoring to ensure that it runs in normal parameters. Some organizations choose to develop their own monitoring mechanisms using existing libraries and development tools. But since this requires an enormous effort and significant expertise from the development team, many organizations choose to use professional solutions. One such solution is offered by Dynatrace.

Dynatrace offers a suite of tools that target these specific scenarios. The tools come in various packaging solutions such as Software-as-a-Service (SaaS) and can be easily integrated with existing software deployed in public or private clouds, on mainframes or even on private servers. Once the service is integrated with the deployed software system, it firstly discovers its topology on all its levels of abstraction: from the underlying infrastructure such as the Virtual Machines to the business applications or services. Its high degree of integration enables the Dynatrace service to monitor a wide variety of components, ranging from cloud platforms and infrastructure components to web applications and their business-specific web pages.

Each of the detected components have certain parameters: a state[1] and various relationships or communication patterns[2] with other components in the topology[3]. All these parameters are closely monitored and analyzed for anomalous behavior. Detected anomalies are captured and reported back to the system as monitoring events, which are consequently correlated using an advanced proprietary mechanism to form monitoring problems ([3]). Data represented at this level of abstraction is the main focus subject for this thesis.

[1] Running environment parameters such as processor or memory usage for virtual machine hosts

[2] As we will see in the next chapter, there are two types of relationships: infrastructure-related and business logic-related

[3] The deployment architecture of the monitored software system

Monitoring problems emerge when one or more components in the distributed system are affected by extrinsic factors such as high load on servers or by intrinsic factors, such as hardware or software malfunctions. All the events which form a problem have the same conceptual root-cause and indicate a real user impact. From the reported data point of view, they can be viewed as a collection of events accompanied by various pieces of metadata.

Similar to process industry [38], it is easy to understand that the growth in software systems complexity – both in terms of business logic and deployment model – correlates with a growth in the amount of detected anomalies or alarms. The more anomalies are reported, the less relevant they can become for the system's supervisor, as their amount becomes overwhelming and nearly impossible to deal with. Hence, instead of reporting monitoring alarms individually, a more compact, less verbose, yet comprehensive and insightful reporting method is needed. One such potential method aggregates the data into categories of alarms (or problems) which show significant similarity and presents the categories of alarms rather than individual alarms to the supervisor. This, in turn, raises the problem of finding the similarity criteria and grouping the alarms into the afore-mentioned categories.

1.2. State of the Art

From a data point of view, as we will see in the subsequent chapters, grouping or classifying objects is strongly related – or rather equivalent – to cluster analysis. As described by Han, Kamber, and Pei [23], cluster analysis (also known as clustering) is the process of finding structure in unstructured data.

The problem of noise reduction by cluster analysis has been tackled in various manners by scientific communities in diverse fields, from distributed systems monitoring to industrial processes or general process models.

The closest field to the current thesis is undoubtedly the one of distributed systems monitoring. In the past decade, from the emergence of *Service-Oriented Architectures (SOA)* [44] and later *Microservices* [32] in software engineering, the deployment footprint has exploded and *cloud computing* has been acquiring an increasing portion of the market share. As a consequence, cloud monitoring and anomaly detection play a significant role in the overall healh of software systems deployed in public or private clouds. Monitoring oftentimes involves significant computational resources over time, however they are easily disregarded. In their recent work, Wang et al. [49] propose a proactive approach to automatically adjusting monitoring data verbosity. This is achieved by firstly correlating metrics and

disregarding redundant metrics, then detecting anomalies based on the deviations given by the metrics' Cosine similarities. Based on this data, a reliability model is built, which helps adjust the metrics collection rate based on the likelihood of fault occurrence. Metrics represent one aspect of monitoring. Another aspect, perhaps even more significant, is represented by events. Quiroz et al. [37] present a relative density-based clustering analysis approach for the events of self-monitoring distributed systems. In the context of this research, events are represented by points in space. To cluster these points, a bottom-up approach is proposed, where the problem space is divided into regions and clusters are identified in each region in a decentralized manner.

The immediately next area of interest, which is closely related to distributed systems monitoring, is diagnostics event log processing. In their article, *A Data Clustering Algorithm for Mining Patterns From Event Logs*, Vaarandi [47] proposes a density-based clustering approach to analyze and group similar log entries. The algorithm iterates over the data in three different steps: firstly it identifies the so-called *1-regions* consisting of data samples that share similar values for just one attribute. Then it builds the cluster candidates by identifiying how many regions a certain entry belongs to. Finally, the regions having a large enough support value are promoted to clusters.

In a slightly different form, event logs can also be used to trace real-life events (and especially events originating from medical systems). Based on this type of logs, process mining is able to extract process models. The downside, however, is that they tend to be unstructured and hard to comprehend. To alleviate this issue, Jagadeesh et al. [30] propose a context-aware approach to (event) trace clustering based on a modified generic edit distance and evaluate the outcome using established metrics in the context of process mining. The edit distance, also known as the Levenshtein distance, computes the number of edit operations needed to transform one sequence into the other. Finally, the authors compare the clustering outcome using various distance measures based on various process mining indices. In another research paper, Song, Günther, and Aalst [46] view this problem from a slightly different angle (and with a different practical example): traces are transformed to points in the n-dimensional space. Next, well-known distance measures and various clustering techniques are used to cluster the traces.

From process models we turn to process industries. In process industries, alarm floods have become a stringent problem with the factory automation. Industrial machines are constantly monitored to ensure they run under normal conditions, also known as nominal parameters. However, depending on the fine-tuning of the

systems, they might under-react, in which case the consequences can be disastrous, or over-react, in which case they send alarms at a rate that is overwhelming for the human workers. The latter cases are called *alarm floods* [2]. In this paper, the authors propose an unsupervised method for grouping alarm floods by first using the Jaccard distance [29] to find dissimilarities between alarm sequences, then clustering them using the Agglomerative Hierarchical Clustering technique [45]. In another approach, *Improved correlation analysis and visualization of industrial alarm data* [52], which relies merely on a binary alarm data moded, the Gaussian kernel [5] is applied to the binary data over time and and the resulting Gaussian functions are superimposed. The obtained continuous functions are then compared with each other using Pearson correlation coefficient [34]. The most systematic approach is used by Rodrigo et al. [38], who combine alarm logs, process data and connectivity / topological model analysis to not only group similar alarms in an alarm flood, but also identify the causal alarm. The most relevant part of the method to the present thesis is the actual classification. After a careful removal of irrelevant chattering alarms and the generation of alarm sequences, the authors propose the use of a modified Smith-Waterman-based algorithm [13], a dynamic programming approach used mainly in molecular sequence analysis, to build a similarity matrix before feeding the data to the clustering algorithm. Since the algorithm is computationally expensive, the authors optimize the similarity computation step by first determining if the sequences have enough alarms in common. A common denominator between the last three approaches presented above is the use of the Agglomerative Hierarchical Clustering to find groups of similar alarms or alarm sequences.

1.3. Solution Overview

This thesis tackles various aspects of grouping monitoring problems. We start by analyzing the business scenarios and identifying the significant attributes which wil compose the problems to be clustered. Based on the type of these attributes and the problem space, we examine various distance measures and various clustering techniques. Finally we evaluate the outcomes of these techniques and perform various tests based on data originating from the Dynatrace system. The novel contributions brought by this thesis are:

- The selection and pre-processing of the data to be clustered.
- The self-normalizing distance function, that is tailored to the domain and the type of data to be clustered.
- The methodology used for parameter estimation and clustering model evaluation.

2. Business Context

The research conducted as part of this thesis is the outcome of an industry project in collaboration with Dynatrace, one of the most important APM solution R&D company. As the stakeholder for this project is Dynatrace, the context of the research was provided by the company and thus, the project aims at solving a very specific business problem.

2.1. Problem Context

The Topology. Each software system has multiple hardware and software components. Generally, there are two methods of discovering the topology: either through a discovery service or by deploying agents / injecting instrumentation code on the underlying infrastructure. Since the former method cannot be used in all cases as not all the software deployments ensure the existence of a discovery service, Dynatrace integrates with the systems using the latter approach. Once integrated, Dynatrace is able to identify and classify all these components, which we will further call **entities**. Subsequently, it builds a directed graph with the nodes being represented by entities and the edges – by the relationships between them. According to the official product documentation [18], there are two types of relationships between entities:

1. The **Business-logic** relationships belong to the "horizontal path" which transactions follow, as illustrated in figures 2.1 and 2.3. For example, a transaction is initiated by a user action and can follow through various applications and services. The capability of Dynatrace to identify and correlate this type of relationships and monitor the transactions flowing through them is called the *PurePath Technology* [18].

2. The **Infrastructure** relationships between different levels of abstraction of the previously-mentioned components: from the service itself relying on a certain technology, for instance *NodeJS*, through the process hosting this service, all the way to the (virtual) machine where the process runs (and eventually the hypervisor hosting the VM). This vertical view of the infrastructure that Dynatrace conceives is called the *PureStack Technology* [18].

These relationships between various entity types are depicted in figure 2.1. As previously stated, discovering these relationships is possible due to the deployed agents. In addition to this, there is another functionality performed by the agents: runtime data collection and instrumentation of the components where they are deployed. They are responsible for collecting and sending data to the Dynatrace infrastructure for further processing. Dynatrace uses heuristics and statistical models to obtain higher-level information from the data sent by the agents. A significant part of this information is represented by events and problems.

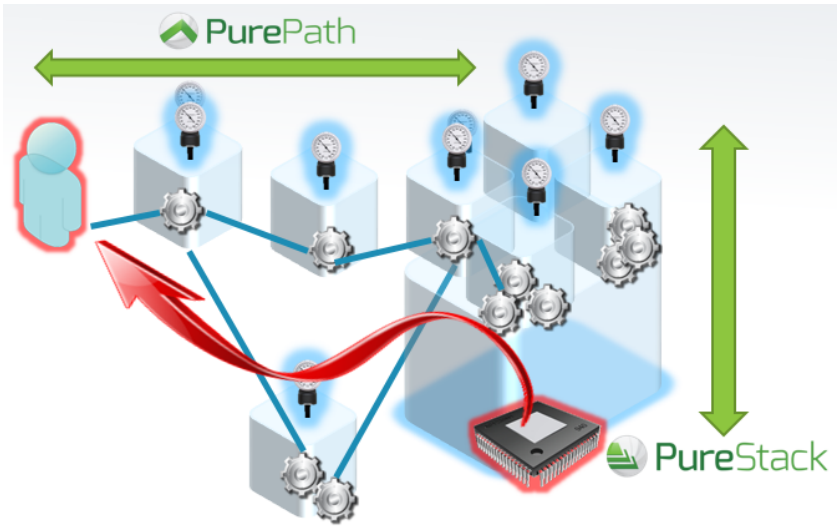


Figure 2.1.: The PurePath and the PureStack. *Source:* Dynatrace [18].

Events and Problems. In the Dynatrace-specific terminology, events can be considered as the building blocks of problems. Each problem in the Dynatrace data model is a subgraph of the infrastructure topology graph presented above, since it represents the events, or malfunctions of one or more entities belonging to the topology. Nodes in problem graphs have a somewhat different interpretation than the ones in topology graphs. They represent events which occur at the level of certain entities from the topology. Edges, however, keep the same interpretation, so they represent the relationships between the entities at the level of whom events occur. The event correlation mechanism as well as the relationship establishment strategies are described in extensive detail in the patent issued by Ambichl et al. [3].

The Problems Feed is an intuitive view of the problems in a system. It provides search functionality as well as various levels of filtering. As shown in figure 2.2,

both open and closed problems can be explored. In contrast to problems which occurred in the past (thus they appear as 'closed'), 'open' problems still persist in the system at the time of view. This can be observed on the barchart in the top side of the figure. Below this barchart, the main view shows cards containing various details about individual problems such as duration and impacted components / entities. As software systems grow in complexity, the number of reported problems can in turn grow drastically due to the numerous relationships which potentially cause anomalous behavior. As a consequence, the problems feed becomes overly cluttered on one hand, and it loses the strength of its objective, that is to show the system administrator an overview upon the anomalous behavior of the system as a whole.

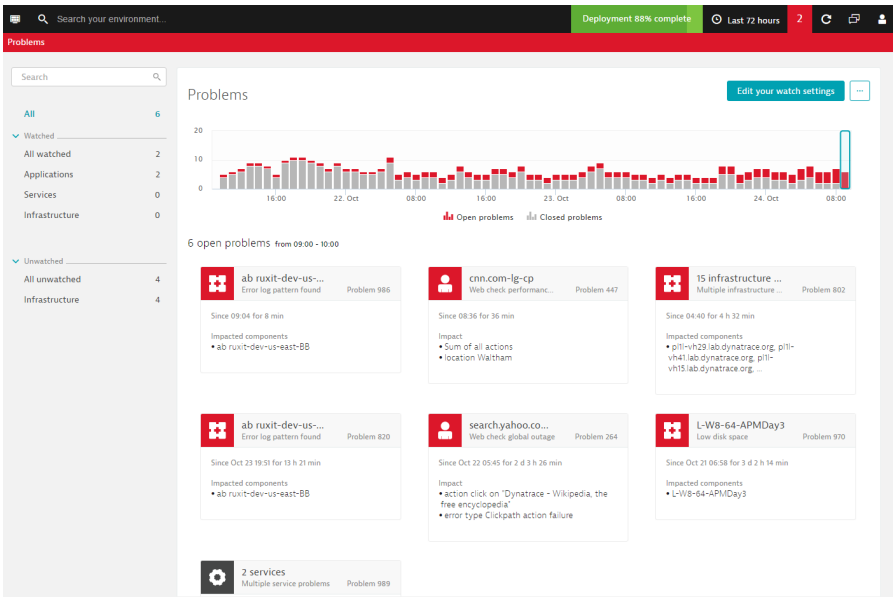


Figure 2.2.: The problems feed screen capture from the Dynatrace client. *Source:* Dynatrace [17].

Problem Example. For a complete understanding of the concepts explained above, let us introduce a simple example, depicted in figure 2.3. By looking at the system depicted by the topology, we notice how the distributed application is deployed. There are two high-level components: **A (web) application** deployed on an *Nginx* web server cluster ran in a *Linux* environment, backed by a collection of **microservices** running on various technologies (*Apache* or *Node JS*) in various environments (*Linux* or *Windows*-hosted *Docker containers*). The figure also illustrates both the *PurePath* and *PureStack* proprietary technologies and how they are used for event correlation purposes. The arrows represent the relationships

between components in the system. An example of infrastructure/*PureStack* relationship is the "*runs on*" relationship between the web application and the *Nginx* web server cluster. In addition to this, a business-logic/*PurePath* relationship is the "*calls*" relationship between the application itself and the various microservices. With the aid of these relationships, the entire system topology is discovered and the events – correlated. The figure intuitively illustrates the entities impacted by the problem as colored in red together with exclamation marks next to them. Just by analyzing them, one can draw the conclusion that one of the *Linux* hosts supporting the microservices cluster is in a bad state or presents anomalous behavior. This causes the microservices cluster to malfunction, which in turn causes the deployed microservices to present abnormal behavior. Thus, the application encounters issues each time it calls any of the microservices. This behavior is detected by Dynatrace and converted into events. All the events are then correlated into a monitoring problem, which is then reported to the end-user/system administrator.

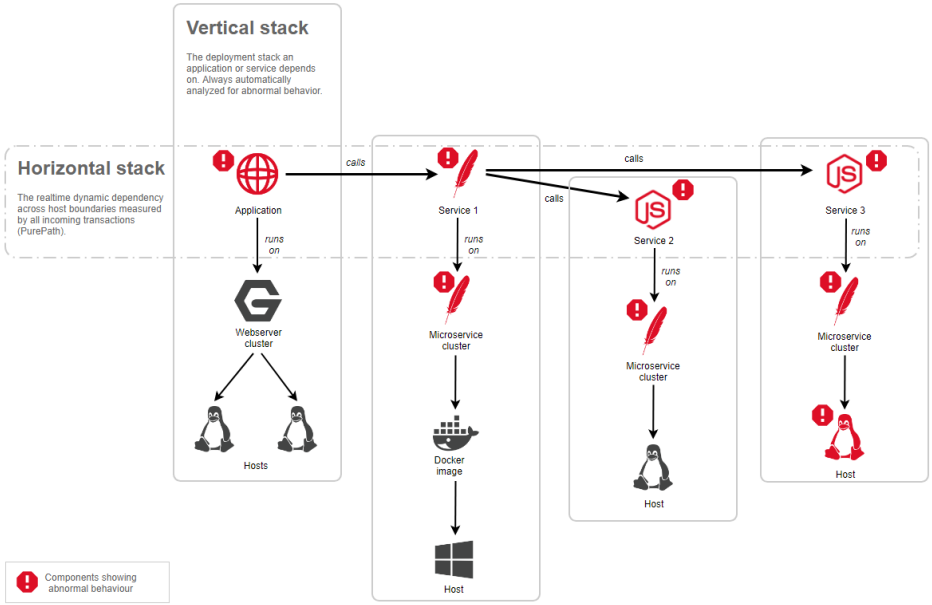


Figure 2.3.: Event correlation graph in the context of a monitoring problem example: graph nodes are represented by components showing abnormal behavior, emphasized by the presence of the exclamation marks. *Source:* Dynatrace [17].

2.2. Business and Research Goals

In the previous section we presented the role of the problems feed as well as how it can be undermined by vast amounts of anomalies. The consequences of this clutter are twofold: on one hand an overview of the problems in the system cannot be easily observed, and on the other hand the overwhelming amount of problems cannot be prioritized and acted upon in a timely manner. In process industry, this phenomenon is called *alarm flooding*.

One particular property of alarm floods is that they are caused by many alerts occurring in a short amount of time. The data that we deal in this business case however is on a superior level of abstraction. Dynatrace monitoring problems are already a result of extensive data and event aggregation and correlation over time[1]. Nevertheless, problems still can and do recur. This is what makes the research for finding a solution to this problem so interesting and challenging.

By relying on the property described above, our aim is to reduce the amount of reported information for the purpose of significantly alleviating the consequences of alarm / problem floods. This implies providing **a solution for this problem, which can be used for grouping the similar / recurring / redundant problems into clusters**.

The user stories which will rely on this solution are:

- The user (system administrator) is able to retrieve problems which are similar to a certain problem selected from the problems feed.
- The user (system administrator) is able to change the view mode of the problems feed for their tenant from displaying the list of problems to displaying the list of groups of similar problems.
- The user (system administrator) can choose to opt out from notifications regarding new problems similar to a certain selected problem.

Having identified the business scenarios for solving this problem, the main research goal is to provide an efficient customized clustering technique for the application performance issues represented by this data. This end goal can be broken down into further sub-goals:

[1] In fact, they are network-like structured events which occur in the context of various system components

1. Collaboration with the Dynatrace engineering team in order to gather information about relevant attributes of problems which can later be used as inputs for clustering.

2. Feature selection of monitoring problems exposed by the Dynatrace systems.

3. Preparing the data for cluster analysis (i.e. event properties embedding, other preprocessing).

4. Research of the state of the art in terms of distance metrics and choice / elaboration of a suitable metric for distance computation between problems.

5. Research of the state of the art in terms of clustering methods and choice of a suitable clustering algorithm.

6. Evaluation of the performance and accuracy of the devised method. This includes ensuring that the user stories which rely on it described in the 2.2 section can be successfully implemented and there is no user impact in terms of responsiveness or false or inaccurate information provided.

3. Theoretical Background

This chapter introduces the theoretical concepts used in the thesis. We start by defining the business domain terminology. Next, we introduce the theoretical knowledge used for elaborating the solution proposed by the thesis, starting with basic data mining concepts and ending with more advanced unsupervised learning concepts.

3.1. Application Performance Management

Application Performance Management or APM solutions are, according to the Dynatrace official website [16], complex software systems whose purpose is to monitor runtime information and user transactions and report potential issues, as well as manage the performance and availability of external software systems in an intuitive and least-intrusive way for its users. Modern APM solutions have a proactive behavior rather than a reactive one. Thus, they are capable of discovering performance issues before system users are (heavily) impacted. This is a key aspect for software service providers whith offerings under SLAs, as APM solutions help them to ensure high availability for their services.

Monitoring Events represent operational patterns of components in the monitored system which are unusual in a sense that affect the runtime of the system. In a nutshell, they represent anomalous behaviors of components. For instance, monitoring events can indicate the excess of memory usage in a certain virtual machine, or unusually high latency of connections to a database service. Hence, they are always characterized by an entity (type) – *where* the event takes place – and a type – *what* exactly happens at the level of that entity.

Monitoring Problems Being the object of this thesis, monitoring problems represent the high-level view of the issues discovered by the Dynatrace APM suite. They are the result of coordinated event collection from all the distributed deployment components of the monitored software system.

Alarm Flooding or in the case of APM, notification overload, is encountered when the monitored system is impacted by multiple recurrent problems which are

similar in certain respects. As a consequence, the APM suite repeatedly (and redundantly) notifies the system administrators about these problems. In many cases, the effectiveness and credibility of the system suffer as the system administrators can ignore, or on the contrary, dedicate effort on investigating and eventually addressing a problem that had been addressed before.

3.2. Data Mining

The problem addressed by this thesis is in the field of data mining. Sometimes interchangeably used for Knowledge Discovery from Data (KDD), the concept of data mining refers to the process of discovering patterns in a large amount of data [12]. In their book, Han, Kamber, and Pei [23] describe the process of data mining as being "automated or convenient" and involve processing data from "large databases, data warehouses, the Web, other massive information repositories, or data streams." As described in the formerly-mentioned curriculum proposal, data mining is often used in conjunction with other domains such as machine learning. Later in this section we are going to address the latter concept in further detail.

3.2.1. Data Collection and Integration

Data collection and integration is the step performed through which data is collected (i.e. queried from databases, requested from service endpoints etc.) and eventually integrated, if multiple data sources are used.

3.2.2. Data Preprocessing

Also known as data preparation, data preprocessing is the phase of a data mining process where data is prepared for modelling. As oftentimes data in its raw format is incompatible with modeling algorithms, through this phase it is selected and integrated from various sources, its quality is ensured and ultimately it is formatted and transformed into a form which is appropriate for processing using data mining / machine learning algorithms.

Feature Extraction is one of the data preprocessing steps through which data is transformed such that non-numeric representations of data such as text and images are transformed into numerically represented features supported by the various data mining and machine learning algorithms.

The "Curse of Dimensionality". As Keogh and Mueen [31] synthesize in their contribution, the curse of dimensionality describes the exponential increase

in terms of complexity caused by additional dimensions to data, as initially stated by Bellman [7] in 1957. The greater the number of features is in a data model, the more complex it is for data mining and machine learning algorithms to process it. Furthermore, the visualization or graphical representation of such a complex data model becomes close to impossibe to conceive (and imagine). In fact, a data set with more than 3 dimensions becomes exponentially unintuitive and infeasible with each added dimension.

The Bag of Words Representation is a data format where sequences of words (also known as documents) are represented as a numeric matrix. This is often called the vector space model (or VSM) [15] and the process through which it is obtained contains the following operations:

- **Tokenizing** the text documents by using certain characters as token separators.
- **Counting** the occurrences of the previously identified tokens in each document.
- Optionally **normalizing** and **weighing** tokens based on certain rules.

The result is a sparse two-dimensional matrix with one row per document and one column per identified token.

The most important VSM transformation technique is the term frequency model which implements tokenization and occurrence counting. As a result, it produces a two-dimensional array which contains histograms of tokens for each document.

The *term frequency – inverse document frequency* (TF-IDF) model also normalizes the *term frequency* computed by the term frequency model, based on the following formula:

$$\text{tf-idf}(t, d) = \text{tf}(t, d) \times \text{idf}(t, d),$$

where the count vectorization is performed by $\text{tf}(t, d)$ and the inverse document-frequency is computed by

$$\text{idf}(t, d) = \log \frac{n_d}{1 + \text{df}(t, d)} + 1.$$

In the equation above, n_d represents the total number of documents and $\text{df}(t, d)$ yields the number of documents that contain token t. Once the tf-idf is computed for each token in each document, each row is then normalized by the Euclidean norm.

Normalization or scaling is a data transformation technique used to scale it in order to make it more comparable on one hand and to help remove the bias introduced by large range differences of the attributes on the other hand. This bias is perceived by data mining algorithms, which generally use absolute differences for comparing objects. Hence, attributes with large ranges have a greater impact than attributes with lower ranges. There are various normalization methods, among which we remind:

- **Min-max normalization**, which scales the data attributes within a specified range.
- **Z-score normalization** (also known as **mean-variance** or **standard scaling**), which removes the attribute mean and scales the values to the unit variance.
- **Robust scaling**, which scales data based on statistics which are resilient to outliers. In this case Z-score normalization does not yield favorable results.

Modeling is the most important phase of a data mining project, as it revolves around choosing the approach or the algorithm, then training and evaluating the model.

3.3. Unsupervised Machine Learning

Machine Learning is, as the term suggests, the task of automated learning based on digital data. With applications in fields like biology, medicine or autonomous driving to name a few, machine learning is currently one of the trending research topics in the fields of engineering and artificial intelligence.

Unsupervised Learning is the branch of machine learning which focuses on learning from unstructured data, or rather understanding and finding structure in this type of data, visualizing or compressing it, unlike supervised learning, which is used for predicting future data based on previous knowledge, according to Prof. Hochreiter [24].

3.3.1. Cluster Analysis and Clustering Algorithms

Also known as clustering, cluster analysis is the task of finding structure in unstructured data, or grouping a set of objects based on their similarities and/or distances between them: the objects within a cluster have high similarity, but have high dissimilarity to objects in other clusters [1]. As for cluster analysis there

is no one-solution-fits-all approach – a circumstance also known under the name of the *No Free Lunch Theorem* [50] – the appropriate algorithm as well as its parameters (including the distance measure) highly depend on the data to be clustered. According to Estivill-Castro [20], who studied the vast domain of cluster analysis, there are multiple types of clustering techniques, among which we remind:

- **Centroid models**, which represent each cluster by a centroid, either virtual or among one of the existing items in the data set. The most representative and widely known centroid-based clustering technique is the *k-means* algorithm.
- **Connectivity models**, which are based on distance connectivity. *Hierarchical clustering* methods are the most representative, with *BIRCH* being one of the methods suitable for high data volumes.
- **Message-passing models**, which iteratively form clusters using the information obtained by passing messages between the data samples. *Affinity propagation* is one of the most representative message passing clustering algorithm.
- **Density models**, which define clusters as uniformly dense regions of samples. *DBSCAN* is the most popular density-based clustering method.

Mean Shift is a clustering approach which, like k-means, characterizes the identified clusters as centroids. What makes it special though, is that it uses mean shift vectors for centroids, which point to the maximum increase in the density of samples. For computing this density, it requires repeated computation of nearest neighbors for samples. This is an iterative method, and for each iteration it performs one step towards the centroids shown by the mean shift vectors. The algorith stops when the changes in centroids are below a certain threshold.

BIRCH or Balanced Iterative Reducing and Clustering using Hierarchies [53], is one of the more recent hierarchical clustering techniques, whose main goal is to process a high volume of numerical data by combining classical hierarchical clustering with iterative partitioning of data based on a Euclidean distance metric and using a tree data structure – the *Characteristic Feature Tree*. This empowers the algorithm to produce clusters with only one scan of the data set, making it highly scalable for large amounts of samples.

Affinity Propagation is a similarity-based clustering, where "affinity"-related information is passed between objects through a message sending mechanism. The affinity is usually chosen to be the negative Euclidean distance. Although affinity

propagation finds "exemplars" or centroids, it does not require the number of clusters as a parameter [21]. The data which are iteratively passed between samples are the *responsibilities* (a quantification of the evidence that a sample serves as an exemplar for the other) and the *availabilities* (a quantification of the evidence that a certain sample is indeed an exemplar).

DBSCAN has been so far the most widely-used density-based clustering algorithm. Although initially proposed over two decades ago by Ester et al. [19], due to its simplicity but at the same time, performance, power and versatility, DBSCAN has been researched and discussed in multiple papers among which we remind *DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation* [22] and *DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN* [42]. The principle behind DBSCAN is simple and intuitive: the *density* of a cluster is defined by the *minimum number of samples* which need to we within a predefined *epsilon* distance from the current sample in order for this sample to become a core sample and thus belong to or create a new cluster. The repeated transitivity of this concept to other sets of points is named *density-reachability*. It follows that any point a is *density-connected* to point b if there exists a core point o such that both a and b are density-reachable from o. An advantage of DBSCAN over k-means is that it is capable to detect noise or samples which should not belong to any cluster, according to Schubert et al. [42].

3.3.2. Distance Metrics as Parameters for Data Clustering

Distance or dissimilarity metrics are undoubtedly the most influential parameter of most of the clustering algorithms in terms of shape and content of the identified clusters. They quantify the concept of distance between samples, as an inverse of the similarity. As a matter of fact, one could say that

$$similarity(a, b) = 1 - distance(a, b). \tag{3.1}$$

There is a plethora of distance metrics, both symmetric and asymmetric. Based on the data set and the problem contstraints, the distance measure should be carefully selected. This, however, is not an easy task and extensive experimentation might be needed to find a suitable candidate. Among the existing distance functions, we remind the *Euclidean* (3.2), *Minkowski* (3.3) or *Wave Hedges* (3.4) distance functions, however custom distance measures can also be conceived for clustering tasks.

$$d_{Euc}(a, b) = \sqrt{\sum_{i=1}^{n} (b_i - a_i)^2} \tag{3.2}$$

$$d_{Mk(p)}(a,b) = \sqrt[p]{\sum_{i=1}^{d} |b_i - a_i|^p} \tag{3.3}$$

$$d_{WH}(a,b) = \sum_{i=1}^{n} (1 - \frac{min(a_i, b_i)}{max(a_i, b_i)}) \tag{3.4}$$

An extensive list as well as a comprehensive survey on distance functions can be found in the work carried by Cha [11].

3.3.3. Parameter Optimization of the Clustering Algorithms

The next important aspect after defining or choosing the distance function is the estimation / optimization of the other clustering parameters. When no ground truth is known, the estimation procedure is specific to each clustering algorithm and can vary significantly. For example, in the case of *k-means* clustering, determining the *n_clusters* (number of clusters) parameter, line search is recommended to be used: for each value assigned to *n_clusters*, an internal evaluation metric of the clusters is computed. We mention some of the internal evaluation metrics in subsection 3.3.4. After the process is completed, the parameter value associated to the best evaluation result is then used for clustering.

For DBSCAN specifically, choosing the parameters is performed through a series of steps. First, the `min_pts` parameter is determined. In a subsequent paper Sander et al. [41] suggest setting the `min_pts` parameter to twice the number of dimensions of the data set. The remaining parameter, `eps` (ϵ), can be approximated in a data-driven way, using the "elbow" / "knee" / "valley" method, described in the initial article and reiterated in 2017 by the authors. A brief description of the ϵ parameter estimation is presented below:

1. Parameter `k` is chosen, as equal to the value of `min_pts`.
2. For each sample, its distance to the k-th nearest neighbor is computed.
3. For all samples, the previously-obtained values are ordered and visualized, as presented in the sample figure 3.1. This is called the *k-distance graph*.
4. The cutoff point is determined: if the maximal k-distance value (or the "valley" point) before the sudden increase in k-distance values is identified, this becomes the `eps` value for the clustering model.

The reasoning behind this is that this cutoff value determines the generation of clusters for the samples whose k-distance is below it given the stability of their values[1] and it leaves all the samples with k-distance above it as outliers or noise, due to the exponential growth of this distance.

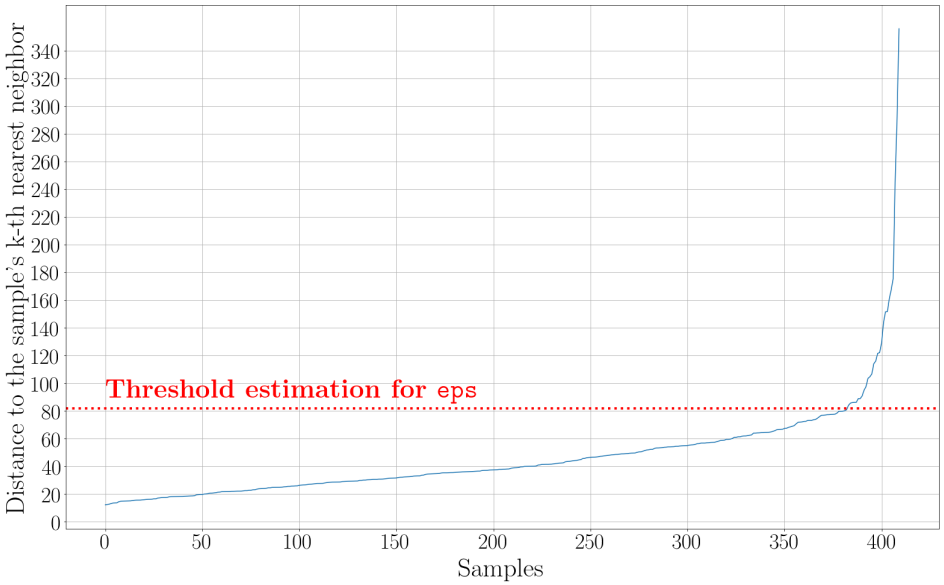


Figure 3.1.: *K-distance* graph: the ordered *k-distances* to each sample are represented by the blue line; the cutoff line is represented in red.

Ground Truth in Parameter Optimization

When the ground truth is known, it can be considered as a form of supervision represented by the cluster labels [23]. Hence, since all custering problems can be considered optimization problems, in this case a parameter search strategy can be used, in a similar way to supervised hyperparameter tuning.

3.3.4. Clustering Evaluation: Measuring the Clustering Quality

Evaluation of clustering can itself be a difficult task and it varies based on the clustering approach as well as the existance or absence of the ground truth labels. This is why there is a significant overlap between parameter optimization and clustering evaluation. Generally clustering validation is performed using an *internal*

[1]Note that since k is set to the `min_pts` parameter, all the samples with k-distance smaller than the cutoff satisfy the cluster creation condition.

/ intrinsic measure or quality score when no information about "correct" clusters exists or *external / extrinsic* when the ground truth data exists.

Internal Evaluation

Internal evaluation of data clustering is performed based on the samples which were clustered by computing an index or score (i.e. the *Silhouette coefficient* [40] or the *Calinski-Harabaz index* [9]). Since these performance criteria are "borrowed" from the objective function optimized by particular clustering techniques [1], the major drawback of such approaches is their limitation to convey a qualitative indicator of the clustering, especially in the context of methods that optimize a different objective function (for example, evaluating DBSCAN clustering results using the scores indicated above, which optimize the objective functions mainly used by centroid-based clustering, respectively hierarchical methods). As a consequence, there is no universal score which could be used for validating results of multiple clustering methods as they heavily depend on the distance metrics as well as the types of clustering approaches.

External Evaluation

External evaluation is performed based on the "ground truth", or sample labeling performed externally by a human expert. The potential problem with this approach is the potentially biased or highly subjective nature of the labels on one hand and the likelihood of low generalization to unseen / unlabeled data on the other hand. Nevertheless, as mentioned by Aggarwal [1], external criteria are preferable to internal ones by virtue of their capacity of avoiding consistent bias in evaluations especially when used to evaluate the clustering of multiple data sets. Moreover, external evaluation can offer a starting point towards a more complex clustering process and result evaluation.However, the usual error computation employed in supervised learning tasks such as precision and recall can no longer be used for external evaluation of the clustering results, as their comparison becomes significantly more complex. Aspects such as similar separation of data into clusters or cluster label permutations need to be taken into account. There are various similarity measures used for clustering evaluation.

One such measure is the **Rand index** [26]. This index computes the similarity of two clusterings by considering all the pairs of samples from both clusterings and counting the pairs which are either in the same (a) or in different (b) clusters. The obtained values are stored in a data structure named the contingency matrix:

$$RI = \frac{a+b}{C_2^{n_{samples}}}. \tag{3.5}$$

The problem with this index is that it does not account for situations where labels are randomly assigned to samples, and as an effect the index is not guaranteed to have a value close to zero when such situations occur. This problem is approached by the **adjusted Rand index** [33], which is computed by the formula

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]}, \tag{3.6}$$

where $E[RI]$ is the expected Rand index of randomly assigned labels.

Another set of important similarity measures for clusterings is the **Mutual Information (MI)**-based scores. Introduced by Bellamy et al. [6], the mutual information is a measure based on information theory. The similarity terminology is here replaced by the *agreement* between the two assignments. The mathematical formulation is based either on the *probabilistic model* of the partitions or clusters or on *set theory*. Thus, the MI score is given by the formula (probabilistic model):

$$MI(U,V) = \sum_{i=1}^{|U|} \sum_{j=1}^{|V|} P(i,j) \log \frac{P(i,j)}{P(i)P(j)}. \tag{3.7}$$

This formula can be easily translated to the set theory by replacing the probabilities with set cardinalities. Finally, the adjusted MI can be computed using the same form presented by formula 3.6. More information about the expectation computation and clustering comparison can be found in the paper *Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance* [51].

In essence, while the *pure* clustering similarity measures provide relative scores, their *normalized* formulas bound the values to the unit interval. In addition, their *adjusted* counterparts ensure that their values are 0 when the compared partitions are random and independent, and 1 when they are identical [39].

3.3.5. Data Visualization

In the field of unsupervised learning in general and especially in clustering, there is a strong emphasis on finding structure in unstructered information (i.e. grouping similar objects). Although there are various methods of validating the quality and soundness of the results, the human nature requires an *intuitive* explanation of a certain behavior. The more abstract the patterns in the data are, the more stringent this need becomes. This is why, a significant part of the unsupervised techniques is dedicated to visualizing data. Perhaps the most challenging part of

viewing unstructured data is the visualization of multivariate data, or data with a large number of dimensions or attributes.

t-SNE The t-Distributed Stochastic Neighbor Embedding, or t-SNE, is an advanced technique used for visualizing multivariate data [48]. The approach uses various mathematical and probabilistic concepts to solve the problem of embedding data characterized by a large amount of features into a highly-reduced space[2]. As it is perfectly synthesized in the Scikit Learn API reference [43], t-SNE first converts similarities between data points to joint probabilities, then it minimizes the Kullback-Leibler divergence between the previously-computed joint probabilities of the low-dimensional embedding and the high-dimensional data. Since t-SNE has a cost function that is not convex, with different initializations we can get different results. Based on the initial approach, SNE, and on the so-called Student-t Distribution instead of a Gaussian to compute similarities, t-SNE solves the problems faced by the initial implementation. This, however, exceeds the scope of this thesis, but the details can be found in the original paper authored by Van Der Maaten and Hinton [48].

[2]Generally, two- or three-dimensional space

4. Method: Clustering the Application Performance Management Problems

This chapter describes the particularities in the approach used to solve the problem introduced in the first chapter. The most notable contributions to the field are the following:

- The data collection and feature extraction task.
- The normalized distance metric introduced for using in conjunction with the clustering algorithm of choice.
- The methodology for optimizing the parameters of the clustering algorithm based on the business-specific, monitoring data.

Based on the user stories whose support is required for this project, a client API for collecting monitoring problem data from the Dynatrace environment has been developed. The API is configuration-based and accepts the following parameters:

- The time interval when problems occurred.
- The Dynatrace backend service endpoint configuration.

This component is called the `DataCollector`. A correctly configured data collector instance can be used to query problem data from a user-defined time period. The client API issues web requests to the Dynatrace service, which sends back data as a JSON-formatted array of monitoring problem objects. Since the amount of received information is considerably large, data selection is also performed at the level of the `DataCollector` for the purpose of ensuring time and memory efficiency throughout the entire process.

4.1. Data Collection

As a result of discussions and feedback received from the stakeholders regarding the relevance of features for the problem at hand, the following information

is extracted from each problem instance in the response body received by the `DataCollector`:

- The problem **id** - *used for reference only.*
- The **timestamp** of the occurrence of the problem - *used for reference only.*
- The **hour of day** when the problem occurred, a value computed based on the timestamp.
- The **monitoring events** composing the problem graph. For each of the events, the following data points are extracted:
  - The **event type**
  - The **entity type/identifier** where the event occurred. The internal naming convention for this attribute is `EntityType_EntityId`. [1]
- The **total duration** of the problem in minutes, computed based on the durations of all the events pertaining to the problem.

The collected data for each problem is aggregated into a data row in the problem data model. Numeric values are persisted as such and event-related data is aggregated into documents (i.e. sequence of words or tokens) containing event types and entity types / entity id's separated by spaces. Each entry in the data set represents a monitoring problem sample.

4.2. The Self-normalizing Distance Metric

In chapter 5 we explain the process of model selection. We emphasize that one of the criteria for choosing a clustering algorithm is the support for custom distance functions or precomputed distances.

4.2.1. Monitoring Problem Data Model

The reason why supporting custom metrics is important for the clustering approach of our choice is the following: as one can notice in figure 4.1, the ranges of attributes can vary significantly. A first obvious step would be to normalize the data set, as we did previously for the model comparison and selction. However,

[1] Generally, the relevant feature is the entity type. However, some entities can be configured internally by the users of the Dynatrace service through a dedicated user interface. Hence, their significance for our problem is higher. So for these entities we collect the entire `EntityType_EntityId` token (i.e. without discarding the `EntityId`).

performing this operation results in important information loss, since attributes would be scaled individually to the same range. This is in principle an issue, since after a careful analysis of the dynamics of values of the monitoring feature samples, we concluded that:

- The impact of features to the overall similarity between two samples is different, depending on the magnitude of the compared values (i.e. the `total_duration` can have values which are orders of magnitude larger than the values of other feaures, so it should influence the similarity to a lesser extent).

- Let $p_i(k)$ and $p_j(k)$ be the values of a feature k of two samples, P_i and P_j. The smaller the difference (or ratio) between $p_i(k)$ and $p_j(k)$, the more similar the samples P_i and P_j are, from the point of view of feature k.

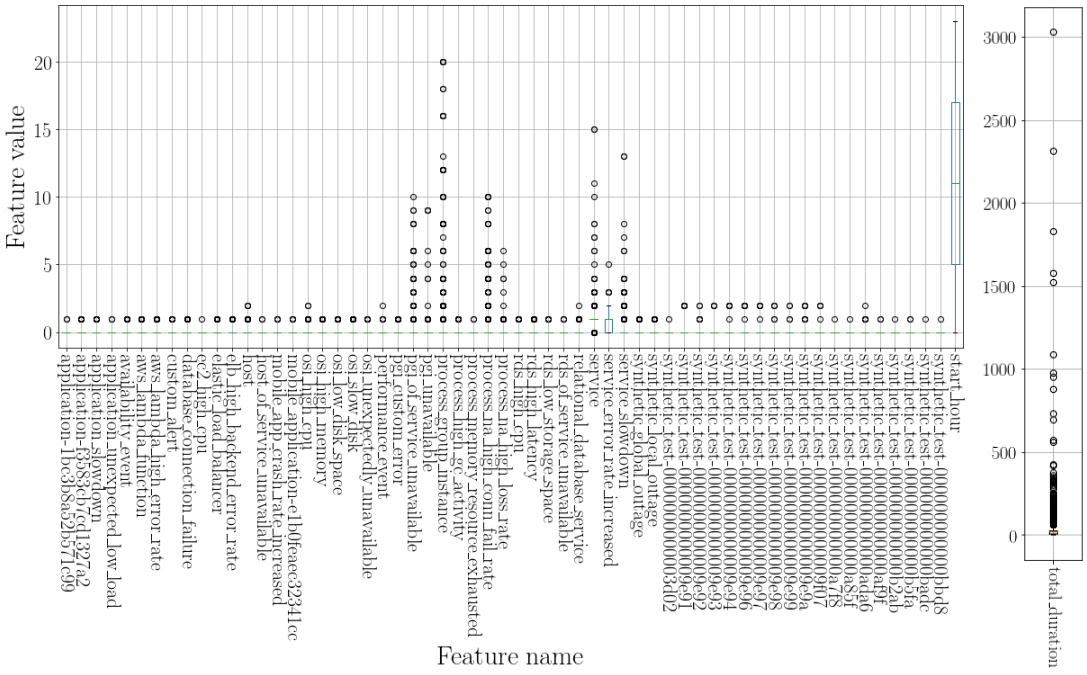


Figure 4.1.: Data distribution for the monitoring problem data set: duration distribution is illustrated separately.

The larger the two values $p_i(k)$ and $p_j(k)$ are, the less impact they should produce on the similarity between P_i and P_j and conversely, the smaller they are – the more impact they should produce. We found that the similarity function which best describes these effects is given by the following formulas. Starting from the

similarity with respect to feature k, we have:

$$s(p_i(k), p_j(k)) = \frac{\min(p_i(k), p_j(k))}{\max(p_i(k), p_j(k))}. \tag{4.1}$$

Subsequently, generalizing this to the entire feature set of the two samples, we obtain

$$s(P_i, P_j) = \sum_{k=1}^{|F|} \frac{\min(p_i(k), p_j(k))}{\max(p_i(k), p_j(k))}, \tag{4.2}$$

where F represents the feature set for the samples.

4.2.2. From Similarity to Distance: Wave Hedges

So far we considered a similarity function. However, clustering algorithms require a distance function for comparing objects. As we emphasized in chapter 3 by introducing equation 3.1, deriving the distance from similarity is trivial. So based on these findings, we searched for an appropriate distance function. The so-called *Wave Hedges* function (3.4) mentioned by Cha [11] adheres to the previously-identified constraints.

A Well-known Caveat The *Wave Hedges* distance metric in its current form, has an obvious caveat: the technical problem caused by the **division by zero**. Especially in the case of our feature set, the chance of 0/0 division is extremely high, due to the sparse representation of the token histogram that we performed on the event type / entity type documents (see subsection 3.2.2). As suggested by Cha [11], these situations require careful attention. So in our implementation, the value 0 is assigned to that term without attempting to perform the division[2].

Next, we show how the *Wave Hedges* distance metric was modified to account for the particularities of our problem.

4.2.3. Distance Metric Optimization

Normalization

In its current form, the *Wave Hedges* metric has the property of **variability of the function codomain**. Each additional feature / dimension added to the data increases the codomain upper bound by 1. Since we wanted to provide reliable

[2]If the values of both samples for an attribute are 0, it means that from the point of view of that attribute, the two samples are perfectly similar, so the distance between them is 0.

metrics over multiple runs and estimate model parameters that can potentially generalize for multiple (disjoint) sets of monitoring problems, we needed to bound the codomain of the distance function to a certain interval. The most obvious method was to bound the codomain to the [0,1] interval by averging the sum:

$$d(P_i, P_j) = \frac{1}{|K|} \sum_{i=1}^{|K|} (1 - \frac{\min(p_i(k), p_j(k))}{\max(p_i(k), p_j(k))}). \tag{4.3}$$

At this point, we concluded that normalizing was no longer needed for the monitoring problem data sets, as long as this distance measure was used. We will refer to this measure as NWH.

Magnitude-sensitive Rectification

Although we performed the most obvious optimizations of the distance metric, there was still one last improvement for the model to completely fit our monitoring problem data. The **"overfitting" of the model for samples whose feature differences are extremely high** while the values themselves are high. This could happen especially for the `total_duration` attribute, where the values are generally many orders of magnitude higher than all the other values (see figure 4.1). In such cases, it is likely that although both `total_duration` values are high, their difference is also high. This could have such a significant impact on the distance metric (as it pushes the corresponding term to 1), that it outweighs the distances between all other features. So, the higher the order of magnitude of attribute values, the higher the chances are that the distance term between them tends to 1, although the significance of the feature in these cases diminishes. To rectify this, a value reduction factor was used to reduce the term contribution proportionally to the order of magnitude of the distance. Hence, our distance function became

$$d(P_i, P_j) = \frac{1}{|K|} \sum_{i=1}^{|K|} \frac{(1 - \frac{\min(p_i(k), p_j(k))}{\max(p_i(k), p_j(k))})}{\log(|p_i(k) - p_j(k)| + 1) + 1}. \tag{4.4}$$

From this point onwards, we will use the NRWH abbreviation to refer to this metric.

Start Hour as a Separate Case

We have presented the distance measure applied to all the dimensions except one: the time of day when the problem started. Since it is a cyclic dimension, where

the next value "above" 23 is 0, it needs to be handled separately. Thus, in order to account for that, the distance between values of this dimension became

$$d(sh_i, sh_j) = \frac{\min(|sh_i - sh_j|, 24 - |sh_i - sh_j|)}{12}. \tag{4.5}$$

By applying this distance metric to all the pairs of samples, we obtained a distance matrix which could be used instead of the data set as input directly to the DBSCAN clustering algorithm. In the next chapter we will use this method throughout the experiments that we performed. The Python code for computing the self-normalizing metric which supports both NWH and NRWH distance metrics is presented in appendix A (listing 2). This metric computation component was implemented using the model adopted by the Scikit Learn toolkit [35] (i.e. using the `fit_transform` function naming and arguments convention).

4.3. Parameter Optimization for the DBSCAN Algorithm

One of the most challenging aspects of cluster analysis is setting the correct parameters for the clustering algorithm. The challenge is amplified by optimizing these parameters so that their *global* values could be successfully used for clustering multiple disjoint sets of samples.

In this thesis we present a novel methodology[3] for finding such parameters by using an optimization technique which is mainly used for hyperparameter optimization in supervised tasks. Various experiments were performed and the parameters obtained were averaged to obtain the *global* value. Then we compare this value to the parameter value obtained by following the *intuitive* approach proposed in the original DBSCAN paper by Ester et al. [19].

[3]Novel in the context of cluster analysis.

5. Experiments

The experiments are divided into two main phases:

1. The model selection phase, covered by section 5.2, where we show how we performed the clustering model selection based on the set of criteria which support our business requirements.

2. The model evaluation phase, covered by section 5.3, where show how we evaluated the model against production data.

The entire process is described together with technical details of the implementation of the project.

5.1. Experiment Setup

For developing this solution, the *Python* programming language was used. Specialized modules were implemented and tested and their functionality exposed through modules for usage in *IPython* [36] (also known as *jupyter*) notebooks, which provide flexibility in the context of a data experimentation-oriented development environment. In terms of libraries, various modules from *Scikit Learn* [35] were used for analysis and clustering, and *Matplotlib* [27] and *Plotly* [28] – for data visualization.

5.1.1. Data Analysis

Problem data is isolated for each customer of the APM suite offered by Dynatrace. However, no customer data was used for the scope of this project. In exchange, data from the self-monitored infrastructure was exposed through a back-end service which was accessible from the Dynatrace internal network. This is, of course, an ideal case in software development, where the development process of an organization includes the use of the developed product not only while testing it, but also for ensuring their services run at nominal parameters. Informally, this is called among the development organizations *dogfooding*. Hence, the data sources for this project are:

- **Development data** or data requested from the service endpoint exposed by the development tenant. This tenant is only used internally for development and test purposes only, and may feature artificially created monitoring problems. Development data was used throughout the development and testing of this project as well as the model selection.
- **Self-monitored Dynatrace production data** or data obtained from the environment provisioned by Dynatrace for monitoring their own production infrastructure[1]. Data obtained from this environment has been used to evaluate the approach proposed by the thesis.

5.2. Model Selection

5.2.1. Data Preprocessing

As we mentioned in section 4.1, the relevant features for the monitoring problem data analysis are the total duration of the problem, the hour of day when it is detected and the event-related document. The first two data attributes are numeric, however the latter is textual. The quality of the data is ensured by the internal components of the Dynatrace software, so we start from the premise that the data quality is ensured.

Since data mining or machine learning models require data in numeric format, once we obtained the data set, the next important step was to transform the textual event content into a numerically represented feature set. It is worth mentioning that we are not interested in the token ordering in the text document, due to the reason that the occurrence order of events in the problem structure is not important and is also highly dependent on the data acquisition mechanism. Hence, we could safely use a bag-of-words transformation technique. As we mentioned in chapter 3, there are various methods which achieve this result. We chose the basic `CountVectorizer`. A more complex vectorization technique such as *Tf-Idf* was not required in this case for the following reasons:

- Each event type / entity type / entity id occurrence is equally important. If a certain token appears with a significantly higher frequency than the others it does not mean that it should be down-weighted.

[1] Although the Dynatrace production environment is monitored, this environment does not involve in any way any type of customer data. However, since the data model is the same across monitored environments, the results presented by this thesis can be generalized across self- and customer-monitored environments.

- Individual vectorized row normalization is not needed, since we have a different normalization strategy, which we will explain in the following section.

5.2.2. Algorithm Comparison

In this section we present the strategy that we adopted for performing cluster analysis. The clustering approaches that we tested are introduced along with some of the advantages and disadvantages that they bring, as well as the approach that we chose in the end.

Cluster Analysis

Since cluster analysis is an iterative, experimental process, as explained in chapter 3, various experiments have been performed to determine the clustering algorithm which is the most suitable for our task. Before describing this process, it is worth noting the challenges posed by our problem:

- *Heterogeneous data types*: as we have seen, problem samples have both numerical as well as textual data attributes.
- *Variable ranges* in terms of the values of the attributes.
- *High dimensionality* of data: the least possible dimension count is 4, in the most simplistic, and rather impractical case[2].
- *Variable dimensionality*, with the variability given by the various event types and/or entities / entity types which can characterize the monitoring problem.

Since there are numerous clustering algorithms to choose from, we limited the evaluation to a subset of the most well-known clustering techniques. Next, we assessed various existing clustering techniques, discussed their benefits and drawbacks, and finally, based on this evaluation, we chose the most suitable technique for our problem.

Comparison and Selection

The comparison phase was itself a two-step task: first we selected a subset of available clustering algorithms, then based on the experiment results, we selected the "best" candidate for solving the problem. Hence, the criterion which we used in the first step to select the candidates for our experiments was the flexibility

[2]This happens when the data set contains problems that have only one event type in their composition (and which occurs at a certain entity / entity type).

of finding the number of clusters: the method should not require the number of clusters as a parameter, as the problem that we have requires an approach which itself should optimize the number of clusters, rather than the clustering approach requiring the optimization of this parameter (such as in the case of *k-means* clustering). This is why *k-means* was ruled out. Another capability which the clustering method should have is finding noise. Since there are samples in our problem space that represent unique problems, we wanted them to be considered outliers / *island* clusters rather than assigning them to the closest centroids, an approach taken by *k-means*. Taking these facts into account, the resulting clustering algorithm candidates were:

1. Affinity Propagation
2. Mean Shift
3. DBSCAN
4. BIRCH.

We performed an experimental clustering of 1010 monitoring problems from the development environment in order to roughly assess the performance of the chosen algorithms based on default parameters and distance metrics, both in terms of runtime and clustering accuracy based on intuitive visual analysis. As normalization technique, the Robust scaler featured by Scikit Learn was used. Visualization of the 28 dimensions was perform by embedding them in 2 dimensions using t-SNE.

The execution time of the algorithms is presented in figure 5.1. It is easy to notice the time inefficiency caused by the iterative message passing performed by *affinity propagation*. On the opposite side, BIRCH and DBSCAN cluster the data well below 100 ms due to their efficiency mechanisms, with BIRCH taking 75% longer to compute the clusters than DBSCAN.

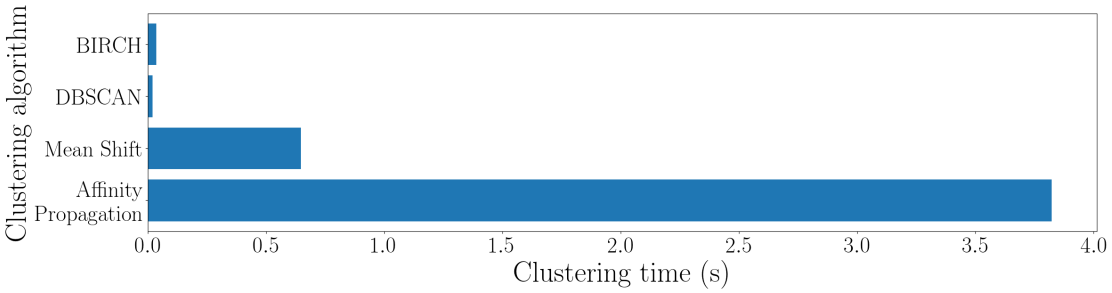


Figure 5.1.: Clustering time for 1010 monitoring problem samples

The cluster labels for each of the four cases is presented in figure 5.2. Based on the coloring of samples[3], one can see how clusters are separated. While affinity propagation does not manage to individualize apparent clusters, mean shift over-clusters the data set. DBSCAN and BIRCH again have comparable results, as they manage to cluster at least apparent "structures" of samples.

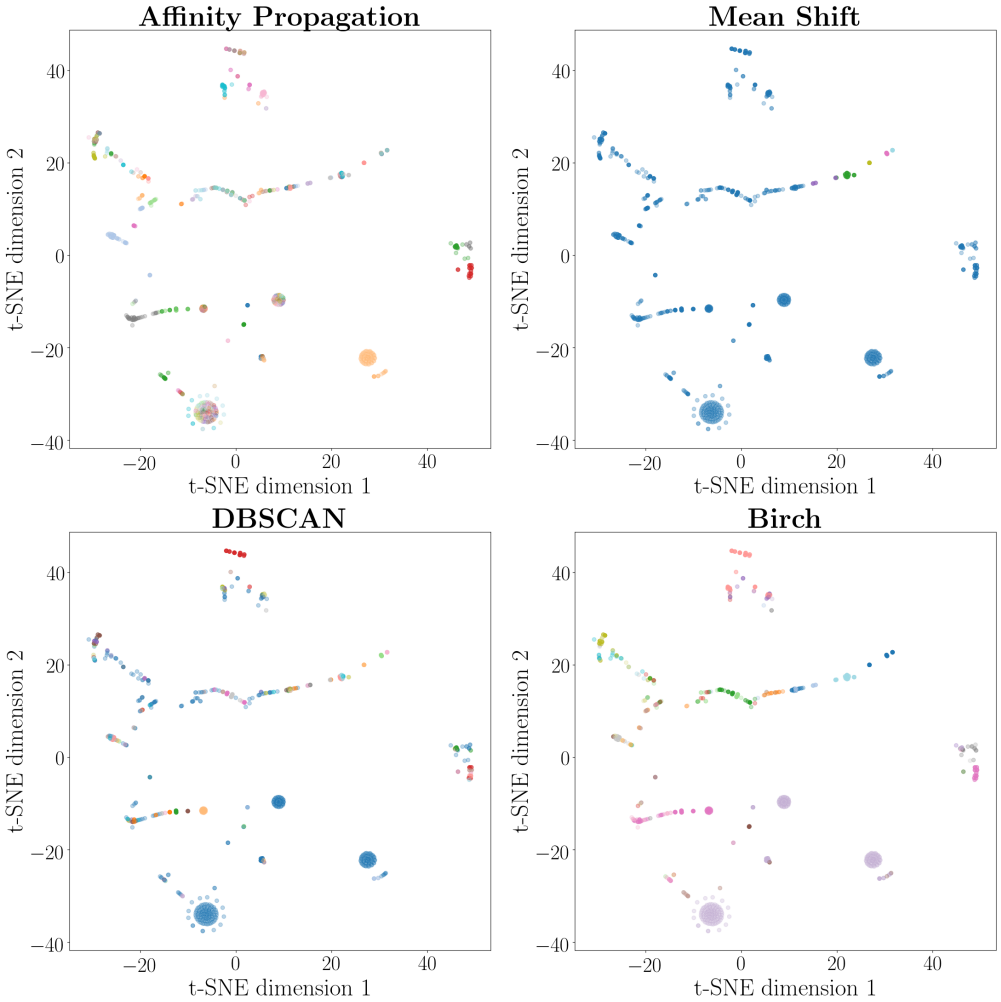


Figure 5.2.: Clustering results. Colors represent cluster labels. *Due to the reduced set in the color scheme, more than one cluster can be depicted using the same color.*

[3]Each cluster is represented by a different color. Due to a high number of clusters though, same colors can be repeated for multiple clusters

Selecting the Clustering Algorithm. The second selection phase has more relevant and requirements-oriented criteria:

- **Time efficiency**: The algorithm should be able to cluster the samples within a time frame which is impercievable by a user, as it would be requested as part of various end-user facing scenarios.
- **Ability to detect outliers**: instead of including all the samples in the cluster models, the algorithm should account for potential outliers which should not belong to clusters.
- **Flexibility in terms of distance metric**: The algorithm should not be tied to a certain metric, but instead it should support a set of predefined and/or custom distance metrics.
- **Arbitrary (concave) shape of clusters**: The algorithm should be able to correctly identify clusters of various "shapes", of course, depending on the chosen distance metric.

The assessment of the selected clustering techniques based on existing documentation as well as the preliminary experiment can be found in table 5.1.

Table 5.1.: Clustering algorithm comparison based on the established criteria

Clustering algorithm	**Time efficiency (ms)**	**Noise detection**	**Distance metric**	**Clusters form factor**
Affinity propagation	3823	No	Negative Euclidean and precomputed[4]	Convex and Concave
Mean Shift	647	No	Internal	Convex and concave
DBSCAN	20	Yes	Predefined and precomputed	Any shape with even density
BIRCH	35	Yes	Euclidean only	Mainly convex

It is easy to see from table 5.1 what the clustering method that fits our problem best is. Not only that it is the fastest algorithm, but **DBSCAN** also features the flexibility of supporting an external distance metric while also being able to detect arbitrary-shaped clusters and the noisy samples around them. In the next section, we present the results of combining this algorithm with the distance metric proposed in section 4.2.

5.3. Evaluation

The purpose of the present thesis is to research and develop a solution to the problem previously stated in chapter 2. Since generally, machine learning solutions to problems do not provide completely correct answers to future data, as in proven mathematical algorithms, evaluation plays a significant role in the process.

5.3.1. Evaluation Test Setup

For the evaluation of the DBSCAN algorithm against the precomputed distance matrix of samples based on the proposed distance function, we switched our attention to data obtained from the self-monitored Dynatrace production environment. We extended our data set to the monitoring problems detected during an entire month[5]. Figure 5.3 depicts an initial look at the entire data set. At a first glance, the t-SNE technique aleady shows an apparent separation of clusters.

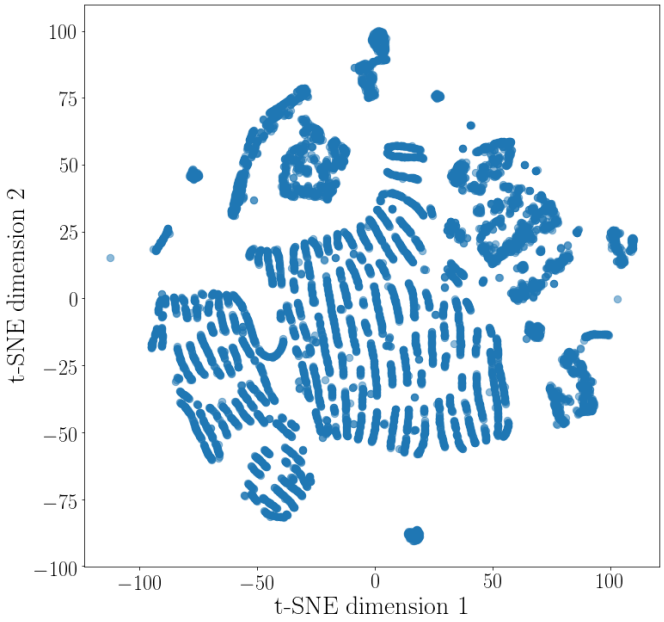


Figure 5.3.: The sample data overview, visualized using 2-dimensional t-SNE

Manual Labeling Methodology. What makes the unsupervised techniques especially challenging to evaluate is the lack of labeled data, also known as "ground truth". In some cases however, the ground truth can be provided by a specialist

[5]The total amount of problems in this data set is 8638

in the problem domain. In the case of the experiments, we labeled the monitoring issue samples based on extensive discussions and feedback from the stakeholders (i.e. the Technical Program Manager and Software Architect). However, since providing labels for such a large data set can be time-consuming, we performed a random sampling of 100 objects from the data set, which would then represent the data set in our evaluation. In order for two monitoring problem samples to be part of the same cluster, the following guidelines have been followed when labeling the data (in this priority):

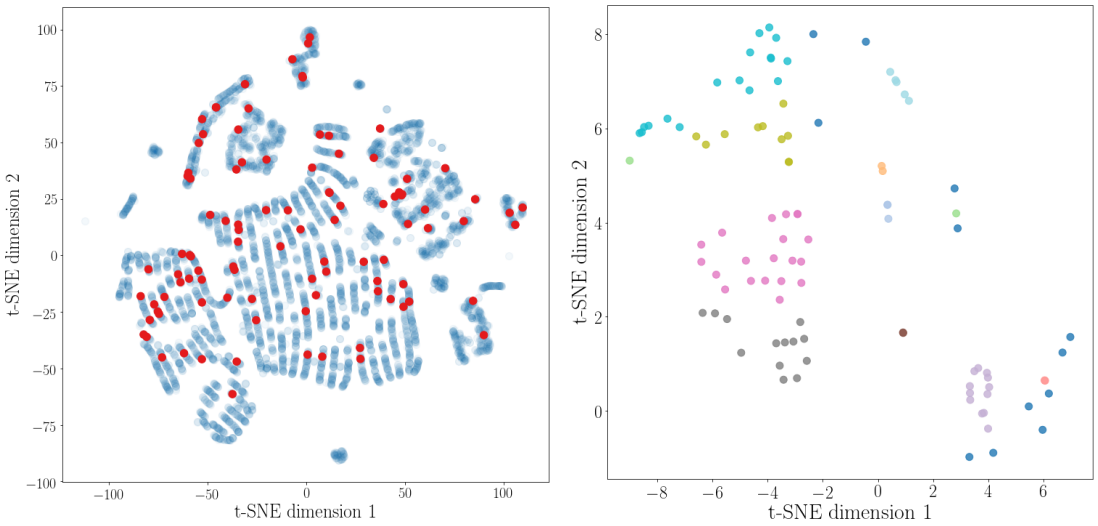
1. The samples must have the same event types as well as the same entity types or applications impacted for all the events composing the problem.
2. The values for the duration of the problems should have the same magnitude.
3. They should occur relatively at the same time of day.
4. Based on the values for `total_duration` and `start_hour`, the priorities for the revious two items can be switched.[6]

Figure 5.4a illustrates the dispersion of the randomly selected samples (red marks) in the context of the entire data set. Projecting the extracted samples in two dimensions separately, one could easily notice that there are some obvious super-clusters. However we could not label the data based on the t-SNE visualization only. This is why we used the data in its tabular form and ordered it based on the event-related document, duration and time of day, as we can observe in figure 5.5. This way it was relatively easy to identify the objects belonging to the same clusters.

Manual Labeling Results. The result of the labeling is illustrated in figure 5.4b. Visualized using t-SNE, the clusters have strongly irregular shapes, which have a low likelihood of being identified using conventional main unsupervied techniques. This supports the importance of using a custom distance metric.

Parameter Optimization and Model Selection. The most important aspect to consider before feeding our data set to the clustering algorithm is the parameter assignment. Having the ground truth simplifies this process by comparing it to the outcomes yielded by the algorithm, for various values of the parameters and choosing the best performing configuration. There are two parameters that need to be optimized for the DBSCAN algorithm: `min_pts` – the minimum number of samples for a sample to be considered a core sample – and , or `eps` – the

[6]For example, if the two samples occurred 12 hours away from each other, the time of day may be a deciding factor to assign the two samples to different clusters.



(a) Unlabelled samples (b) Labelled samples ("ground truth")

Figure 5.4.: Monitoring problem distribution of the 100 randomly chosen samples

threshold under which the distance to other samples should be for those samples to be considered aa core sample. The procedure is described below:

- **`eps`**: for the epsilon parameter to be optimized, we combined the grid search approach usually employed in supervised model selection with external unsupervised evaluation by computing various evaluation metrics based on the ground truth versus the computed label assignments for each of the *eps* values considered in the gridsearch range. Then we selected the epsilon value corresponding to the greatest obtained index.

- **`min_pts`**: in the initial paper on DBSCAN, Sander et al. [41] suggest that generally min_pts should be chosen to be equal to half the number of dimensions in the data set for general scenarios of cluster analysis. Our problem nature however is not on the same line with this general guideline, as monitoring problems can have very strict forms and thus, clusters of as little as two problems can exist[7]. Although this can greatly simplify the model selection, for experimental completeness as well as objectivity reasons we chose to include this parameter in the grid search tuning procedure.

Model Performance Comparison Methodology. After grid search was performed based on the concepts presented in chapter 3 and mentioned previously,

[7]This means that the problems which would be identified as outliers (and thus labeled with -1), would themselves be representing "island" clusters.

id	total_duration	start_hour	text_members	expected_labels
991	3	13	AVAILABILITY_EVENT PGI_CUSTOM_ERROR PROCESS_GROUP_INSTANCE PROCESS_GROUP_INSTANCE	-1
773	5	14	EC2_HIGH_CPU HOST	-1
619	16	10	ELB_HIGH_BACKEND_ERROR_RATE ELASTIC_LOAD_BALANCER	-1
517	5	10	OSI_HIGH_CPU HOST	0
1208	5	16	OSI_HIGH_CPU HOST	0
699	5	10	OSI_HIGH_MEMORY HOST	-1
436	46	0	OSI_HIGH_MEMORY PROCESS_NA_HIGH_CONN_FAIL_RATE HOST PROCESS_GROUP_INSTANCE	-1
533	159	6	OSI_LOW_DISK_SPACE HOST	-1
503	5	1	OSI_SLOW_DISK HOST	1
1637	5	2	OSI_SLOW_DISK HOST	1
2124	5	11	OSI_SLOW_DISK HOST	1
1375	5	13	OSI_SLOW_DISK HOST	1
1116	35,1	9	OSI_UNEXPECTEDLY_UNAVAILABLE PGI_OF_SERVICE_UNAVAILABLE HOST PROCESS_GROUP_INSTANCE	-1
2302	5,37	18	OSI_UNEXPECTEDLY_UNAVAILABLE PGI_OF_SERVICE_UNAVAILABLE PGI_OF_SERVICE_UNAVAILABLE HOST PROCESS_GROUP_INSTA	-1
179	33	8	PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN_FAIL_RATE PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN	-1
1184	17	13	PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN_FAIL_RATE PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN	2
1090	23	8	PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN_FAIL_RATE PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN	2
1015	5	7	PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE PROCESS_GROUP_INSTANCE	4
2118	5	13	PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE PROCESS_GROUP_INSTANCE	4
191	184,04	16	PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE PROCESS_GROUP_INSTANCE	3
1136	196,31	12	PGI_OF_SERVICE_UNAVAILABLE PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE PROCESS_GROUP_INSTANCE	3
2141	0,92	6	PGI_UNAVAILABLE PROCESS_GROUP_INSTANCE	-1
1412	29	19	PROCESS_HIGH_GC_ACTIVITY PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE PROCESS_GROUP_INSTANCE	-1
1326	36	16	PROCESS_HIGH_GC_ACTIVITY PROCESS_NA_HIGH_CONN_FAIL_RATE SERVICE_ERROR_RATE_INCREASED SERVICE_SLOWDOWN PRO	-1
990	5	9	PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE	5
444	5	19	PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE	5
1312	12	5	PROCESS_NA_HIGH_CONN_FAIL_RATE PROCESS_GROUP_INSTANCE	5

Figure 5.5.: The figure depicts an intuitive view of a part of monitoring problem samples from an arbitrary data set which were clustered manually. Each row in the spreadsheet represents a monitoring problem.

the best-performing model was chosen. The following evaluation measures were used for comparison:

1. Adjusted Rand index
2. Normalized Mutual Information index
3. Adjusted Mutual Information index.

All three measures are normalized, which means that their maximum values can reach 1[8]. This optimization process was repeated using two different clustering configurations:

1. DBSCAN algorithm with distance matrix computed using the normalized Wave Hedges function (equation 4.3).
2. DBSCAN algorithm with distance matrix computed using the normalized and rectified Wave Hedges function (equation 4.4).

Thus, the grid search-based evaluation was performed six times: once for each of the *evaluation measure – clustering configuration* combinations. Next, the results

[8]In this case there is a one-to-one similarity between the gound truth labeling and the evaluated one.

were compared to the ground truth, labeling was performed for the entire data set, and in the end, the best performing parameter values – compared to the value obtained by performing the "valley" method suggested generally by literature.

5.3.2. Experiments Statistics

The results of a grid search round can be consulted in figure 5.6. It is easy to notice how the highest AMI index values were obtained for low `min_pts` values[9], which validates our previously-mentioned hypothesis where we mentioned the existance of very small clusters.

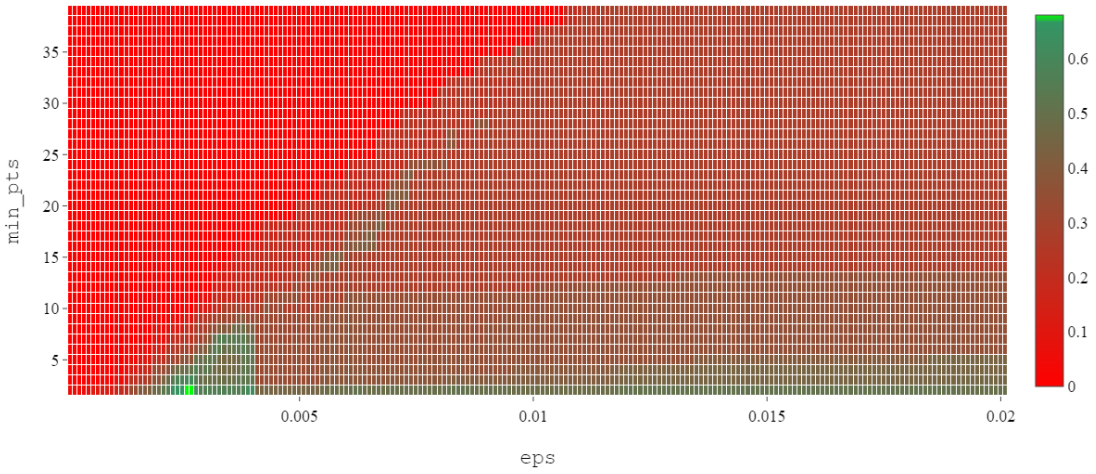
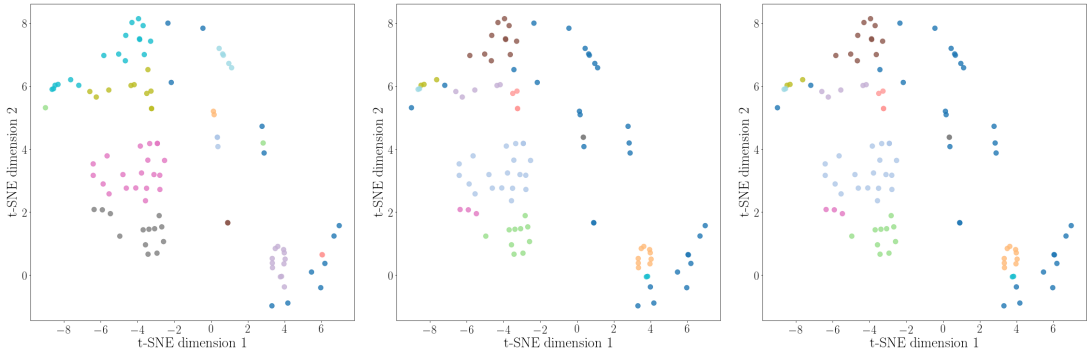


Figure 5.6.: The result of grid search for determining the `eps` (X axis) and `min_pts` (Y axis) parameters in the context of DBSCAN clustering using the normalized and rectified Wave-Hedges metric, based on the adjusted mutual information evaluation measure. The color scheme was chosen to emphasize relative score changes for the parameter configurations: while green indicates the highest (best) possible values for the evaluation measure, red indicates very low values.

Figure 5.7 illustrates the labelings performed by the models using the two configurations, based on the grid search parameter optimization. Comparing to the ground truth (manual) labeling initially performed, clustering based on both distance measure customizations performs reasonably well, with a few exceptions.

[9]For the experiment illustrated in the figure, the value of `min_pts` for which AMI has the highest value, is 2.

They identify considerably more outliers than expected, however the most significant clusters are generally correctly identified.



(a) "Ground truth" labeling (b) Normalized Wave-Hedges (c) Normalized and rectified Wave-Hedges

Figure 5.7.: The t-SNE visual representation of the data clustering performed by DBSCAN using the customized version of the Wave-Hedges distance metric, compared to the ground truth. Each monitoring problem is reprensented by a colored mark; each cluster is represented by a different color.

By carefully inspecting the cluster labels for the clustered problems, we drew the conclusion that the data set contains an uncertainty-inducing source. This source is the `start_hour` dimension, introduced in section 4.1. Taking the problems characterized by the *SERVICE_ERROR_RATE_INCREASED* event as an example, one could notice that they simply occur at different times throughout the entire day. So in these situations, the question that we needed to answer was how the clustering should have been performed. This leads us to the initial manual labeling step or ground truth establishment, where the guideline was that *they should occur relatively at the same time of day* (subsection 5.3.1). This guideline is rather generic and relative, as it depends on the occurrence frequency of the problems in a certain time of day, the amount of time between occcurrences and so on. Having this stated, the ultimate question was whether this classification criterion made sense whatsoever. For answering this question, we needed to recall the business goals of the project; more specifically the usage scenarios and the business justification of grouping the application performance problems. In section 2.2 there is a strong emphasis on the word *similar* (i.e. "groups of similar problems"). This does not completely answer our question, however by understanding the problem similarity in the context of the problem occurrence time, we were able to answer it: given the previous example, the concept of similarity was clearly dilluted by

the occurrence of the same type of problem at relatively arbitrary time during the day so that it covered the entire 24-hour interval. Hence, based on its other attributes (events, entities, duration), we concluded that there was no difference between problems occurring at different times of day and that `start_hour` should not be a splitting criterion in these situations. This is clearly the representative case of problems, especially in the context of global systems, where users around the world use the applications monitored by Dynatrace. However, there are two more situations that we needed to evaluate before drawing a conclusion on the contribution of the `start_hour` to the overall problem similarity:

1. **Applications / services dedicated to users from specific time zones.** Some applications or services are only available for a certain region, contry or time zone. Due to their load peaks, problems can emerge in these situations in certain time intervals. Should this be a sufficient reason to have the `start_hour` as a cluster division criterion? This was only appropriate if the problems occurring outside the (peak) usage time window were different than the ones occurring during the normal usage hours. Although in practice these situations are highly improbable, they are theoretically plausible.

2. **Maintenance operations.** Problems are likely to occur in situtations like periodic redeployment, database backups or other application or service maintenance-related operations. If these operations reoccur at the same time of day, then the emerging problems are as a consequence also recurring. However, their duration and event types entities are in theory constant over time[10], so according to our other clustering criteria, they should be part of the same cluster even without taking the `start_hour` into account. The outstanding case is represented by similarly looking problems[11], but that occur outside of maintenance windows, due to other causes. While highly unlikely in practice, this situation is theoretically possible. Hence, the clustering model should be able to distinguish between the two situations and create two separate clusters. But there is also a chance that unrelated but similar problems[12] occur very close to maintenance windows. So according to the information that they convey, the clustering model would incorrectly assign them to the same cluster. As we can see, for this corner case, the presence of the `start_hour` dimension brings both advantages and disadvantages.

As demonstrated above, although in most cases the `start_hour` attribute does not make any difference in terms of grouping the monitoring problems that occur

[10]The duration of recurring maintenance operations is generally constant over multiple runs, thus the service outages or issues caused by them should also be in the same duration magnitude

[11]Similar in terms of duration, types of events and of entities.

[12]Having a different root cause than maintenance operations, but similar in terms of duration, types of events and of entities.

in a round-the-clock manner, there are specific cases when they could be the key towards correctly labeling the problems. Hence, we decided to reconstruct the experiment using the same data set and the same extracted samples, but this time re-labeling the data according to these conclusions. Hence, the third rule of the *Manual Labeling Methodology* introduced in subsection 5.3.1 became:

> **Depending on the distribution of the problems with similar duration and event and entity structures, the labels should be assigned in the following manner:**
>
> - **If the problems are distributed in a relatively uniform manner throughout the 24 hours of day, the time of occurrence should not be a cluster splitting criterion. Hence, they should belong to the same cluster;**
> - **Otherwise, the furthest away[13] the problems occur from each other and the fewer similar problems there are in the data set, the more this should weigh in for them being assigned to separate clusters.**

Relabeling of the data was performed, the second round taking the previously-mentioned rule into account. Grid search was performed again for the newly labeled data and using the same performance evaluation measures. As illustrated by figure 5.8, the performance score distribution was considerably different than for the previous iteration of the experiment. While the optimal `min_samples` parameter value remained the same, the `eps` value changed considerably by almost an order of magnitude. The overall performance comparison is presented in the subsequent sections.

5.3.3. Quantitative Evaluation

According to Aggarwal [1], evaluating cluster analysis based on an internal measure function is not preferable, as it is biased toward certain clustering methods. For example, the Silhouette coefficient is computed using the mean inter-cluster distance and mean intra-cluster distance. Hence, it is adequate for evaluating labelings performed by clustering algorithms which identify convex- or hyperellipsoid-shaped clusters. Although we considered using such an internal evaluation measure at first, the Silhouette coefficient has a considerably high risk of computing a score which does not objectively assess the real performance of DBSCAN, especially due to the high probability of identified clusters not satisfying the above-mentioned shape constraint.

[13] Time-wise

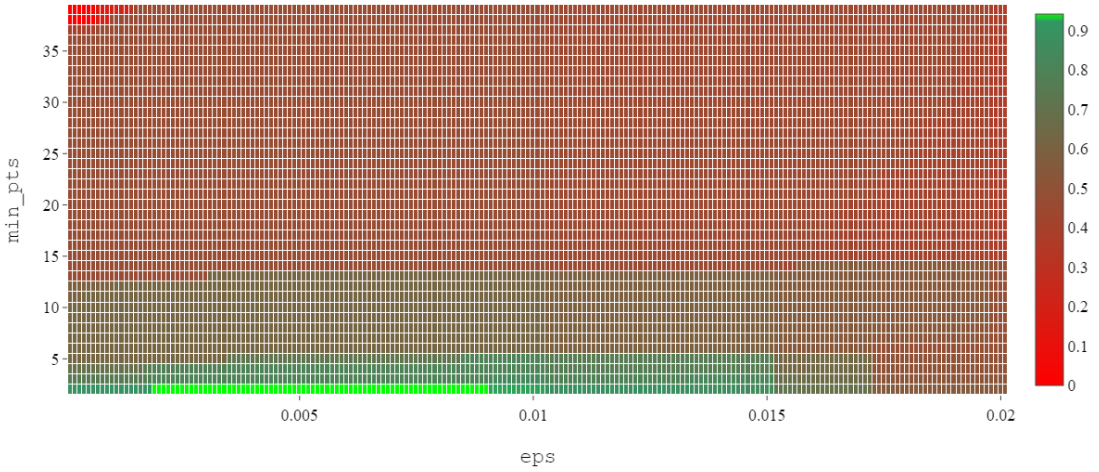


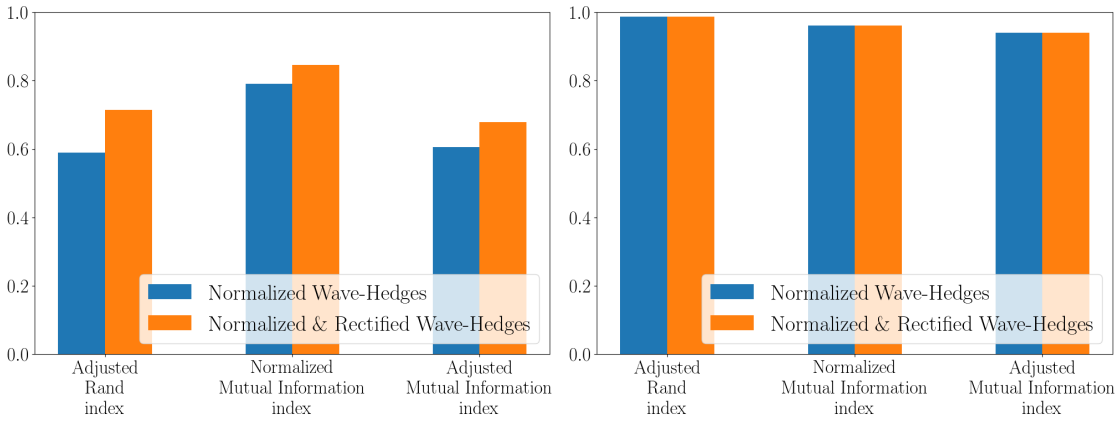
Figure 5.8.: The results of grid search for determining the optimal `eps` (X axis) and `min_samples` (Y axis) in the context of DBSCAN clustering using the normalized and rectified Wave-Hedges metric, based on the adjusted mutual information evaluation measure, as a second iteration of the process presented in figure 5.6.

As a consequence, the decision to externally evaluate the results was taken. We first obtained a representation of the ground truth by manually labeling a subset of representative samples from the entire data set. This enabled us to externally evaluate the proposed clustering method in a less biased manner, by comparing the labels for the ground truth with the ones obtained during grid search using three external evaluation metrics. During the previous subsection we presented how the experiment revealed a very fine, but crucial aspect for the success of the clustering. Hence, the two evaluation iterations for the same set of data.

The evaluation results of the best performing models for these configurations are compared in figure 5.9. By reiterating the experiment and re-tuning the model, the clustering performance of the model was improved by around 38% more in terms of accuracy.[14]

In addition to the pure clustering performance metrics, table 5.2 contains a suite of metrics drawn from the two experiment iterations, and thus, offers a more holistic view upon the clustering performance. While during the first iteration we

[14]We obtained the AMI score of 0.67 in the first iteration and 0.94 in the second iteration.



(a) Evaluation for iteration I of the experiment

(b) Evaluation for iteration II of the experiment

Figure 5.9.: Evaluation of the results obtained by the clustering algorithm using the two modified versions of the Wave-Hedges distance metric, based on the three evaluation measures.

noticed significant differences in terms of clustering performance between the two evaluated configurations[15] for all three measures that we used, the second iteration of grid search yielded the same values for all three performance measures, the only difference being the `eps` parameter value.

In terms of clusters and outliers (non-clustered samples), although during the first iteration we obtained a model with exactly the same number of clusters as the ground truth, the outlier count was almost three times larger than indicated by the ground truth. Despite the incomplete identification of one cluster during the second iteration, the results showed an almost perfect match in terms of outlier count.

To evaluate the clustering similarity we used the three clustering similiarity measures: ARI, NMI and AMI. While ARI differs from the other two by relying on pair-counting, NMI differs from ARI and AMI by not accounting for purely random cluster assignment. Since we want to ensure that randomness is taken into accout, we conclude that our clustering scenario requires and adjusted measure for evaluation. According to Romano et al. [39], AMI should be used in scenarios, where the reference clustering is unbalanced, with potentially high number of small clusters. as a result of our experiment, we obtained clusters of sizes from 2 to 61

[15]NWH and NRWH.

Table 5.2.: Statistics and performance metrics for the best performing configuration of the parameter tuning procedure for both iterations of the experiment.

		Statistics		Parameters		Performance indices		
		Clusts.	Noise	eps	min_pts	ARI	NMI	AMI
Iter. I	NWH	11	28	0.0040	2	0.5057	0.7187	0.6052
	NRWH	11	29	0.0026	2	0.5793	0.7709	0.6793
	Gr. Truth	*11*	*12*					
Iter. II	NWH	7	10	0.0164	2	0.9881	0.9621	0.9409
	NRWH	7	10	0.0119	2	0.9881	0.9621	0.9409
	Gr. Truth	*8*	*11*					

samples. Hence, we concluded that the AMI measure fits best for our problem domain.

5.3.4. Qualitative Evaluation

In table 5.3 we present the computed clusters and their dominant features. Their distribution is solid and according to the ground truth, most of the cases were correctly identified. The exception were the *DATABASE_CONNECTION_FAILURE* problems, which should have been further divided into two clusters containing 2 and 3 problems each. The reason why the ground truth was established this way was that two of the problems occurred in the morning, two in the afternoon, and one at night (which was relatively close in time to the ones occurring in the morning).

While the clustering accuracy was significantly higher in the second iteration, it also became clear that the new model provided a greater flexibility for forming clusters, due to the higher `eps` parameter value. In this case however, as we have seen with the incorrectly-clustered probems, it might be slightly too flexible so it might overfit certain data structures. Based on the business needs, this can be regularized by providing a custom weight to the `start_hour` parameter. Based on the metric implementation provided in chapter 4, having a custom weight to select features in the data set can be performed by parameterizing the constructor with the optional attribute configuration key-value pairs (where the weight is specified as the second parameter of the tuple value). This feature is exemplified in listing 1.

Table 5.3.: Cluster structure comparison of the best performing model with the ground truth. The cluster sizes are expressed based on the dominant features for each cluster.

Cluster dominant feature	Cl. size DBSCAN	Cl. size Ground truth
SERVICE_ERROR_RATE_INCREASED SERVICE	61	61
SERVICE_SLOWDOWN SERVICE	5	5
PROCESS_NA_HIGH_LOSS_RATE PGI	13	13
PGI_OF_SERVICE_UNAVAILABLE ... PGI PGI	2	2
DATABASE_CONNECTION_FAILURE SERVICE	5	2 cls. [2,3]
RDS_HIGH_LATENCY RELATIONAL_DB_SERVICE	2	2
OSI_SLOW_DISK HOST	2	2

```
metric = snm.SelfNormalizingMetric(feature_distance_params={
        'start_hour': ('hour_difference', 1.2)
    })
distance_matrix = metric_transformer.fit_transform(data_set).
```

Listing 1: The interface to the `SelfNormalizingMetric` component: the constructor allows the user to parameterize the distance computation and specify custom weights for specified dimensions (i.e. for the `hour_difference` dimension).

As the performance problem grouping method was evaluated and the results proved to fulfil the requirements of a productizable model, the next step was to evaluate the model on an extended data set, which perfectly resembles the customer data. We chose this data set to be the initially collected data from the self-monitored Dynatrace environment. We need to point out that since the validation suggests the performance of the NRWH metric is not superior than the performance of the normalized Wave-Hedges metric, thus invalidating our initial assumptions, we chose to continue our experiments with the *lightweight* NWH. While this choice saves considerable computation time, it also adheres to the Occam's Razor principle applied to KDD [14], which states that if two models perform similarly in terms of performance / accuracy, the simplest of the two should be chosen. Thus, we performed the clustering of performance problems based on the NWH distance metric. The grouping of problems can be visualized in figure 5.10. The clusters are easily identifiable, with the one very apparent cluster dominating

the entire picture: this is the same *SERVICE_ERROR_RATE_INCREASED SERVICE* problem type that also dominated the downsampled data. This recurring problem accounts for more than half of the entire problem space in the data set, which means that it clearly affects the backend services and should be investigated and fixed. The other important figures that characterize the result of the problem grouping are the total number of clusters (63) and the amount of outliers or unique problems (78), which accounts for less than 1% of the entire collected problem domain.

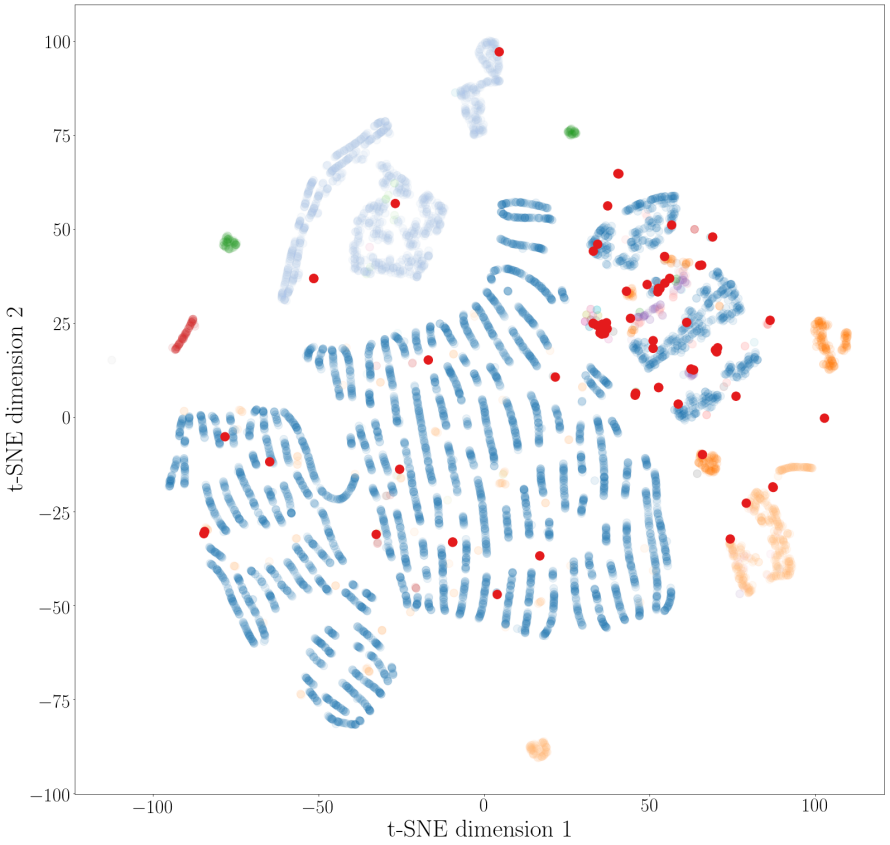


Figure 5.10.: Coloring of the labels produced by the clustering of a data set projected using t-SNE. The data set contains 8638 monitoring problems from the Dynatrace production environment. Clusters are represented by similarly colored transparent marks, while outliers are emphasized as opaque red marks.

5.3.5. Discussion

In this chapter we performed a data clustering experiment where the parameter selection was performed atipically based on external evaluation measures. This approach was not only fruitful in terms of model and hypothesis evaluation, but also in terms of perfecting our goal and model selection derived from the initial business knowledge as well as an initial evaluation of the model. Through this approach, a very significant improvement in terms of clustering accuracy was achieved.

Throughout the experiment, evaluating the type, value ranges and dynamics of the attributes played a key role in the process of model inception and selection. For example, the high dimensionality of the data set was one of the decisive factors for choosing the DBSCAN clustering algorithm from the point of view of simplicity and low computation time, while the volatile nature of the problem duration and occurrence was the key aspect of confirming that this choice is the suitable one for our problem, due to its capability of detecting convex-shaped clusters. Later we discovered the fine, but important line where the occurrence time of problems can, but do not have to further divide larger clusters. This was an important finding not only for the improvement of overall clustering accuracy, but also for simplifying the computation model of the distance between samples in particular and for generally simplifying the clustering model. This also brings the benefit of the model being able to link samples which recur at various times throughout the 24-hour interval of a day into the same cluster, while separating problems occurring at the same or close time intervals from others occurring at completely opposite time.

Table 5.4.: Statistics and performance metrics obtained from clustering three different data sets.

	Dataset size	Dim. count	Best AMI 100 samples	`eps` for best AMI	Cluster count	Outlier count
Experiment 1	8638	61	0.9409	0.0164	63	79
Experiment 2	2516	51	0.9133	0.0180	63	68
Experiment 3	11199	46	0.8288	0.0162	78	82

As we mentioned earlier, we identified the clustering model as being production-ready / accurate in the context of the 1-month data set which we clustered. But one problem still standed. How could this model generalize for clustering performance problem data sets containing a different data distribution, eventually from

completely different environments? To answer this question, further experiments needed to be performed. As a consequence, we performed two more experiments following the procedure from the second iteration of the first experiment, based on completely different data sets extracted from the same environment and using the same 1 month time window, but starting at different points in time. The statistics for these experiments are presented in table 5.4. It is worth pointing out that the best AMI score was obtained in the three different cases for an `eps` value in a rather narrow interval: [0.0162 – 0.0180]. This reinforces our parameter generalization assumption. The overall average **AMI performance of 0.8943** was obtained for an average **`eps` value of 0.0168**. But in order for us to draw a conclusion, we refer back to chapter 3 where we presented the general approach for DBSCAN parameter selection procedure in an intuitive manner and we compare the results obtained through the two methods.

First, the `min_pts` parameter can be generally set to 2, since our hypothesis about the minimum cluster sizes has been confirmed by the grid search procedure in all three experiments, where all the performance measures unanimously identified 2 as the value yielding the best performing models. After setting this parameter, we assessed the `eps` parameter resulted from the tuning procedure, in the context of the intuitive "elbow" method[16]. Figure 5.11 illustrates the relative value between the experimentally-obtained and the intuitive `eps` values. The chart suggests that the intuitive value lies approximately at half distance between the value obtained in the first experiment iteration and the average of the three values obtained using the NWH distance measure. This suggests that while the model from the first iteration was too strict, the second approach could be rather coarse in identifying clusters. On the other hand, one can notice a "hump" in the curve exactly at the level of the highest value, which indicates the trend change from ascending to asymptotic.

In conclusion, the solution of the application performance management issue clustering is rather a solution space, with no obvious optimum. The model can be thus chosen depending on the value of the `eps` parameter of the DBSCAN algorithm, with the suggested interval being bound by the intuitively obtained value and the superior computed value. Based on the business scenario however, it is recommended (and allowed) to have false positives rather than fals negatives. This means that the model should be rather coarse and flexible in terms of clustering than strict. Subsequently, this indicates that a value closer to our computed `eps` value is recommended.

[16]We performed this method for the data set used in the first experiment

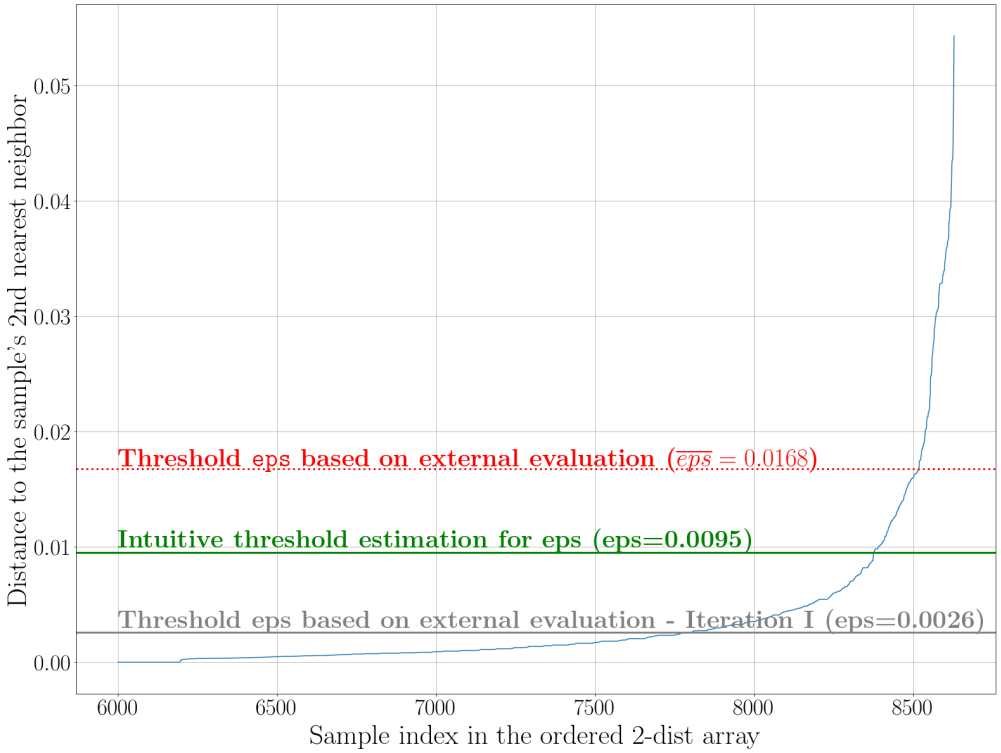


Figure 5.11.: *k-dist* chart showing the distances to the second nearest neighbors, for samples, in ascending order. For a closer and more accurate look, the chart was constrained to the portion of the samples containing the distance inflexion curve elbow. The red dotted horizontal line shows the value of `eps` obtained via grid search in the second experiment iteration. The grey dotted horizontal line shows the value obtained in the first experiment iteration. The green continuous line shows the intuitively obtained `eps` value.

6. Related Work

So far we have seen the great advantages of DBSCAN: fast clustering, flexibility in terms of cluster sizes and shapes as well as custom distance metric compatibility, support for high dimensionality and outlier detection / isolation. But based on our experiments implemented and documented in chapter 5, we discovered how important but at the same time how volatile and difficult to approximate the `eps` parameter value can be. In their recent article, Schubert et al. [42] approach the `eps` parameter selection problem as well as other problems in a structured manner where they also survey alternative algorithms that may alleviate these problems.

The `eps` selection problem. According to Schubert et al. [42], the higher the dimentionality of the data becomes, the harder it is for the `eps` parameter to be correctly estimated, due to the decline of contrast between distances. This problem has been extensively studied by Beyer et al. [8], Houle et al. [25], and Zimek, Schubert, and Kriegel [54]. As the authors suggest, while other algorithms like OPTICS [4] or HDBSCAN [10] optimize the epsilon parameter automatically, they still suffer from the curse of dimensionality when clustering high-dimensional data sets. Returning to DBSCAN, as the authors later suggest, while `eps` should be as small as possible, it also depends on:

- The distance metric
- The application domain and its contstraints.

Optimizations and critiques. In the same article, besides a refresh of the original DBSCAN article [19] with new experimental evaluation and a set of recommendations, Schubert and the authors of DBSCAN also discuss a modified version of DBSCAN proposed by Gan and Tao [22]. While acknowledging the (new) contributions to DBSCAN and the initial inaccuracies indicated by Gan and Tao, the initial authors deconstruct the article proposing the new approach and demonstrate its fragility in terms of:

- Writing style (i.e. subjectivity)
- Incomplete experimentation

- Superficial analysis
- Over-generalization (i.e. they state that their approach should "should replace DBSCAN on big data").

Nevertheless, the newly proposed grid-based approach to optimizing the DBSCAN algorithm can identify regions of high density and prune regions of low density as noise in a very efficient manner and can process multiple points at once. However it is also limited by two disadvantages:

1. **The curse of dimensionality**, which causes the number of possible grid cells to grow exponentially with data dimensionality.
2. **The dependence on Minkowski distances**, which makes it impossible to be used for problems such as the one presented in this thesis, where a different distance metric is used.

In conclusion, the authors show that DBSCAN remains the clustering algorithm of choice for most of the situations where high-dimensional data requires clustering in a time-efficient manner and where noise should be detected and excluded from clusters.

7. Conclusion

In this thesis, we proposed a methodology for clustering business-specific data, where we covered aspects from business analysis and data collection to model selection and evaluation. We performed a comprehensive experiment which followed a logical line where we focused on model improvement while reiterating on the business requirements.

Business analysis. Business requirements were identified and explained in extensive detail, based on formal and informal discussions with company stakeholders[1]. Based on these requirements, besides understanding the terminology, scope and structure of the internal data representation, two artifacts emerged:

1. The data model
2. The set of rules that describe how data[2] should be grouped.

Data collection. Based on the identified data model, a configurable data collection mechanism was implemented. This component is able to query time-bound problems from the Dynatrace service endpoints.

Data preparation. Once the relevant data was collected, we used the *bag of words* approach to transform the documents in numerical representation which is supported by clustering algorithms. Based on the new data structure and feature dispersion, we proposed two levels of customization for a previously-known distance metric function[3], which was used to compute the distance matrix that could be later provided to the clustering algorithm.

Cluster analysis. In order to ensure the suitable algorithm is used for the type of data suggested by this problem, a preliminary cluster analysis was performed, where a range of four well known algorithms were compared. The experiment showed a clear winner: the DBSCAN algorithm.

[1] Technical Program Managers and Software Architects from Dynatrace.
[2] In our case, data sets of application performance management problems.
[3] Wave-Hedges

Clustering model selection and evaluation. In an extensive, 2-iterations experiment, we performed grid search for parameter search and optimization based on external clustering performance measures which used the manual problem labeling as their ground truth. After the first iteration, we studied the dynamics of data in the clustered set and further discussed and correlated business requirements with the clustering algorithm's strengths and limiations. As a result of this discussion, the initial labeling rule set was improved and the labeling was corrected. Then, parameters search was repeated. As a consequence, the following aspects were identified:

- The hypothesis of selecting 2 as the minimum cluster size[4] was validated by the clustering performance scores.
- The `eps` parameter was optimized.
- The second, more complex, customization of the Wave-Hedges distance metric function was discarded after a second clustering round due to no longer visible performance increase over the first, simpler and faster function customization.

Finally, based on the discovered parameters and performance metrics, the selected model was evaluated both quantitatively and qualitatively.

Readiness for use in production environment. After model validation, the clustering of a large set of data was performed and visually inspected. Finally, the parameter configuration was proposed and contrasted to the intuitively obtained parameter[5]. While in theory lower values are better suited for an accurate clustering, business requirements can influence its choice by certain constraints. In our case, the predilection for more flexibility and coarser-grained clusters supported the obtained higher-than-theoretically-recommended `eps` value and the lower-than-theoretically-recommended `min_pts` value.

While it can be proven that the clustering model itself can have limitations, such as generalization to data sets containing completely different problems feature dynamics, it is a robust starting point which could be improved in many different ways. We propose some of them in the next chapter.

[4]The `min_pts` DBSCAN parameter.

[5]The "elbow" method proposed by Ester et al. [19] for approximating the `eps` DBSCAN parameter

To conclude, the proposed clustering model for application performance management issues is a reasonably good fit[6] for deploying it into Dynatrace's production environment.

[6]With a proven AMI score of 0.89 – based on our experiments – presuming the minimum acceptable score would be 0.70.

8. Future Work

We propose two directions for future development of the subject / problem addressed by this thesis:

1. Product engineering
2. Cluster analysis performance optimization research.

8.1. Product Engineering

In terms of product engineering developments, we firstly identify the implementation of the clustering mechanism presented in this thesis, so that it fulfills the business scenarios described in chapter 2. Based on this implementation, we further propose the following development directions:

- Introduction of a collaborative filtering model where the users are able to assess / provide feedback if clusters are too coarse (they accept more monitoring problems than users expect) or too fine-grained (there are more small clusters than expected and some could be consolidated). Based on this feedback, the model could adjust the `eps` parameter value.
- Parameterizing the `eps` parameter so that users can change it based on their preference.
- Automatic naming of identified clusters based on predominant / similar features characterizing the problems in the same cluster.

8.2. Cluster Analysis Performance Optimization

In terms of performance optimization research, we propose the following directions:

- Peform multiple experiments using problem samples from multiple (heterogeneous) environments and by applying the methodology proposed in this thesis and further fine-tune the `eps` parameter to ensure a generic value can be used for multiple data sets.

- Starting from the `eps` value proposed in this thesis and the proposed procedure to approximate it intuitively, we propose researching how its computation could be automated. OPTICS or HDBSCAN are two clustering algorithms which theoretically do not require the `eps` parameter, so they could be considered as baselines.

- Model problems as graph structures and perform graph clustering on this data. Each additional dimension (i.e. duration and start time of problems) could be fragmented as event attributes / weights as part of the problem graphs.

While we provided a few potential development directions, the area of unsupervised learning is overwhelmingly vast and models can be infinitely improved. Thus, since the thesis addresses a business problem, we also recommend that the effort/cost-to-outcome ratio is kept in mind while attepting to improve an existing methodology.

A. Code listings

Listing 2: The generic component used to compute the Wave-Hedges-based distance metric customizations.

```
import numpy as np
import pandas as pd
import math as m


class SelfNormalizingMetric:

    def __init__(self, feature_distance_params=None):
        """
        Initializes a new instance of the SelfNormalizingMetric class.
        :param feature_distance_params: A dictionary of parameters in the form
        feature_index: ( distance_method, weight )
        The currently supported distance_method is only 'hour_difference'.
        """
        self._feature_distance_params = feature_distance_params

        if self._feature_distance_params is not None:
            if len(self._feature_distance_params.keys()) < 1:
                raise ValueError('The "feature_distance_params" should contain key-value pairs.')
            for key in self._feature_distance_params.keys():
                if self._feature_distance_params[key][0] not in [None, 'hour_difference'] or \
                        type(self._feature_distance_params[key][1]) != float:
                    raise ValueError('The "feature_distance_params" should contain key-value \
                                      pairs of the form feature_index: (distance_method, weight)')

    def fit_transform(self, X, normalization_only=False):
        """
        Transforms the data set provided as argument by computing the
        self-normalizing distances between the samples.
        :param X: The data set in the form n_samples x n_features
        :return: The distance matrix.
        """
        if type(X) != pd.DataFrame:
            raise ValueError("The input matrix needs to be of the pandas DataFrame format.")

        if self._feature_distance_params is not None:
            self._transform_indexes(X.columns.values)
        X = X.values

        if type(X) != np.ndarray:
            X = np.array(X)
        distance_matrix = np.zeros([X.shape[0], X.shape[0]])
        for i in range(X.shape[0]):
            for j in range(i + 1):
                distance_matrix[i, j] = distance_matrix[j, i] \
                    = self._compute_distance(X[i], X[j], normalization_only)
        return distance_matrix
```

```
@staticmethod
def _min_max(self, a, b):
    return 1 if a == b == 0 else min(a, b) / max(a, b)

def _compute_distance(self, x, y, normalization_only):
    if x.shape != y.shape:
        raise ValueError('The samples do not have the same number of features.')

    if normalization_only:
        distances = [(1 - self._min_max(f_x, f_y)) for f_x, f_y in zip(x, y)]
    else:
        distances = [(1 - self._min_max(f_x, f_y)) /
                     (m.log10(m.fabs(f_x - f_y) + 1) + 1) for f_x, f_y in zip(x, y)]

    if self._feature_distance_params is not None:
        for key, params in self._feature_distance_params.items():
            distances[key] = distances[key] * params[1] if params[0] is None \
                else getattr(self, '_' + params[0])(x[key], y[key]) * params[1]
    return np.sum(distances) / len(distances)

def _hour_difference(self, a, b):
    if a < 0 or a >= 24 or b < 0 or b >= 24:
        raise ValueError('Compared values are not hours of day:', a, b)
    return min(abs(a - b), 24 - abs(a - b)) / 12.

def _transform_indexes(self, columns):
    columns = list(columns)
    keys = list(self._feature_distance_params.keys())
    for key in keys:
        self._feature_distance_params[columns.index(key)] = self._feature_distance_params.pop(key)
```

List of Figures

List of Tables

List of Code Snippets

Glossary

cloud computing A paradigm or practice which refers to the pooling of compute resources exposed through the Internet, that can be used discretionarily and for which the payment is done based on the amount of resources consumed. This paradigm shifts the use of locally hosted resources to the cloud-hosted resources.. 2

distance matrix A square symmetric matrix, whose elements, a_{ij}, represent the distances between samples i and j, computed based on a distance function.. 27, 34

Euclidean norm Also known as the length of vector $x = (x_1, ..., x_n)$, the norm is computed by the formula $\|X\|_2 = \sqrt{x_1^2 + ... + x_n^2}$. 13

grid search Technique used in machine learning for (hyper)parameter optimization, also known under the name of *parameter sweep*, which performs a model search based on parameters within specified ranges, guided by a performance metric.. 36, 37, 38, 41, 42, 43, 48, 49, 53, 59, 60

Acronyms

AMI Adjusted mutual information. 38, 42, 43, 44, 47, 48, 54

API Application Programming Interface. 21, 22

APM Application performance monitoring / Management. 5, 11, 12, 28

ARI Adjusted Rand index. 43, 44

BIRCH Clustering algorithm: Balanced iterative reducing and clustering using hierarchies. 15, 31, 32, 33

DBSCAN Clustering algorithm: Density-based spatial clustering of applications with noise. 15, 16, 17, 19, 27, 31, 32, 33, 34, 35, 36, 37, 38, 39, 41, 42, 47, 48, 50, 51, 52, 53, 59, 60

HDBSCAN Clustering algorithm: Hierarchical density-based spatial clustering of applications with noise. 50, 56

JSON JavaScript Object Notation. 22

KDD Knowledge discovery in data. 12, 45

NMI Normalized mutual information. 43, 44

NRWH Clustering using the normalized and rectified Wave-Hedges distance metric. 26, 27, 43, 44, 45

NWH Clustering using the normalized Wave-Hedges distance metric. 26, 27, 43, 44, 45, 48

OPTICS Clustering algorithm: Ordering points to identify the clustering structure. 50, 56

R&D Research and development. 5

SaaS Software-as-a-Service. 1

SLA Service-level agreement. 11

t-SNE t-Distributed Stochastic Neighbor Embedding. 21

TF-IDF Term frequency – inverse document frequency. 13

VSM Vector space model. 13

Bibliography

[1] Charu C. Aggarwal. *Data Mining: The Textbook*. 2015, p. 734. ISBN: 9783319141411. DOI: 10.1007/978-3-319-14142-8. arXiv: arXiv:1011.1669v3.

[2] Kabir Ahmed et al. "Similarity Analysis of Industrial Alarm Flood Data". In: *IEEE Transactions on Automation Science and Engineering* 10.2 (Apr. 2013), pp. 452–457. ISSN: 1545-5955. DOI: 10.1109/TASE.2012.2230627. URL: http://ieeexplore.ieee.org/document/6419854/.

[3] Ernst Ambichl et al. *Method And System For Real-Time Causality And Root Cause Determination Of Transaction And Infrastructure Related Events Provided By Multiple, Heterogeneous Agents*. Sept. 2016. URL: https://patents.google.com/patent/US20170075749A1/en (visited on 07/01/2018).

[4] Mihael Ankerst et al. "OPTICS: Ordering points to identify the clustering structure". In: *ACM Sigmod Record* (1999), pp. 49–60. ISSN: 01635808. DOI: 10.1145/304182.304187. arXiv: 10.1.1.71.1980. URL: http://dl.acm.org/citation.cfm?id=304187.

[5] Jean Babaud et al. "Uniqueness of the Gaussian Kernel for Scale-Space Filtering". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-8.1 (Jan. 1986), pp. 26–33. ISSN: 0162-8828. DOI: 10.1109/TPAMI.1986.4767749. URL: http://ieeexplore.ieee.org/document/4767749/.

[6] John Bellamy et al. *Elements of Information Theory*. Wiley, 1991. ISBN: 0-471-20061-1. URL: http://www.academia.edu/download/31823797/information%7B%5C_%7Dtheory.pdf.

[7] Richard Bellman. *Dynamic programming*. Dover Publications, 2003, p. 340. ISBN: 0486428095.

[8] Kevin Beyer et al. "When Is "Nearest Neighbor" Meaningful?" In: 1999, pp. 217–235. ISBN: 978-3-540-65452-0. DOI: 10.1007/3-540-49257-7_15. arXiv: 9780201398298. URL: https://rd.springer.com/content/pdf/10.1007%7B%5C%%7D2F3-540-49257-7%7B%5C_%7D15.pdf%20http://link.springer.com/10.1007/3-540-49257-7%7B%5C_%7D15.

[9] T. Calinski and J. Harabasz. "A dendrite method for cluster analysis". In: *Communications in Statistics - Theory and Methods* 3.1 (1974), pp. 1–27. DOI: 10.1080/03610927408827101. URL: http://www.tandfonline.com/doi/abs/10.1080/03610927408827101.

[10] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. "Density-Based Clustering Based on Hierarchical Density Estimates". In: *Advances in Knowledge Discovery and Data Mining* (2013), pp. 160–172. ISSN: 16113349, 03029743. DOI: 10.1007/978-3-642-37456-2_14. arXiv: 1602.03730. URL: http://link.springer.com/10.1007/978-3-642-37456-2%7B%5C_%7D14.

[11] Sung-hyuk Cha. "Comprehensive Survey on Distance / Similarity Measures between Probability Density Functions". In: *International Journal of Mathematical Models and Methods in Applied Sciences* 1.4 (2007), pp. 300–307. ISSN: 14337347. DOI: 10.1007/s00167-009-0884-z. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.154.8446%7B%5C&%7Drep=rep1%7B%5C&%7Dtype=pdf.

[12] Soumen Chakrabarti et al. "Data Mining Curriculum: A Proposal (Version 1.0)". In: *ACM SIGKDD* (2006). URL: http://www.kdd.org/exploration%7B%5C_%7Dfiles/CURMay06.pdf.

[13] Yue Cheng, Iman Izadi, and Tongwen Chen. "Pattern matching of alarm flood sequences by a modified Smith–Waterman algorithm". In: *Chemical Engineering Research and Design* 91.6 (June 2013), pp. 1085–1094. ISSN: 0263-8762. DOI: 10.1016/J.CHERD.2012.11.001. URL: https://www.sciencedirect.com/science/article/pii/S0263876212004261.

[14] Pedro Domingos. "The Role of Occam's Razor in Knowledge Discovery". In: *Data Mining and Knowledge Discovery* 3 (1999), pp. 409–425. URL: https://link.springer.com/content/pdf/10.1023/A:1009868929893.pdf.

[15] David Dubin. "The Most Influential Paper Gerard Salton Never Wrote". In: *LIBRARY TRENDS* 52.4 (2004), pp. 748–764. URL: https://www.ideals.illinois.edu/bitstream/handle/2142/1697/Dubin748764.pdf?sequence=2.

[16] Dynatrace. *Application performance management (APM): Monitor and manage the performance and availability of software applications to optimize customer experience.* URL: https://www.dynatrace.com/capabilities/application-performance-management/ (visited on 06/16/2018).

[17] Dynatrace. *Dynatrace Official Documentation - Problem detection & analysis.* URL: https://www.dynatrace.com/support/help/problem-detection-and-analysis/ (visited on 07/09/2018).

[18] Dynatrace. *PurePath Explained*. URL: https://community.dynatrace.com/community/display/DOCDT65/PurePath+Explained (visited on 06/16/2018).

[19] Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: (1996). URL: https://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf.

[20] Vladimir Estivill-Castro. "Why so many clustering algorithms — A Position Paper". In: (). URL: http://web.cs.iastate.edu/%7B~%7Dhonavar/clustering-survey2.pdf.

[21] Brendan J Frey and Delbert Dueck. "Clustering by Passing Messages Between Data Points". In: *Science* 315 (2007).

[22] Junhao Gan and Yufei Tao. "DBSCAN Revisited: Mis-Claim, Un-Fixability, and Approximation". In: *Sigmod*. New York, New York, USA: ACM Press, 2015, pp. 519–530. ISBN: 9781450327589. DOI: 10.1145/2723372.2737792. URL: http://dl.acm.org/citation.cfm?doid=2723372.2737792.

[23] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques - Third Edition*. Elsevier Science, 2011, p. 744. ISBN: 0123814790.

[24] Sepp Hochreiter. *Machine Learning: Unsupervised Techniques - Lecture Notes*. Linz. URL: http://www.bioinf.jku.at/teaching/current/ss%7B%5C_%7Dvl%7B%5C_%7Dmlut/ (visited on 07/02/2018).

[25] Michael E. Houle et al. "Can shared-neighbor distances defeat the curse of dimensionality?" In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6187 LNCS (2010), pp. 482–500. ISSN: 03029743. DOI: 10.1007/978-3-642-13818-8_34. URL: http://www.dbs.ifi.lmu.de.

[26] Lawrence Hubert and Phipps Arabie. "Comparing partitions". In: *Journal of Classification* 2.1 (Dec. 1985), pp. 193–218. ISSN: 0176-4268. DOI: 10.1007/BF01908075. URL: http://link.springer.com/10.1007/BF01908075.

[27] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing In Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[28] Plotly Technologies Inc. *Collaborative data science*. 2015. URL: https://plot.ly (visited on 07/02/2018).

[29] Paul Jaccard. "Étude comparative de la distribution florale dans une portion des Alpes et des Jura". In: *Bulletin del la Société Vaudoise des Sciences Naturelles* 37 (1901), pp. 547–579.

[30] R P Jagadeesh et al. “Context Aware Trace Clustering: Towards Improving Process Mining Results”. In: *Proceedings of the 2009 SIAM International Conference on Data Mining* (Apr. 2009), pp. 401–412. DOI: 10.1137/1.9781611972795.35. URL: http://epubs.siam.org/doi/abs/10.1137/1.9781611972795.35%20https://epubs.siam.org/doi/pdf/10.1137/1.9781611972795.35.

[31] Eamonn Keogh and Abdullah Mueen. “Curse of Dimensionality”. In: *Encyclopedia of Machine Learning and Data Mining*. Boston, MA: Springer US, 2017, pp. 314–315. DOI: 10.1007/978-1-4899-7687-1_192. URL: http://link.springer.com/10.1007/978-1-4899-7687-1%7B%5C_%7D192.

[32] James Lewis and Martin Fowler. *Microservices - a definition of this new architectural term*. 2014. URL: https://www.martinfowler.com/articles/microservices.html (visited on 07/01/2018).

[33] Leslie C. Morey and Alan Agresti. “The Measurement of Classification Agreement: An Adjustment to the Rand Statistic for Chance Agreement”. In: *Educational and Psychological Measurement* 44.1 (Mar. 1984), pp. 33–37. ISSN: 0013-1644. DOI: 10.1177/0013164484441003. URL: http://journals.sagepub.com/doi/10.1177/0013164484441003.

[34] Karl Pearson. “On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling”. In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 50.302 (July 1900), pp. 157–175. DOI: 10.1080/14786440009463897. URL: https://www.tandfonline.com/doi/full/10.1080/14786440009463897.

[35] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[36] Fernando Pérez and Brian E. Granger. “IPython: a System for Interactive Scientific Computing”. In: *Computing in Science and Engineering* 9.3 (May 2007), pp. 21–29. ISSN: 1521-9615. DOI: 10.1109/MCSE.2007.53. URL: http://ipython.org (visited on 07/02/2018).

[37] Andres Quiroz et al. “Robust clustering analysis for the management of self-monitoring distributed systems”. In: *Cluster Computing* 12.1 (Mar. 2009), pp. 73–85. ISSN: 1386-7857. DOI: 10.1007/s10586-008-0068-5. URL: http://link.springer.com/10.1007/s10586-008-0068-5.

[38] Vicent Rodrigo et al. “Causal analysis for alarm flood reduction”. In: *IFAC-PapersOnLine* 49.7 (Jan. 2016), pp. 723–728. ISSN: 2405-8963. DOI: 10.1016/J.IFACOL.2016.07.269. URL: https://www.sciencedirect.com/science/article/pii/S2405896316304761.

[39] Simone Romano et al. “Adjusting for Chance Clustering Comparison Measures”. In: (2015). ISSN: 15337928. arXiv: 1512.01286. URL: http://arxiv.org/abs/1512.01286.

[40] Peter J. Rousseeuw. “Silhouettes: A graphical aid to the interpretation and validation of cluster analysis”. In: *Journal of Computational and Applied Mathematics* 20 (Nov. 1987), pp. 53–65. ISSN: 0377-0427. DOI: 10.1016/0377-0427(87)90125-7. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257?via%7B%5C%%7D3Dihub.

[41] Jörg Sander et al. “Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications”. In: *Data Mining and Knowledge Discovery* 2.2 (1998), pp. 169–194. ISSN: 13845810. DOI: 10.1023/A:1009745219419. arXiv: 10.1.1.71.1980. URL: http://link.springer.com/10.1023/A:1009745219419.

[42] Erich Schubert et al. “DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN”. In: *ACM Transactions on Database Systems* 42.3 (2017), pp. 1–21. ISSN: 03625915. DOI: 10.1145/3068335. URL: http://dl.acm.org/citation.cfm?doid=3129336.3068335.

[43] Scikit-learn. *t-SNE API Reference*. URL: http://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html (visited on 07/02/2018).

[44] *Service-Oriented Architecture*. URL: http://www.opengroup.org/soa/source-book/soa/index.htm (visited on 07/01/2018).

[45] R. Sibson. “SLINK: An optimally efficient algorithm for the single-link cluster method”. In: *The Computer Journal* 16.1 (Jan. 1973), pp. 30–34. ISSN: 0010-4620. DOI: 10.1093/comjnl/16.1.30. URL: https://academic.oup.com/comjnl/article-lookup/doi/10.1093/comjnl/16.1.30.

[46] Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. “Trace Clustering in Process Mining”. In: Springer, Berlin, Heidelberg, 2009, pp. 109–120. DOI: 10.1007/978-3-642-00328-8_11. URL: http://link.springer.com/10.1007/978-3-642-00328-8%7B%5C_%7D11%20http://www.processmining.org/%7B%5C_%7Dmedia/publications/song2008a.pdf.

[47] R. Vaarandi. “A data clustering algorithm for mining patterns from event logs”. In: *Proceedings of the 3rd IEEE Workshop on IP Operations & Management (IPOM 2003) (IEEE Cat. No.03EX764)* (2003), pp. 119–126. ISSN: 03029743. DOI: 10.1109/IPOM.2003.1251233. arXiv: 1008.3701. URL: http://ieeexplore.ieee.org/document/1251233/.

[48] Laurens Van Der Maaten and Geoffrey Hinton. "Visualizing Data using t-SNE". In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605. URL: http://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

[49] Tao Wang et al. "Self-adaptive cloud monitoring with online anomaly detection". In: *Future Generation Computer Systems* 80 (Mar. 2018), pp. 89–101. ISSN: 0167-739X. DOI: 10.1016/J.FUTURE.2017.09.067. URL: https://www.sciencedirect.com/science/article/pii/S0167739X1730376X.

[50] David H Wolpert and William G Macready. "No Free Lunch Theorems for Optimization". In: *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION* 1.1 (1997). URL: https://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf.

[51] Nguyen Xuan Vinh, Unsweduau Julien Epps, and James Bailey. "Information Theoretic Measures for Clusterings Comparison: Variants, Properties, Normalization and Correction for Chance". In: *Journal of Machine Learning Research* 11 (2010), pp. 2837–2854. URL: http://jmlr.csail.mit.edu/papers/volume11/vinh10a/vinh10a.pdf.

[52] F. Yang et al. "Improved correlation analysis and visualization of industrial alarm data". In: *ISA Transactions* 51.4 (July 2012), pp. 499–506. ISSN: 0019-0578. DOI: 10.1016/J.ISATRA.2012.03.005. URL: https://www.sciencedirect.com/science/article/pii/S0019057812000341.

[53] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. "BIRCH: An Efficient Data Clustering Databases Method for Very Large". In: *Proceedings of the 1996 ACM SIGMOD international conference on Management of data - SIGMOD '96* 1 (1996), pp. 103–114. ISSN: 01635808. DOI: 10.1145/233269.233324. URL: http://portal.acm.org/citation.cfm?doid=233269.233324.

[54] Arthur Zimek, Erich Schubert, and Hans-peter Kriegel. "REVIEW A Survey on Unsupervised Outlier Detection in High-Dimensional Numerical Data". In: (2012). DOI: 10.1002/sam.

Vlad I Precup

Mehedinti 37/30
400675 Cluj-Napoca, Romania

Phone: +40 743 013 868
Personal Email: vlad.precup@live.com

Work Experience

Feb. 2018 – Present
Dynatrace – Master Project Collaborator – *Linz*
Research and implementation of an end-to-end cluster analysis methodology it in the context of Application Performance Monitoring.

May 2017 – Aug. 2017
Ve Interactive – Technical Lead – *Cluj-Napoca*
Responsible with driving the technical strategy of the merchant integration solutions. Among the most notable contributions:
- Migration of the integration back-end services from IaaS to PaaS and streamlining the monitoring, diagnostics and telemetry story using the Azure Cloud Services.
- Driving the integration with Google services end to end, from architecture, through the implementation and testing, to release, in a team of 7.
- Closely managing and mentoring two SCRUM team members.

Oct. 2016 – April 2017
Ve Interactive – Senior Software Engineer – *Cluj-Napoca*
Member of the merchant integration / licensing team with technical leadership valence.
- Responsibilities: Technical direction of the integration of the product into external systems.
- Technologies: VSO, Git, VS, C#, Azure Services, Octopus.

Dec. 2014 – Aug. 2016
Microsoft – Software Engineer II – *Copenhagen*
Member of the Core Marketing team within the Dynamics CRM – Marketing organization.
- *Responsibilities:* Scrum Master and backlog owner (2016); Design, development and test of a high-volume ingestion engine for the next-gen Marketing services (2016); Development and test of a cross-platform controls suite for the next-gen CRM client (2015); Perf and resilience of the Marketing services (2014).
- *Technologies:* TFS, Git, VS, C#, SQL, TypeScript, JavaScript, CSS3, HTML5, Microsoft Azure

Sep. 2014 – Nov. 2014
Microsoft – Software Engineer II – *Copenhagen*
Member of the Server and Tools engineering team for the Microsoft Dynamics NAV ERP.
- *Responsibilities:* Development and testing of the single sign-on mechanism, federated user authentication, integration with Microsoft Azure AD across the client applications (Web, Windows and tablet), integration with SharePoint Online, integration with Microsoft Office products as well as performance testing.
- *Technologies:* TFS, VS, C#, WIF, PowerShell, JavaScript, Script#, Microsoft Azure AD

Feb. 2013 – Aug. 2014
Microsoft – Software Engineer – *Copenhagen*
Member of the Office 365 Integration team for the Dynamics NAV ERP.
- *Responsibilities:* implementation and design of the federated user authentication, automated configuration scripts, integration into the SharePoint Online portal, OData concurrency.
- *Technologies:* TFS, VS, C#, WIF, PowerShell, Microsoft Azure AD, OData

Aug. 2011 – Jan. 2013
Microsoft – Software Development Engineer in Test – *Copenhagen*
Member of the Web Client team for the Dynamics NAV ERP.
- *Responsibilities:* Designed and implemented a Request-Response test framework whose main purpose was to provide long-time support for a part of the regression test automation for the web client features. Additional responsibilities included user authentication testing and automation, deployment testing.

Jul. 2010 – Oct. 2010
Fortech S.R.L. – 4 Month internship – *Cluj-Napoca*
Outsourcing Team Project: The migration of a large ERP software from WPF to Silverlight.

Education and qualifications

2017 – Present	***Johannes Kepler* Universität (Linz, Austria)** MSc in Universal Computing and Business with strong focus on Machine Learning and Artificial Intelligence	Graduation: *Estimated August 2018*
2016 – Present	***Babeș-Bolyai* University (Cluj-Napoca, Romania)** MSc in Distributed Systems	1st year Average: *10.00*
2007 – 2011	**Technical University of Cluj-Napoca (Romania)** Faculty of Automation and Computer Science BSc in Computer Science - English Medium, *4 years of studies*	Admission: *9.56 / 10.00* Graduation: *9.17* Bachelor Thesis: *10.00*
2003 – 2007	***Emil Racoviță* High-School (Cluj-Napoca, Romania)** Mathematics and Computer Science Profile	Graduation: *9.51* Baccalaureate: *9.67*
2007	**Ministry of Education, Romania** Certificate in Computer Use and Programming (C/C++ language)	Mark: *10.00*

Voluntary Activities

2010 – 2011 **Team Lead** of the *Microsoft Student Partners* team, TUCN

2009 – 2010 **Marketing Coordinator** of *the Microsoft Student Partners* team, TUCN

2008 – 2011 **Member** of the *Microsoft Student Partners* team, TUCN

2008 – 2011 **Member** of the Representatives Council, *Faculty of Automation and Computer Science*, TUCN

Other Projects/Presentations/Workshops

Jul. 2015 **Microsoft OneWeek Hackathon – ILostYouFound – 1st Prize – *Copenhagen***
Member of a team of 4 – prototyped a collaborative cross-platform mobile app for automatically matching lost items and reported found items based on several factors.
Technologies: Xamarin, C#, iOS, WP8, Maps, Project Oxford

Nov. 2014 **Conference: NAV TechDays – Antwerp**
Advanced session: Office 365 Integration – Focus on Simplicity
Held an in-depth technical demo about the simplified integration story of the Dynamics NAV ERP with the Office and Office 365 Services.

Oct. 2010 – May 2011 **ImagineCup - Software Development Competition: AccessiGaze Studio – *Cluj-Napoca***
Participated in a team of four with an application platform based on Alternative Computer Human Interaction – AccessiGaze Studio. This was also my Bachelor's thesis.
- *Outcome: 1st place University-wide, participation at the national round*
- *Responsibilities: Team Lead, Developer*
- *Technologies: WPF, OpenCV, Microsoft Speech API*

Dec. 2010 **Microsoft Student Partners-TUCN - Silverlight Workshop**
Delivered a four-hour workshop which focused on the distributed side of Silverlight applications and Silverlight browser integration.
Responsibility: Coordinator, Trainer

May 2010 **Microsoft Student Partners-TUCN - Academic Tour**
Organized an event for evangelizing the latest Microsoft technologies and their strong features to the audience - from high-school and university students to company employees.
Responsibility: Coordinator, Speaker (on Visual Studio 2010 and Office 2010)

Nov. 2010, Nov. 2009 **Microsoft Student Partners-TUCN - Using Microsoft Academic Program**
Organized two events whose main purpose was to familiarize the students with the Microsoft Student Partners program and the Microsoft technologies.
Responsibility: Coordinator, Speaker

Jul. 2008, Jul. 2009 **Microsoft Student Partners-TUCN – .NET Summer Rally 2008, 2009**
Held one- and two-week workshops on Object Oriented Programming using C#.
Responsibility: Coordinator, Trainer

Software Development Skills

Software Development	Object Oriented Analysis and Design, Design Patterns, Architectural Patterns, UML	
Programming, Scripting Languages	***Advanced***	*C#, Python*
	Upper Intermediate	*JavaScript (JQuery, KO, Angular), HTML, CSS, PowerShell*
	Intermediate	*Java, C, R, SQL*
	Beginner	*C++, LISP, Assembly, VHDL, ActionScript, Mathematica*
Technologies / Frameworks	Windows Forms, WPF, ASP.NET [MVC], Azure, Silverlight, WIF, PyTorch, Keras	
Development Tools	Visual Studio, Eclipse, NetBeans, PyCharm, IntelliJ Idea, TFS, SVN, Git, PowerShell ISE, Adobe Flash	

Languages Spoken

Romanian	Mother tongue
English	Advanced Level – 14 years of study (Cambridge CAE – Dec. 2006)
German	Intermediate Level – 4 years of study (currently studying)
French	Lower intermediate Level – 6 years of study
Danish	Beginner level – 1 year of study

Hobbies and Other Skills

Basketball, Tennis, Snowboarding, Mountain biking, Trekking, Singing, Dancing.
I love mountains, nature and the beauty within it.

Sworn Declaration

I hereby declare under oath that the submitted Master's Thesis has been written solely by me without any third-party assistance, information other than provided sources or aids have not been used and those used have been fully documented. Sources for literal, paraphrased and cited quotes have been accurately credited.

The submitted document here present is identical to the electronically submitted text document.

Linz, July 2018

Vlad-Ilarie Precup