DEGREE PROJECT

Clustering server properties and syntactic structures in state machines for hyperscale data center operations

Johan Jatko

Computer Science and Engineering, master's level (120 credits)
2021

Luleå University of Technology
Department of Computer Science, Electrical and Space Engineering

LULEÅ UNIVERSITY OF TECHNOLOGY

Abstract

In hyperscale data center operations, automation is applied in many ways as it is becomes very hard to scale otherwise. There are however areas relating to understanding, grouping and diagnosing of error reports that are done manually at Facebook today. This master's thesis investigates solutions for applying unsupervised clustering methods to server error reports, server properties and historical data to speed up and enhance the process of finding and root causing systematic issues. By utilizing data representations that can embed both key-value data and historical event log data, the thesis shows that clustering algorithms together with data representations that capture syntactic and semantic structures in the data can be applied with good results in a real-world scenario.

Acknowledgements

First of all, I would like to thank Facebook and the Data Center Production Operations team in Luleå, Sweden for allowing me to write my master's thesis as part of my employment there. Special thanks to my manager Johan Larsson that has enabled me with this opportunity and supported me throughout the years.
Secondly, I would like to thank Dr. John Stamford who works as a Data Center Engineering Specialist at Facebook, and has served as an external supervisor and helped me shape and for the master's thesis, and has helped unblock me in situations where I have felt stuck.
Thirdly, I would like to thank Professor Evgeny Osipov at Luleå university of technology for being both my thesis supervisor as well as lecturer in several of the courses I have taken during my years at the university. If it wasn't for the course in Applied Artificial Intelligence I would have never gotten an interest in non-conventional AI and ML methods that are now part of this master's thesis.
Lastly, I would like to thank friends, coworkers, family, and my dog Aslat for supporting (or just putting up with) me both during my master's thesis but overall during my studies at the university.

Contents

1 Introduction **1**

1.1 Background 1
1.2 Problem Description 2
1.3 Challenges 3
1.4 Scope of work and delimitations 4
1.5 Thesis structure 5
1.6 Contributions 5

2 Related Work **7**

3 Theory **9**

3.1 Automatas, Encodings and Languages 9
3.1.1 Finite-state automatas 9
3.1.2 Regular Language 10
3.1.3 n-grams 11
3.1.4 Hypervectors 11
3.1.5 Paragraph Vectors with Doc2Vec 12
3.2 Clustering 13
3.2.1 Similarity 13
3.2.2 Hierarchical Agglomerative Clustering 14
3.2.3 DBSCAN 15
3.2.4 Cluster Visualization 15

4 Methods **17**

4.1 Data Gathering 17
4.2 n-gram testing 19
4.3 Clustering Setup 19
4.3.1 Hierarchical Agglomerative Clustering 19
4.3.2 DBSCAN 21
4.3.3 Set representation, Jaccard Similarity 21
4.3.4 Hypervector representation, Cosine Similarity 22
4.3.5 Binary Hypervector representation, Normalized Hamming Distance 22
4.3.6 Doc2Vec representation, Cosine Similarity 23

4.4 Evaluation 24
4.4.1 Clusters 25
4.4.2 FB Accuracy 25
4.4.3 Silhouette Score 25

5 Results **27**
5.1 Impact of n-grams 27
5.2 Clustering results 31
5.3 Cluster Contents 35

6 Discussion **41**
6.1 Results Evaluation 41
6.2 Research Questions 42
6.3 Future Work 44

7 Conclusion **45**

Chapter 1

Introduction

1.1 Background

Hyperscale Data Centers are becoming more prominent in our day and age[1]. Large IT and tech companies run data center fleets on the level of 'hyperscale' with geographical regions across multiple continents housing millions of servers to be used either for providing online services to users, or as part of Infrastructure-as-a-Service offerings to other tech companies. Always online is a key requirement for these data centers as they can serve customers from all around the globe, and process enormous amounts of data each waking second. Uptime is maintained by having large operations teams working around the clock with all components of a data center including cooling, power, networking and servers.

Due to the size of these data centers, the server-to-tech ratio (how many servers each technician can be considered responsible for) can often be 10 to 100 times higher than a traditional enterprise data center[2]. One solution to this is to utilize automation to assist humans during operations, such as automatically applying remediations for known problems[3], providing context and feedback during hands-on work, or by deduplicating and aggregating common and similar issues into bigger tasks for a person to handle.

At Facebook, when a server is considered unhealthy by monitoring systems, it is automatically taken out of production [4]. It then embarks on a journey through multiple autonomous systems that attempts to repair a server, as shown in figure 1.1. If all automated measures fail a manual repair ticket/error report is created. If the issue at hand is known and can be correlated to a faulty part, it is automatically diagnosed and sent to the data center where the server resides for local technicians to swap out the faulty part. However, if an automatic diagnosis cannot be determined for the server, it ends up in a triage queue where engineers manually sift through error reports for the servers and attempt to figure out the root cause.

Figure 1.1: High-level overview of current repair flow at Facebook

1.2 Problem Description

Because these large data centers are for the most part rather homogeneous thanks to common and sometimes open source[5] standards for both building, network, and server design, opportunities exist for treating otherwise one-off problems as distinct groups of issues between multiple data halls, buildings, and even geographical regions, and apply the same remediation for similar error reports at the same time. This thesis will investigate methods to assist with grouping error reports on servers from different data centers, regions and models, as clusters of error reports that make sense for engineers to handle as distinct issues, to create a repair flow as depicted in figure 1.2. Today some of these efforts are done manually by several people analyzing error reports and grouping them together as figure 1.1 depicts, but this is very time consuming and only a small subset of the available data is used as the process becomes too overwhelming otherwise. Distinct issues are identified as a set of error reports that have the same diagnosed outcome that resolved the underlying issue, which is verified by the server going back into production without any alarms from monitoring systems.

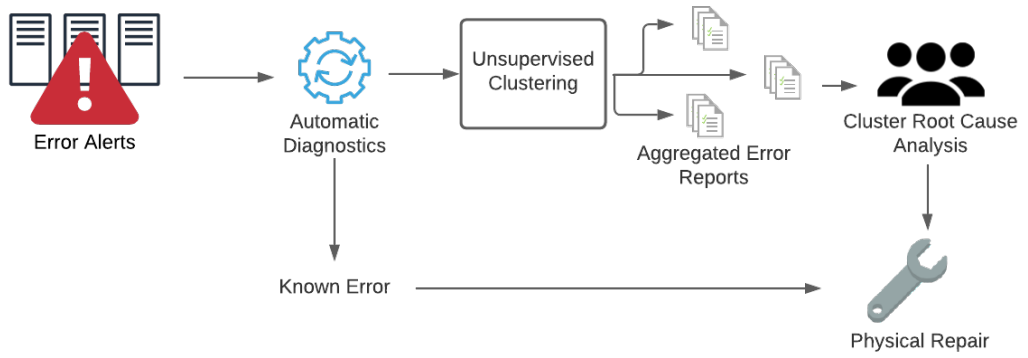

Figure 1.2: High-level overview of an improved repair flow using automatic clustering, allowing engineers to focus time on examining clusters and root cause analysis

To properly represent how an engineer would manually group error reports, both metadata for the server and issue at hand as well as historical data in the form of state transitions need to be considered when evaluating each error report. This provides a challenge as data with different syntactic structures and values needs to be combined and clustered on together, whilst still preserving the syntactic structure.

This thesis will attempt to answer the following questions:

- **Q1**: How well can common clustering methods be used to cluster server error reports in an unsupervised manner?
- **Q2**: Can you cluster on key-value data and state transition data together effectively?
- **Q3**: How does different data representations affect the ability to build good clusters?

The problem can be broken down into a few sub-problems:

- Decide which data to use as part of evaluation
- Evaluate clustering methods available and suitable for the problem at hand
- Evaluate different data encodings and preprocessing to use as representation
- Evaluate and tune parameters and measure their effect on results
- Present resulting clusters in a suitable way to be digested by engineers

For this thesis, the data used will be a combination of metadata for the server such as data center, make and model, manufacturing date, firmware versions etc, as well as a log of the different states and transitions the server has been in since it arrived at the data center. The states include data such as if the server is in repair, its error report with an error signal, or any remediations taken on the server. Exactly which values are included can vary slightly and are mostly up to implementation specifics. The data presentation should result in clusters being represented as a list of error report ids that an engineer can look up in a list. Visual grouping of results can be done but should not be considered as a deliverable.

1.3 Challenges

The main challenge will be with the data used, as the outcome/result of an error report can be a bit different even with identical input information. This is mainly due to how engineers interpret errors when they are debugging problems with varying degrees of

error information, and may come to two different conclusions. This becomes a little more complex also as some diagnoses/outcomes may involve another outcome as well. An example is the diagnosis 'Reseat Server', which tells a technician to pull out a server and make it powered off for a couple of seconds before plugging it in again. This allows certain always-on components such as Baseboard Management Controller and Network Interface Card to properly reinitialize its firmware and clear out potential lockouts. However, if someone would go ahead and diagnose 'Replace Network Interface Card' instead, it would also require them to effectively 'Reseat Server' as the server has to be removed from the rack when replacing components. Keeping this in mind, one can expect noise in the data that can affect the efficiency factor for each algorithm used.

Another challenge that relates to the data is how one can properly verify that the clusters and metrics generated by a method is actually of value, as the selection of metrics may not describe the full story for the data at hand.

Because the work carried out is partly evaluated using Facebook Production data, everything needs to run inside Facebook's infrastructure and results will be generalized and anonymized if needed.

1.4 Scope of work and delimitations

To ensure that the scope of the thesis does not expand too far, a number of delimitations have been set to focus efforts on a particular type of problem within the space. These are numbered so they can be easily referenced throughout the thesis.

- **D1**: As there are many aspects when considering what is a good cluster of error reports for engineers, this thesis will work with the assumption that a good cluster is only defined by the diagnosed outcome of all error reports.

- **D2**: How to acquire the data will not be part of the thesis as this will vary depending on how and on what you want to apply the algorithms.

- **D3**: The evaluation part of the thesis will not take into account resource consumption or actual execution time by any of the algorithms, but may comment on notable differences between them. Additionally the input size of the algorithms will be constant or near-constant across all tests, so whilst complexity for different algorithms will be mentioned it will not be considered a deal-breaker.

- **D4**: As the thesis will focus on finding large systematic issues, we will filter out any clusters that contain less than 50 objects. This value corresponds to a rough estimate of the volume an engineer would look for to start considering something being a systematic issue.

1.5 Thesis structure

The master's thesis is divided into 7 chapters, starting with chapter 1 which contains the background for the thesis, the problem description and some challenges and delimitations. In chapter 2 we explore related work within the fields of using machine learning and clustering in data center operations, as well as the general areas of clustering on different types of data structures. In chapter 3 theory for the different concepts used in the thesis are presented and motivated. Chapter 4 describes the methods used to gather data, setup and carry out experiments as well as how the experiments were evaluated. In chapter 5 the results from the experiments are presented, together with some analysis. The results are further examined and evaluated in chapter 6 where both the methods used and the results are discussed. The research questions are also evaluated here. Finally, in chapter 7 a conclusion for the master's thesis is given, and improvements and future work are presented.

1.6 Contributions

The results from this thesis can be used to enhance ML models used at Facebook to predict diagnoses for otherwise undiagnosed error reports. By building clusters that get a common diagnosed outcome, the ML models get fed higher quality data that allows them to take actions without having engineers manually examine error reports.

The results also allows time that was dedicated to manually find and identify clusters to shift focus on deep-diving into aggregated issues across multiple regions, data centers, and hardware instead.

Lastly, the thesis as a whole contributes to the area of hyperscale data center operations which as a whole does not have that many publications to date, and can hopefully promote further research into this area.

Chapter 2

Related Work

As the area of hyperscale data center operations is fairly new and modern, many companies are in the early stages of experimenting with and applying machine learning and AI solutions [6] to tackle problems on an unprecedented scale. There is also a limited number of publications from academia due to the vast environments that define the area of hyperscale data center operations being hard to replicate.

As described in [4], Facebook has multiple systems at work today to tackle the ever-changing environment in their large-scale data centers. One of the main goals of both the mentioned research and this thesis is to reduce the need for manual debugging of data center equipment, as the scale that is operated on would require an enormous amount manpower if all work was done manually by engineers.

In the articles [3] and [7] a machine learning model is deployed on a group of error reports called 'undiagnosed', namely error reports where a diagnosis that would fix the issue could not be derived, and manual diagnosis would have to be carried out by engineers. The idea is to have engineers root cause undiagnosed problems, and then utilize the results from that effort to fit an ML model based on input data such as failure logs and metadata to automatically attempt certain diagnoses for otherwise undiagnosed issues. Whilst tackling the same problem, such a model can only be effective for issues that have already existed as it needs that training data, and the point of this thesis is to work on the group of errors without prior knowledge. However, one outcome from this thesis could be to provide better and more accurate training data for that model, as the hope is that clusters of error reports will end up having the same diagnosed outcome.

In [8], a framework for dimensional analysis on the same type of error reports is proposed based on structured logs that somewhat resemble the data that will be used in this thesis. They do however only use the last few days of data and treat it as a time series problem, whereas this thesis will focus on the full lifetime of events of each server and how certain events can reoccur during a server's lifetime. The results from running the dimensional analysis can resemble clustering in the sense that combinations of features are evaluated

to how many error reports distinctly match those features while they don't match others.

Looking at the clustering of logs in a data center setting, [9] uses DBSCAN [10] and Levenshtein distance [11] to find unique syntactic structures of different log messages. They then compute a regular expression for each cluster that can parse every message in each cluster, effectively storing the syntactic structure of each cluster. This is similar in that we want to utilize the syntactic structure in logs, but it differs in that we want to use the syntactic structure between log messages, while they do it only within each log message. Additionally the research was conducted in an environment with around 100-200 servers, which is not representable of a hyperscale data center environment. To my knowledge there exists no prior research on clustering on syntactic structure between log messages.

The same article also implements Hierarchical Agglomerative Clustering together with Jaccard Distance to cluster problems that existed within the same slots. There are similarities between comparing which timeslots different items were together in, and when items were in the same state, as both are a form of temporal information.

Examining ways of mixing different types of structured data, there are plenty of solutions that simply concatenate vectors of different types [3] and use that as input. However, that is not as straight forward when dealing with similarity and distance metrics as the different vector dimensions in concatenated vectors will contain different scales and putting that through a standard metric would produce skewed results. This problem is tackled in [12] by leveraging the concept of "Hybrid clustering", where the similarity between items with two or more different types of structured data is weighted for each piece of data, and then combined as a final similarity metric. This is a potential solution that could be used in this thesis, but this thesis will also explore solutions that do not a hybrid clustering approach.

Overall, this thesis will build on existing ideas such as clustering on syntactic structures and clustering on mixed data types, but will focus on expanding those to new areas including syntactical structures between logs from state transitions, and clustering on error reports in a hyperscale data center operations environment.

Chapter 3

Theory

This chapter gives an introduction and some information regarding the methods used in the thesis, as well as some background for why some methods can be used on the problem we have at hand. In the first part we look into the features and properties of automatas, as one of the main challenges of the thesis is to combine event history data with regular key-value data. We then look at appropriate encodings, and lastly methods for clustering the data and measuring the result.

In chapter 2, when looking at existing ways of handling syntactic structure the subject of regular expressions came up as a way to represent such structure within text. As is mentioned, this thesis is looking to work with syntactic structures between individual log messages rather than inside log messages. However, because the event history is primarily records of a server moving between states, an idea to model the data as a finite-state automata and use the relationship between finite-state automatas and regular grammars and expressions for embedding this syntactic structure was proposed.

3.1 Automatas, Encodings and Languages

3.1.1 Finite-state automatas

A finite-state automata, also known as finite-state machine or FSA/FSM, is a mathematical model based on states and inputs within the theory of automatas [13][14]. An FSA has a finite number of states that it can move between based on input received. The change between two states is called a transition.

Many behaviours of modern everyday devices can be modelled using finite-state automatas, as they can be described by a finite set of states, and take input from either humans or other devices. Automatic doors and coffee machines are systems where user-defined input drives transitions between two or more states.

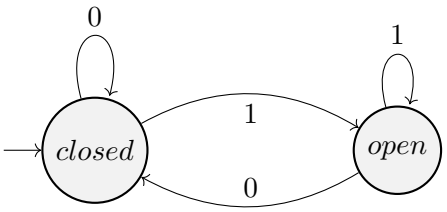

Figure 3.1: FSA representation of an automatic door, with input representing the sensor where 0 = no person sensed and 1 = person sensed.

3.1.2 Regular Language

In formal language theory, a regular language is the most basic family of languages and can be identified as a language that can be accepted by a finite automation. The definition of a language is "a set (finite or infinite) of sentences, each of finite length, all constructed from a finite alphabet of symbols" [15]. Furthermore the grammar of a language is a set of rules and constraints that can generate all of the strings that are sentences of a language and only those.

As an example of the relationship between regular languages, grammars and finite-state automatas we can define an example regular grammar $G = (N, \Sigma, P, S)$ where $A, B, S \in N$ are non-terminal symbols, $a, b, c, d \in \Sigma$ as terminal symbols, S as the start symbol, ε as the empty string, and P as the production rules

$$S \rightarrow aS$$
$$S \rightarrow bA$$
$$A \rightarrow cA$$
$$A \rightarrow dB$$
$$B \rightarrow \varepsilon.$$

This grammar will generate sentences according to the regular expression a*bc*d, such as aaaaaabcccd, abccccccccccd, and bcd. In figure 3.2 we have an automation that validate any sentence for the example grammar.

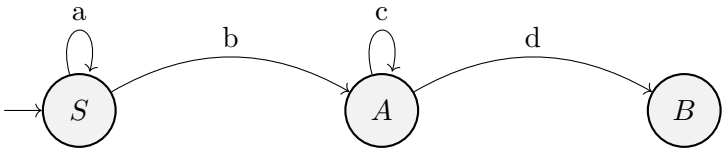

Figure 3.2: FSA for validating the example grammar.

By observing a finite automata and recording the state transitions for the input string aabcccd, we get the log

```
S a S a S b A c A c A c A d B
```

which correspond to the input sentence being valid, and the log consisting of the valid input sentence including the non-terminals (states) that were visited. The grammatical properties from the input string are also kept in the observation log, as long as you look over the states.

Because of this connection, any sentence that can be parsed and accepted by any finite-state automata will by definition be a part of a regular language and will have the grammatical properties of a regular grammar [13][16]. This also means that any grammatical properties of any part of such a sentence are represented in the logs created from observing a finite-state automata. Lastly because an finite-state automata determines next state only based on current state and input [13], we can consider that the grammatical properties of regular languages are negligible over longer distances. This is a property that we will use to our advantage when selecting a data representation.

3.1.3 n-grams

N-grams is a common data representation within the field of computational linguistics [17], and represent a contiguous sequence of n items from a sample of text while embedding context for each item [18]. It can however be used for sequences of items of almost any data type to model dependencies between items. It finds applications within areas such as statistical natural language processing [17], computational biology [18], and data compression to name a few. In language identification, sequences of characters from text are used to capture language specific patterns, and in biology DNA base pairs are used as the unit.

As the amount of context is bounded by the value of n, n-grams are not suited for modelling long range dependencies [19] of more complex grammatical properties that exist in our everyday languages such as english. However, they can be used for text generated by regular grammars as those grammatical properties are limited to the immediately surrounding words.

3.1.4 Hypervectors

Hypervectors are high-dimensional randomized vectors that can be used to encode and represent features for tasks like clustering, classification and analogical mapping [20]. They consist of vectors with $d = 10000$ or similar, where each coordinate is initialized by a random number, binary in the form of (0, 1) or (-1, 1). Every vector representing a feature is randomized to be as unique from any other vector in the vector map/dictionary .

The key features of hypervector algebra includes three operations commonly referred to as bundling ($+$), binding ($\circ$) and permutation ρ. The implementation of these operations

depends on which type of hypervector representation is used. When woring with bipolar hypervectors, bundling is performed as coordinate by coordinate addition, binding is performed as coordinate by coordinate multiplication, and permutation is performed by shifting the coordinate positions by ρ to the left/right.

When working with strictly (0, 1)-binary hypervectors, as can be the case in embedded systems, the methods are different. Bundling of vectors is performed by calculating the sum for each dimension of the vectors, and setting each dimension to a 1 only if the sum reaches a threshold of $n/2$ where n is the number of vectors being bundled. For binding between vectors, XOR is used instead.

These operations can be used to represent other data structures as hypervectors. Given the features a, b, c, d, s, x, y, z and their upper-case vector representations A, B, C, D, S, X, Y, Z, the following representations can be made:

Set: A set of values $s = \{a, b, c\}$ can be encoded as:

$$S = A + B + C$$

Dictionary/Map: A map of key-value pairs $d = (x = a)\&(y = b)\&(z = c)$ can be encoded as:

$$D = X \circ A + Y \circ B + Z \circ C$$

Sequence: A sequence of values a, b can be encoded using rotation:

$$AB = \rho A \circ B$$

Appending another element c to a,b is encoded as:

$$\begin{aligned} ABC =& \rho(AB) \circ C \\ =& \rho^2 A \circ \rho B \circ C \end{aligned}$$

As a form of random indexing, they can be and have been used to represent and cluster n-gram data with good results [20], and since many different data structures can be encoded by the same vectors and then be bound together, hypervectors are a good candidate data representation for combining different types of data.

3.1.5 Paragraph Vectors with Doc2Vec

Word embeddings have become a popular way of quickly building an understanding of how different words have relations to each other in enormous datasets. Word2Vec [21] is one of the most popular models that can produce vector representations of words in texts, where the syntactic and semantic similarity between words is represented in the vectors that the model produces for each word. Word2Vec is implemented as a a neural

network that takes sentences as input, and learns about connections to other words in the corpus to produce fixed-size output vectors based on the context of surrounding words in sentences or just by association of which words exist in the input sentences. To get the similarity between two words, the cosine distance can be calculated between the word embeddings.

A further development to the Word2Vec model is the so called Doc2Vec model [22], which instead of just words works on full paragraphs or documents to produce vectors that embed similarity traits between documents. This allows for more complex operations such as document clustering as the resulting vectors are still compared only using the cosine distance metric.

One caution with both models is that they are generally better suited for very large corpuses with millions of words.

3.2 Clustering

The basic idea of clustering is finding items within a dataset that are in some way considered to be similar to each other[11].

One of the main things to consider when selecting clustering algorithms for this thesis was finding ones that could work with arbitrary pairwise distance/similarity metrics, as the chosen data representations above have several different metrics. As the idea for the thesis was to find an arbitrary amount of clusters the need for algorithms that did not need a predefined amount of clusters beforehand had to be taken into account.

3.2.1 Similarity

Clustering requires a metric that tries to describe how appropriate it is for two elements to be a part of the same cluster. This will generally be some kind of similarity or distance metric that is calculated between all elements[11]. Similarity and Distance can, as long as they are normalized, be considered each other's opposite metric. Two identical objects would have a similarity of 1.0 and a normalized distance of 0.0, while two fully dissimilar objects would have a similarity of 0.0 and a normalized distance of 1.0.

For this thesis we will look at three different similarity/distance metrics that are selected based on the data representations that will be used.

Jaccard Similarity

A way to calculate the similarity between two sets is to calculate the Jaccard Coefficient or Jaccard Similarity between them [11]. Jaccard Coefficient is a similarity metric between 0.0 and 1.0 (where 1.0 is fully similar) that is calculated as

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}.$$

Cosine Similarity

To compare similarity between vectors one can use cosine similarity, which is a measure of cosine of the angle between two vectors[23].By doing this, the measurement does not take magnitude of vectors into account, only which direction they are pointing to. This functionality gives the vectors the ability to strengthen a feature representation by multiplying a vector with a scalar or adding identical vectors element-wise multiple times as it won't change the resulting angle, only shifting the individual dimensions further away from origo making the angle harder to affect. Both hypervectors [20] and word embeddings from the Word2Vec [21] and Doc2Vec [22] methods can be compared by using cosine similarity.

Hamming Distance

The hamming distance $H(x, y)$ between two vectors x and y is defined to be the number of positions i such that $x_i \neq y_i$ [24]. This metric is mostly applicable when working with (0, 1)-binary vector representations. In this thesis we use the normalized hamming distance, which is the hamming distance divided by the number of dimensions in the vector. This ends up producing a number between 0.0 and 1.0.

Another way of expressing the Normalized Hamming distance of two vectors x and y with d dimensions is

$$H_{norm}(x,y) = \frac{\sum_{i=0}^{d} x_i \oplus y_i}{d}.$$

3.2.2 Hierarchical Agglomerative Clustering

Hierarchical Cluster is a method of cluster analysis where the idea is to build a hierarchy of clusters [23]. Agglomerative clustering is the so called "bottom-up" approach where each item starts as its own cluster, and pairs of clusters are merged as the hierarchy is built up.

The time complexity for hierarchical agglomerative clustering is atleast $O(n^2)$ [10]. It will also use huge amounts of I/O when clustering over a large number of objects. This can cause it to scale badly if the input size increases, but as the thesis uses a

near-constant input size time complexity is not factored into the selection process or evaluation.

When using hierarchical clustering, there are a couple of methods that determine how clusters in the hierarchy should be evaluated when determining if they should be merged or not [10]. **Single-link clustering** consider the distance between two clusters to be the shortest distance from any member of one cluster to any member of the other cluster. **Average-link clustering**, as the name advertises, uses the average distance between all members of one cluster and the other cluster. **Complete-link clustering** can be considered the opposite of single-link clustering, and will use the longest distance between any member of one cluster to any member of the other cluster.

In a situation where it is extra important that the cluster only contains items that definitely should belong there, Complete-link clustering would provide the best fit as it examines the clusters in a 'worst-case scenario'. This should help reduce false-positives, but will increase the overall amount of clusters being returned.

3.2.3 DBSCAN

DBSCAN, short for Density-Based Spatial Clustering of Applications with Noise, is a clustering algorithm that discovers clusters of any shape by examining the neighborhood of each object and checking if it contains more objects than a defined minimum threshold [10]. One particular feature of DBSCAN is that it can classify items as either being part of a cluster, or just being noise. This is different from many other clustering methods that attempt to force every item in a dataset into a suitable cluster (or give it a separate cluster for itself).

3.2.4 Cluster Visualization

To get a better grasp of how data that has up to 10000 dimensions get properly clustered, the t-SNE method can be used to represent high dimensional data as 2D-data in a scatter plot. t-SNE, or t-distributed Stochastic Neighbour Embeddings, is a technique for dimensionality reduction that still tries to keep the overall features from each dimension [25].

A common way of applying this when evaluating clustering results is by using t-SNE to get 2D coordinates from the input data and draw them in a scatter plot which causes similar items to get roughly the same coordinates. The points are then also colored with a unique color for the cluster that they were associated with. The resulting scatter plot gives an indication of if your clustering algorithm is building clusters for items that are seemingly similar. As an example, in figure 3.3 the MNIST dataset, a large labeled collection of images of handwritten numbers between 0 and 9, has had its raw

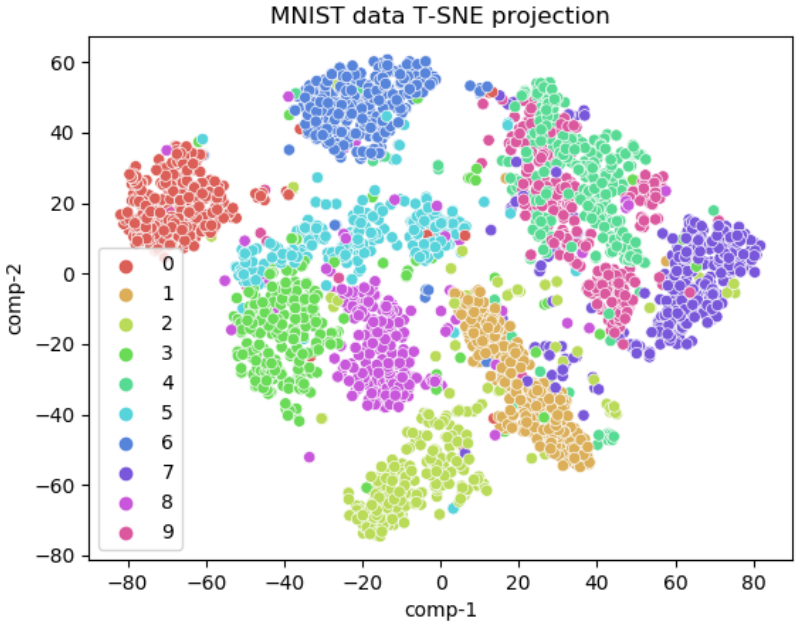

Figure 3.3: Scatter plot of running the MNIST dataset through t-SNE.

pixel values from each image fed directly to the t-SNE algorithm as large vectors. The algorithm has downscaled it from over 700 dimensions to just a 2D grid. Each 2D point has then been painted on a scatter plot with a color representing the label it corresponds to. The resulting graph shows that a majority of the handwritten numbers get 2D coordinates fairly close to each other which is exactly what t-SNE models. More in-depth information on how t-SNE works can be found in [25].

Chapter 4

Methods

The methods and their implementations were loosely split up to address the different sub-problems that were presented in chapter 1. For the first step the neccesary data is gathered and pre-processed. Afterwards an initial test to assess the impact of n-gram representations. With knowledge from that test, the other data representations are tuned and tested towards all datasets, and certain evaluation metrics are collected to be presented in chapter 5. A high-level overview of the whole process is depicted in figure 4.1.

4.1 Data Gathering

The data used to represent prior and current state for servers and error reports was queried from a data warehouse using a query engine called PrestoSQL [26]. This allows you to make queries across many different data sources that have different storage backends, such as doing SQL joins between Apache Hive and MySQL clusters.

To decide which servers to look at, error reports for servers from around 5 different dates were chosen. Each dataset was formed to contain around 10000 servers, but can have very different composition of clusters. Further queries were then made for these servers to get historical error reports. Some baseline metrics for the datasets are presented in table 4.1. These metrics were calculated by counting the unique amount of error signals present in each dataset. Since this does not take into account different models, firmware versions, or anything else that might contribute to data being clustered, any computed clusters cannot directly be compared to this ground truth. However, it serves as a good enough baseline and it can be expected that the algorithms should find atleast a subset of these clusters.

These datasets were sampled from very spread out dates throughout the year, as an attempt to mitigate 'overfitting' clustering parameters to a particular dataset.

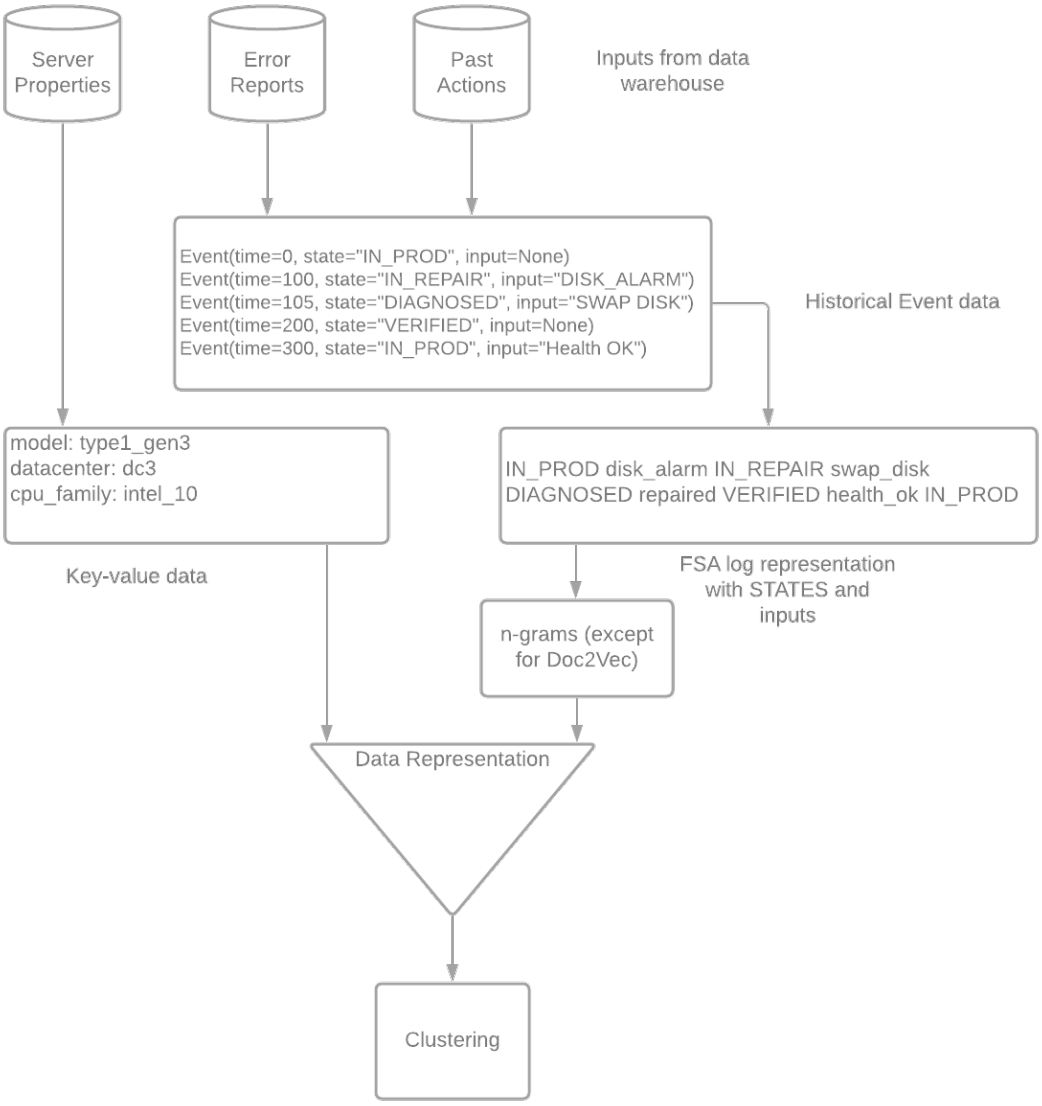

Figure 4.1: High-level overview of converting input data to desired data representations.

The resulting data contains things like current and prior error reports (including signals about the failures), server model, CPU architecture, workload running on the server, which datacenter it is installed in, as well as historical and current repairs done. The data was divided up into key-value pairs for metadata, and as state/state transitions for event history data.

All data was downloaded to a Jupyter Notebook [27] to be further processed using pandas [28], a library for handling tabular data. The data was then sorted based on timestamps to build an event timeline of each server from the day it entered a data center up until a set date for which the analysis was made on. Each event corresponds to a definite state the server was in at a point in time, with the inputs that made it end up in that state.

The event history for each server is then modelled as the finite state automata depicted

Table 4.1: Statistics for the five datasets used.

Name	Size	Known Clusters	Avg Cluster Size
Dataset 1	∼10000	14	165.64
Dataset 2	∼10000	19	200.95
Dataset 3	∼10000	22	201.18
Dataset 4	∼10000	17	185
Dataset 5	∼10000	21	220

in figure 4.2, by rebuilding the FSA log from the event timeline. The only difference is that inputs such as bad_part, error_signal, investigate contain more detailed information of exactly which bad part, error signal or escalation reason was used to move it into the new state. This is not detailed in the finite-state automata model, but is important metadata for the actual clustering step to provide more accurate results. The data was then further processed into the desired data representations described in sections 4.3.3-4.3.6.

4.2 n-gram testing

The first theory to test was that n-grams can and will improve clustering of FSA logs due to their relationship to regular grammars that was described in section 3.1.2. An initial test bench was setup via the "Hypervector representation, Cosine Similarity" approach described in section 4.3.4.

The test was then ran on one dataset, only changing the n value when building the n-grams. The results for each run was recorded using the evaluation metrics described in section 4.4.

4.3 Clustering Setup

4.3.1 Hierarchical Agglomerative Clustering

For HAC the scikit-learn [29] implementation was used. It could either take a distance matrix as input, or vectors together with a distance method that would output a distance matrix. The implementation has several parameters that allows it to be used for several different problems and inputs. For this thesis the following parameters were involved, and the rest were set to default:

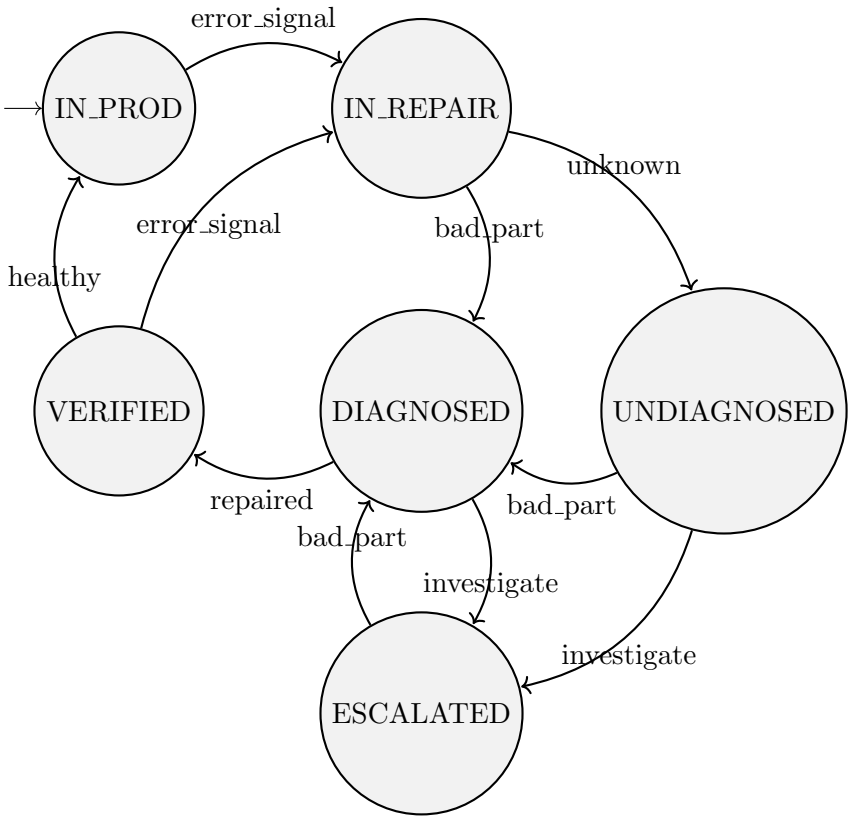

Figure 4.2: FSA representation of the state of a server's lifecycle.

Parameter	Type	Description
n_clusters	int or None	The number of clusters to find. If None then distance_threshold is used instead, which is what is used here.
affinity	str	Metric used to calculate linkage or similarity. Values used were 'precomputed' or 'cosine'.
linkage	str	Which linkage criteria to use. Only 'complete' was used.
distance_threshold	float	The distance threshold above which, clusters will not be merged.

The main tunables of the algorithm are either specifying the number of clusters to find with n_clusters, or specifying a distance threshold which served as a limit between when clusters are too far away from each other and should not be merged. For these tests n_clusters was set to None so that the algorithm would find an unknown amount of clusters only based on distance_threshold as a variable parameter, as the goal is to find both known and new clusters. In further tests and results distance_threshold is denoted as the t parameter.

4.3.2 DBSCAN

scikit-learn [29] was also the source for the DBSCAN implementation. Using the same library for both had the added benefit that they shared the same API and could be switched interchangably without changing much of the surrounding code. In this case the algorithm also accepted either take a distance matrix as input, or vectors together with a distance method that would output a distance matrix. Similar to the HAC implementation, DBSCAN has multiple parameters but only a few listed ones were changed from their default values:

Parameter	Type	Description
eps	float	The maximum distance between two samples for them to be considered in the same neighborhood.
min_samples	int	The number of samples in a neighborhood for a point to be considered as a core point. The default value of 5 was used.
metric	str	Metric used to calculate similarity. Values used were 'precomputed' or 'cosine'.
n_jobs	int	While not used in the calculations themselves, specifies how many parallel jobs will be used to calculate the clusters.

DBSCAN was primarily tuned by the *eps* parameter, and as the min_samples parameter was set to 5 many small clusters that HAC produces were considered noise by DBSCAN.

4.3.3 Set representation, Jaccard Similarity

The first method of clustering on similarity was to represent each server's data as a set, and then use Jaccard Similarity to calculate the pair-wise similarity between all servers in the dataset.

To represent the FSA log history as a set, we recorded each state and its input as a long sentence of words. We then built tri-grams of words from the sentence and stored those in a set. Key-value pairs were also stored in the set as "key: value" strings.

```
states = (
    "IN_PROD memory_incorr IN_REPAIR ram_stick DIAGNOSED "
    "repaired VERIFIED healthy IN_PROD"
)
ngrams = set(ngrams(states, n=3))
kv = {"datacenter dc1", "model type1_gen3"}

S = ngrams + kv
```

Figure 4.3: Pseudocode for generating sets from error report data

The next step was to calculate the Jaccard Coefficient between all servers. The results are stored in an nxn matrix called a similarity matrix, where each row/column represents each server in the list of selected servers. Each element e_ij is a value between 0 and 1 representing how similar server i is to server j, with 1 being exactly similar and 0 fully dissimilar.

The resulting matrix is transformed from a similarity matrix A_s to a distance matrix A_d by taking $A_d = 1 - A_s$, causing fully dissimilar pairs that had a value of 0 to get a max distance of 1 instead, and vice versa.

In chapters 5 and 6 any results from this data representation method will be labeled 'Jaccard'.

4.3.4 Hypervector representation, Cosine Similarity

The second method was representing each server's data as a bipolar hypervector, and use cosine similarity to calculate the pair-wise similarity between all servers in the dataset.

The approach was similar to the first method with sets, by recording each state and its input as a sequence. We then built tri-grams from the sequence that were encoded as hypervectors and added together. Key-value pairs were also encoded as hypervectors and added to the resulting vector of each server similar to in figure 4.4.

The next step was, much like the first approach, to build a distance matrix of size nxn. This time the cosine similarity metric was used instead.

In chapters 5 and 6 any results from this data representation method will be labeled 'HDVector'.

4.3.5 Binary Hypervector representation, Normalized Hamming Distance

The third method was representing each server's data as a binary hypervector, and use normalized hamming distance to calculate the pair-wise similarity between all servers in the dataset.

The approach is almost identical to the second method in 4.3.4, with the difference that the operations were for binary (0, 1)-hypervectors instead of bipolar (-1, 1)-hypervectors. This in turn means that binding and permuting of hypervectors are implemented using different operations.

In chapters 5 and 6 any results from this data representation method will be labeled 'HDVector2'.

```
hdmap = {}
def get_vector(data):
  if not data in hdmap:
    hdmap[data] = random_vector(d=10000)
  return hdmap[data]

states = (
    "IN_PROD memory_incorr IN_REPAIR ram_stick DIAGNOSED "
    "repaired VERIFIED healthy IN_PROD"
)
kv = {"datacenter": "dc1", "model": "type1_gen3"}

Vstates = sum([
    get_vector(ngram[0])*get_vector(ngram[1])*get_vector(ngram[2])
    for ngram in ngrams(states, n=3)
])
Vkv = sum([get_vector(k) * get_vector(v) for k,v in kv])
V = Vkv + Vstates
```

Figure 4.4: Pseudocode for generating hypervectors from error report data

4.3.6 Doc2Vec representation, Cosine Similarity

The fourth approach was representing each server's data as an word embedding vector calculated by Doc2Vec. Here cosine similarity was also used for pair-wise similarity.

Because we have no good way of combining the metadata and the state transitions separately without applying something like hybrid clustering [12], they were simply appended together as one long sentence representing each server. Doc2Vec was then initialized and all sentences were fed as the training corpus. Because we are using magnitudes less data than doc2vec would normally train on, the parameters available were tuned to ensure that relationships in the data were somewhat captured in the vectors.

When the model had been trained, each document was given it's paragraph vector, and a distance matrix was built the same way as in section 4.3.4 by calculating the cosine similarity between all vectors.

In chapters 5 and 6 any results from this data representation method will be labeled 'Doc2Vec'.

```
states = (
    "IN_PROD memory_incorr IN_REPAIR ram_stick DIAGNOSED "
    "repaired VERIFIED healthy IN_PROD"
)
kv = ["datacenter dc1", "model type1_gen3"]
document = " ".join(kv) + states

# repeat for each server and build document db
documents = [document ...]

model = Doc2Vec(documents, vector_size=300, epochs=50)

for doc in documents:
    V = model.infer_vector(doc)
```

Figure 4.5: Pseudocode for generating paragraph vectors from error report data

4.4 Evaluation

Each approach was ran on the datasets mentioned in section 4.1 with each data representation. Since the goal is to evaluate methods that in general are viable for this type of data, the contents of the datasets vary to give an accurate representation of what might be seen during a year in a data center fleet. Additionally one parameter was varied for both of the approaches, to see how that affects the results. For HAC the distance threshold t was tested with four different values from 0.2 to 0.75, and for DBSCAN the *eps* parameter was varied between 0.1 to 0.5.

The first analysis is to determine which parameter(s) for each of the algorithms produces results that are worth looking further into. This is done by weighting the aggregated average metrics that are described in the sections below.

Evaluation for each run is done by first filtering out all clusters with less than 50 items in them as per delimitation D4. The rest of the metrics are then computed and stored into a table. When a few combinations have been deemed more successful than others (based on the collective weight of all metrics), the metrics are split up from its aggregate view to show a more detailed view per dataset.

The results from the detailed evaluation will also be compared to the t-SNE visualization graphs that will be produced for the best and worst run.

4.4.1 Clusters

The first metric for a run was how many clusters with more than 50 items were detected in the run. This is not a metric to optimize for alone, but plays a part together with the accuracy of the clusters. If a particular run finds many clusters but also has high accuracy, it could be considered a better method.

One thing we don't take in to account is how many items above 50 that each cluster contains, as you could have 2 clusters with seemingly identical items that should technically be one cluster. We examine this by doing a deep dive of clusters from the most successful run and attempt to explain why seemingly similar clusters have been split into two or more.

4.4.2 FB Accuracy

The accuracy for a cluster, in this thesis called FB Accuracy, was devised from delimitation D1 specified in section 1.4 to provide a metric for how much value a clustering method could potentially bring when running in a real world scenario. It is calculated by looking at the occurrences of diagnosed outcome of each error report, and divide the highest occurrence of a diagnosed outcome with all error reports in the cluster. A score of 1.0 would mean that all error reports in the cluster had the exact same diagnosed outcome, which is the ideal score, but generally only achievable for known errors with automatic diagnoses.

One thing to keep in mind is that the data is composed by both humans and automation, so there exists a natural accuracy loss due to people selecting different solutions for the same problem.

4.4.3 Silhouette Score

Silhouette score is a measure of how how well objects have been clustered[30]. The value measures how similar an object is to its own cluster compared to other clusters. It is defined as

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

where

$a(i)$ = average dissimilarity of i to all other objects in own cluster

and

$b(i)$ = minimum value of average dissimilarity of i to all objects in other clusters.

The resulting value is in the range of $[-1, 1]$ where values towards 1 means that the correct cluster is chosen, values close to 0 mean that the item is on the border between

clusters, and values towards -1 means that the item would be better suited in another cluster.

Because we apply the limitation of only wanting clusters with more than 50 items, two silhouette scores are calculated. The first is for all items and clusters regardless of size, and the other is an "adjusted" silhouette score only involving the selected clusters. In this implementation the silhouette score is also not calculated for the "noise" cluster that the DBSCAN algorithm assigns items it considers to be noise, as it would severely impact the metric negatively. This has to be kept in mind as one compares the results between HAC and DBSCAN.

Chapter 5

Results

In this chapter the results from chapter 4 are presented. First we examine the theory that n-grams can be applied to FSA log data and achieve higher accuracy, then we present the results from all the data representations we tested with our two clustering algorithms selected. Lastly a deep dive into one specific run is made to examine the contents of a few clusters. This chapter will only make some observations regarding the results and some simpler explanations. A deeper evaluation will be done in chapter 6.

5.1 Impact of n-grams

The first test described in section 4.2 was to evaluate how using n-grams on the FSA logs would affect the accuracy of the clustering. The results presented in table 5.1 shows that the measured accuracy goes up when raising n from 1 through 3, but raising it further does not provide any increase in accuracy, it only finds less clusters. The overall silhouette score also stabilizes at $n = 3$, giving an indication that for all data you are unlikely to get any better results. The adjusted silhouette score keeps increasing as you increase n, but this is can be attributed to the number of clusters decreasing at the same time.

Table 5.1: Results for n-gram representations of FSA with respect to clustering.

n-gram	Clusters	FB Accuracy	Sil Score	Adj Sil Score
n=1	50	0.667887	0.322974	0.496032
n=2	32	0.819857	0.314197	0.639761
n=3	26	0.831289	0.348428	0.662777
n=4	23	0.825428	0.347941	0.744172
n=5	18	0.834127	0.348703	0.713653

When observing the t-SNE graphs in figures 5.1-5.5 for each run, one can observe some trends as n is increased. First of all, clusters are more distinct and not as mixed into each other. This can be attributed to the additional information provided by the surrounding context eliminating uncertainties about in which cluster an item actually belongs to, and the additional differences given by the context pushing items further away from each other when using t-SNE. You also have less 'random' stragglers that affect the centroids of each cluster as those become too distant from clusters when the additional features are introduced. Worth noting is the fact that even with $n = 1$, the FB accuracy is still quite decent. This is because each individual keyword is still used on its own during the clustering step, and the data overall has clustering potential that will be further evaluated by the tests described in 4.3.

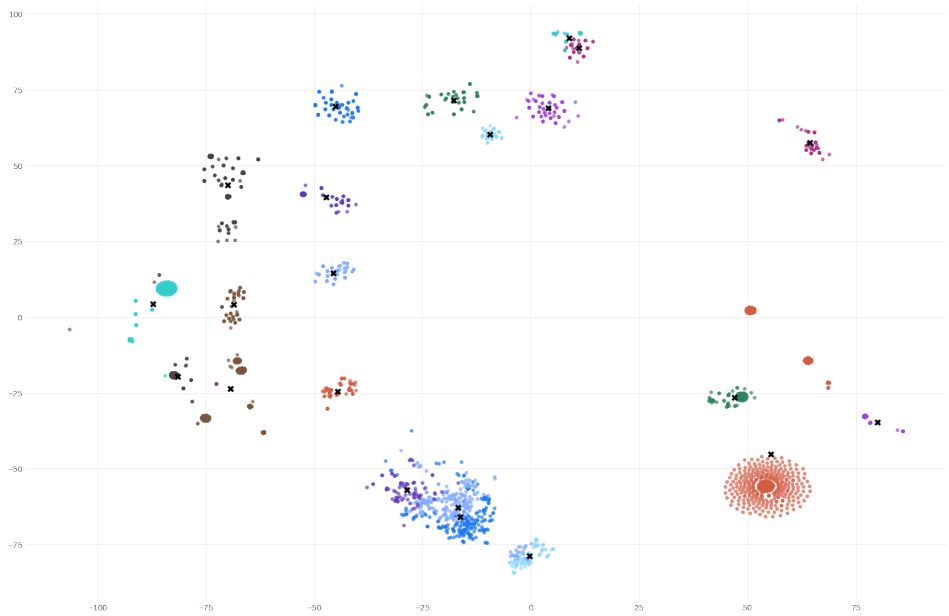

Figure 5.1: HDVector n-grams $n = 1$

Figure 5.2: HDVector n-grams $n = 2$

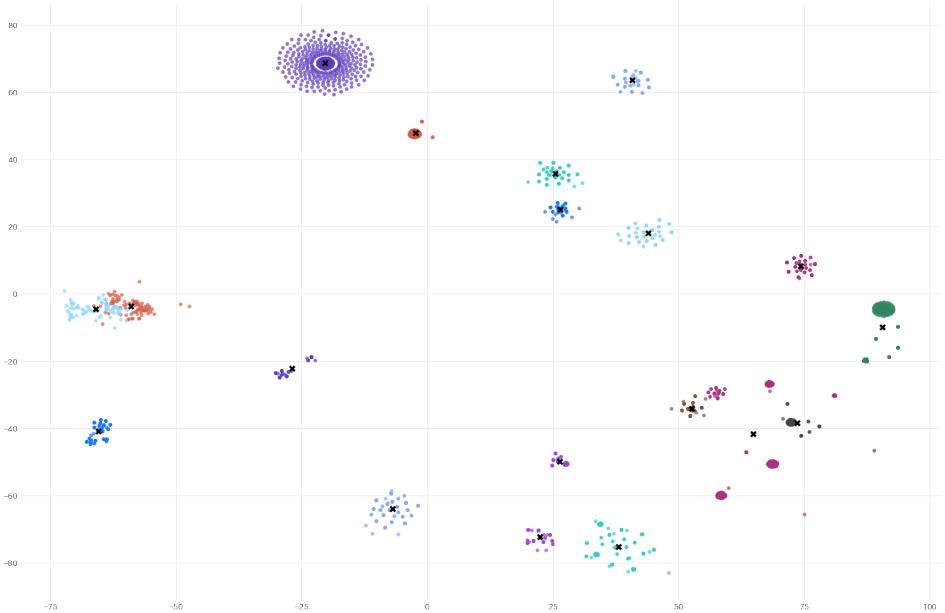

Figure 5.3: HDVector n-grams $n = 3$

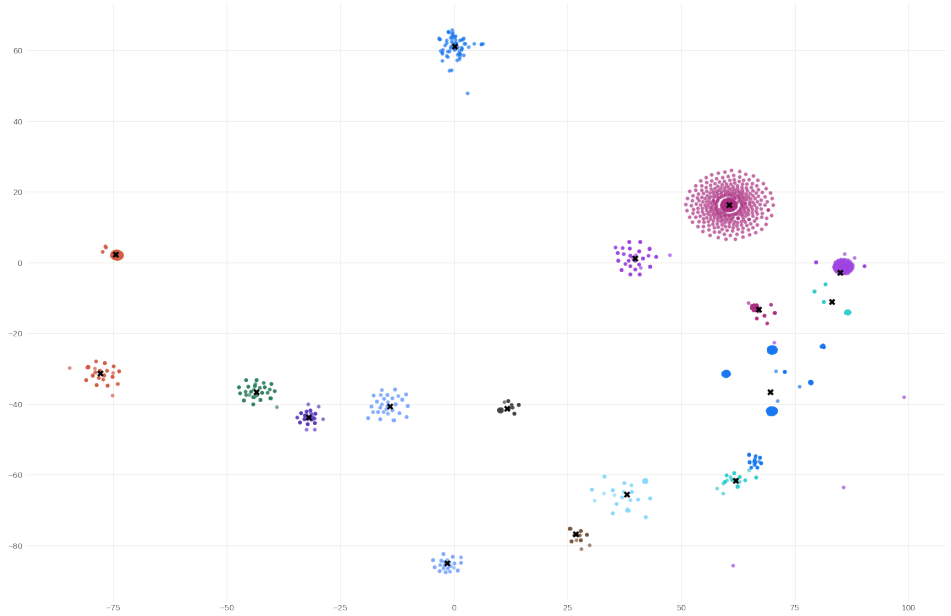

Figure 5.4: HDVector n-grams $n = 4$

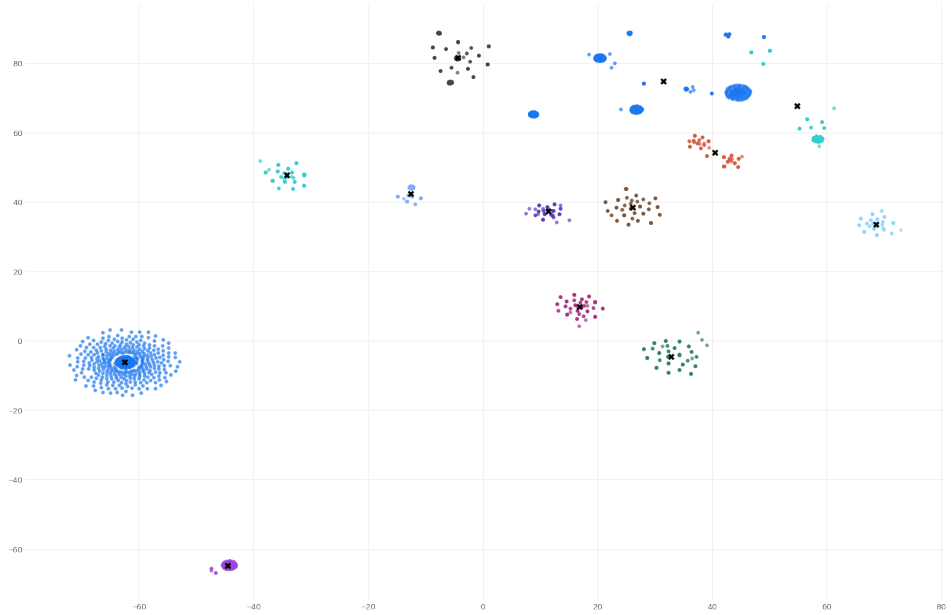

Figure 5.5: HDVector n-grams $n = 5$

5.2 Clustering results

The results of running our two clustering algorithms described in section 4.3 are presented in the tables 5.2 and 5.3. For each algorithm, each data representation method was tested whilst varying one primary parameter of the algorithm. This was done as mentioned in section 4.1 the results presented in the tables are the averages over those 5 datasets, which is why 'Clusters' is not presented as a integer number.

Avg Size represents the average size of the clusters for a run. FB Acc is calculated for each cluster as described in section 4.4.2, and is then averaged first per run and then between all datasets. Sil(houette) Score and Adj Sil Score are calculated as described in section 4.4.3.

The overall goal of the value in each column is to be as large as possible while not negatively affecting any of the other columns. Therefore an 'Overall' score is calculated for each row simply by multiplicating all values except avg cluster size in each row, which gives a guiding value for which combination of algorithm, data representation and parameter yielded the overall best results.

Table 5.2: Results for clustering with Hierarchical Agglomerative Clustering, using different data representations.

Method	Clusters	Avg Size	FB Acc	Sil Score	Adj Sil Score	Overall
Jaccard t=0.2	12.4	97.0	0.888750	0.311340	0.749159	2.57
Jaccard t=0.4	14.8	113.35	0.888779	0.314197	0.673119	2.78
Jaccard t=0.5	17.2	117.53	0.855634	0.310708	0.666675	3.05
Jaccard t=0.75	31.8	118.91	0.855073	0.250363	0.447783	3.05
HDVector t=0.2	15.8	112.2	0.891730	0.333868	0.714308	3.36
HDVector t=0.4	27.8	118.89	0.878886	0.335961	0.514565	4.22
HDVector t=0.5	29.8	121.27	0.861274	0.316833	0.483834	3.93
HDVector t=0.75	44.0	148.18	0.712813	0.273230	0.402436	3.45
HDVector2 t=0.2	15.6	100.71	0.871843	0.333868	0.535248	2.43
HDVector2 t=0.4	38.8	133.8	0.762654	0.265864	0.308051	2.42
HDVector2 t=0.5	30.2	308.13	0.487405	0.213583	0.144330	0.45
HDVector2 t=0.75	1.0	9570.6	0.336196	0.0	0.0	0.0
Doc2Vec t=0.2	21.2	97.78	0.844697	0.186407	0.295855	0.99
Doc2Vec t=0.4	22.4	151.59	0.785514	0.280665	0.476130	2.35
Doc2Vec t=0.5	24.2	157.88	0.770898	0.274395	0.449959	2.30
Doc2Vec t=0.75	34.2	158.6	0.676039	0.227502	0.389680	2.05

A common theme for almost all HAC runs in table 5.2 is that the number of clusters increase when t is increased. This can be expected as t controls how far between clusters

and items can still be considered similar, and therefore more clusters with more than 50 items are found. Examining the different data representations, all representations had quite decent accuracy for $t = [0.2, 0.4]$. The main thing that differs for these values is how many clusters are found, and what silhouette scores are calculated for them. Additionally Jaccard and Doc2Vec produce a steady value of clusters when $t = [0.2, 0.4, 0.5]$ compared to the hypervector implementations. The HDVector2 t=0.75 run resulting in one cluster (which in turn produces a silhouette score of 0) is because the normalized hamming distance for two random (0,1)-binary hypervectors will be around 0.5, meaning that a distance threshold above that would consider all items to be from one big cluster.

Table 5.3: Results for clustering with DBSCAN, using different data representations.

Method	Clusters	Avg Size	FB Acc	Sil Score	Adj Sil Score	Overall
Jaccard eps=0.1	4.2	123.57	0.788243	0.966728	0.797842	2.55
Jaccard eps=0.2	17.8	143.38	0.886407	0.552311	0.594982	5.18
Jaccard eps=0.4	14.1	420.13	0.714311	0.146582	0.103012	0.15
Jaccard eps=0.5	4.8	1649.57	0.614064	-0.013862	0.063106	0.00
HDVector eps=0.1	14.2	190.99	0.811893	0.641321	0.654076	4.84
HDVector eps=0.2	17.6	284.79	0.723152	0.222881	0.376874	1.07
HDVector eps=0.4	1.8	5029.1	0.416580	-0.010272	0.076180	0.0
HDVector eps=0.5	1.0	9464	0.338606	0.059307	0.0	0.0
HDVector2 eps=0.1	6.6	198.73	0.851350	0.732700	0.681204	2.80
HDVector2 eps=0.2	17.0	332.64	0.722329	0.132759	0.235136	0.38
HDVector2 eps=0.4	1.0	9568.4	0.336120	0.0	0.0	0.0
HDVector2 eps=0.5	1.0	9570.6	0.336055	0.0	0.0	0.0
Doc2Vec eps=0.1	11.4	496.5	0.706639	0.176955	0.340904	0.49
Doc2Vec eps=0.2	2.0	6450	0.350994	-0.187435	0.144124	-0.02
Doc2Vec eps=0.4	1.0	9472	0.335993	0.029505	0.0	0.0
Doc2Vec eps=0.5	1.0	9567.6	0.336384	0.0	0.0	0.0

Overall DBSCAN results in table 5.3 have a few distinct differences compared to HAC runs in table 5.2. DBSCAN finds less clusters with more than 50 items, but can have a much higher silhouette score. This is because anything DBSCAN considers noise is not included in the silhouette score calculation, contrary to HAC that will force a cluster and silhouette score for every item n the dataset. However, DBSCAN is also reporting negative silhouette scores, which is described in section 4.4.3 as having items that most likely should belong to another cluster than the one they were assigned to. Some of the runs even report only one cluster, meaning that the algorithm likely has its parameter set too high. Looking at the dataset sizes in table 4.1 we see that the only cluster it finds nears the size of the whole dataset, which strengthens this claim.

From these aggregated results, we select a few ones to look at closer:

- **HAC Jaccard t=0.5**
- **HAC HDVector t=0.2**
- **HAC HDVector t=0.4**
- **HAC Doc2Vec t=0.4**
- **DBSCAN Jaccard eps=0.2**
- **DBSCAN HDVector eps=0.1**

These are selected because they have the best combinations of values and scores, and subsequently produce the highest Overall scores. Details for these runs are summarized in table 5.4, with values for each selected method versus each dataset. By doing this, the HDVector2 data representation is not further examined. This was somewhat expected as the binary hypervectors are known to have less accuracy than bipolar hypervectors that don't get truncated, and for the aggregated results HDVector2 performed more poorly for all parameters and combinations.

Table 5.4: Results per dataset for methods that we are diving into.

Method	Dataset	Clusters	Avg Size	FB Acc	Sil Score	Adj Sil Score
HAC						
Jaccard t=0.5	Dataset 1	7	109.71	0.931178	0.282980	0.681091
Jaccard t=0.5	Dataset 2	16	145.31	0.835228	0.320002	0.677073
Jaccard t=0.5	Dataset 3	25	106.72	0.858929	0.318254	0.616092
Jaccard t=0.5	Dataset 4	15	95.67	0.766408	0.288431	0.629339
Jaccard t=0.5	Dataset 5	18	130.22	0.893284	0.318272	0.697565
HDVector t=0.2	Dataset 1	6	117.33	0.917393	0.311145	0.760073
HDVector t=0.2	Dataset 2	15	133.8	0.912523	0.340481	0.690754
HDVector t=0.2	Dataset 3	23	98.86	0.898861	0.350009	0.676802
HDVector t=0.2	Dataset 4	15	92.87	0.860888	0.319670	0.654318
HDVector t=0.2	Dataset 5	17	118.71	0.935092	0.335455	0.693338
HDVector t=0.4	Dataset 1	13	111.85	0.916742	0.305052	0.432025
HDVector t=0.4	Dataset 2	23	137.91	0.869518	0.345985	0.596120
HDVector t=0.4	Dataset 3	32	114.13	0.876756	0.348655	0.588531
HDVector t=0.4	Dataset 4	25	108.76	0.821769	0.317627	0.529345
HDVector t=0.4	Dataset 5	35	121.8	0.889445	0.355528	0.543546
Doc2Vec t=0.4	Dataset 1	11	130.45	0.931510	0.245740	0.483257
Doc2Vec t=0.4	Dataset 2	25	156	0.810802	0.299183	0.468620
Doc2Vec t=0.4	Dataset 3	24	186.83	0.724893	0.285468	0.413601
Doc2Vec t=0.4	Dataset 4	24	132.92	0.745924	0.289819	0.479546
Doc2Vec t=0.4	Dataset 5	26	151.73	0.793207	0.301431	0.471515
DBSCAN						
Jaccard eps=0.2	Dataset 1	7	160.71	0.929998	0.502084	0.556105
Jaccard eps=0.2	Dataset 2	15	180.53	0.862234	0.556639	0.625499
Jaccard eps=0.2	Dataset 3	27	107.67	0.887277	0.625450	0.654552
Jaccard eps=0.2	Dataset 4	17	126.59	0.852698	0.540423	0.556135
Jaccard eps=0.2	Dataset 5	24	141.38	0.902769	0.541889	0.566242
HDVector eps=0.1	Dataset 1	6	220.5	0.813889	0.676462	0.648211
HDVector eps=0.1	Dataset 2	13	210.77	0.839949	0.596388	0.610169
HDVector eps=0.1	Dataset 3	22	136.5	0.810578	0.670981	0.717888
HDVector eps=0.1	Dataset 4	15	161.87	0.781724	0.630856	0.635677
HDVector eps=0.1	Dataset 5	15	225.33	0.851480	0.587761	0.651982

If we recall the ground truth values for the datasets described in table 4.1, there are a few observations to be made. When comparing the selected HAC runs with the selected DBSCAN runs, the two main differences are the average cluster sizes being closer to or higher to the ground truth values for DBSCAN, as well as the silhouette scores being higher as mentioned earlier. This points towards DBSCAN possibly being more suitable for clustering data with these types of data representation, and could be evaluated further with more tuning to the DBSCAN algorithm. None of the algorithms are able to find 14 known clusters for Dataset 1, and while HAC HDVector t=0.4 came close it, the average cluster size is far lower. Apart from that, HAC HDVector t=0.4 and HAC Doc2Vec t=0.4 have found more clusters than what are known which either means there are unknown clusters, or known clusters might have been split in two. This can be verified by diving into the contents of the clusters, which is done in section 5.3.

5.3 Cluster Contents

To get an understanding of what type of clusters the combinations of algorithms and data repreresentations actually have created, five runs were picked from table 5.4 and the clusters were examined with the ground truth values from table 4.1. The comparison was done by extracting the error signal from all items in the cluster and picking the majority value. This is also how the known clusters in the ground truth data were calculated. If the error signal exists both in the ground truth and the clustered data, it is considered a known cluster. Any duplicate error signal clusters in the same run are counted towards "Known Duplicates". If the error signal was not found in the ground truth data, it is counted as an unknown cluster, and any duplicates of that error signal as "Unknown Duplicates". The ground truth data contains no duplicate clusters due to the way it was built. The data is presented in table 5.5.

Table 5.5: Known and unknown clusters found

Method	Dataset	Known	Known Dup	Unknown	Unknown Dup
HAC					
Jaccard t=0.5	Dataset 5	9	3	4	1
HDVector t=0.4	Dataset 5	13	13	4	1
Doc2Vec t=0.4	Dataset 3	12	2	11	1
DBSCAN					
Jaccard eps=0.2	Dataset 5	14	4	4	1
HDVector eps=0.1	Dataset 3	7	4	8	4

While some of the runs reached the number of known clusters in table 5.4, here we can see that all of the runs are a mix of known and unknown clusters, plus several of them duplicates. One explanation for not finding all known clusters can possibly be

explained by the same way duplicate clusters have been found here. If a known cluster is split into smaller clusters due to other features causing them to seem unique enough, it could create clusters with fewer than 50 items in each which would not show up in our analysis. By looking at the t-SNE graphs in figures 5.6-5.10, we get some indication of this behavior in figure 5.7 where several clusters are closely intertwined.

Another observation that can be made in the t-SNE graph in figure 5.8 is that the data representation Doc2Vec produces way tighter clusters overall when compared to any of the other representations. It will however not create the round eye-shaped clusters that can be seen in Jaccard and HDVector t-SNE graphs. One explanation for this could be that Doc2Vec did not use n-grams and was given more context of the surrounding words when building the vectors.

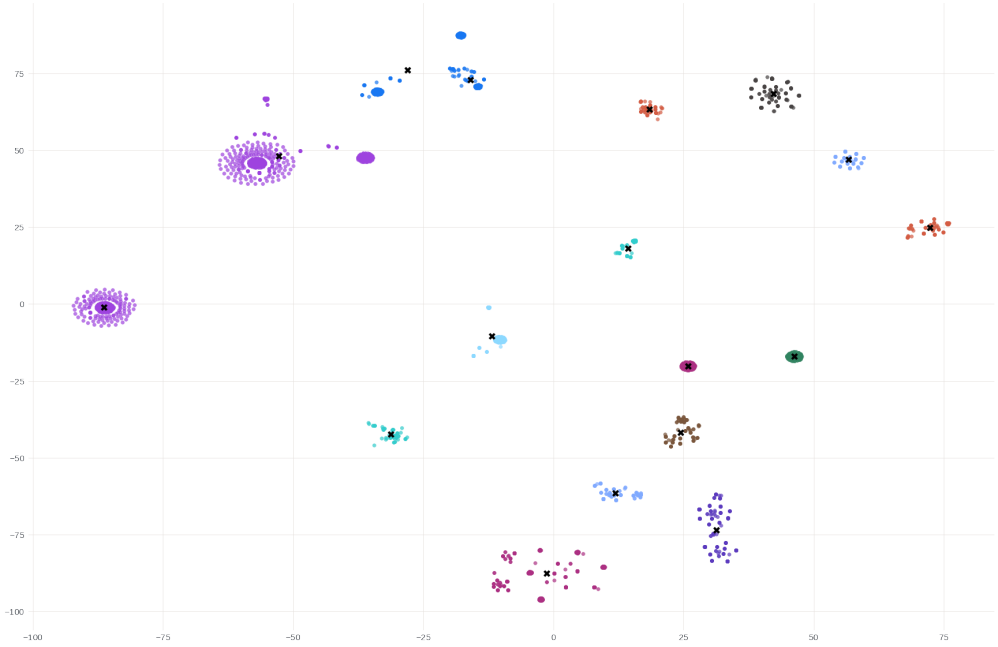

Figure 5.6: Jaccard HAC t=0.5 on Dataset 5

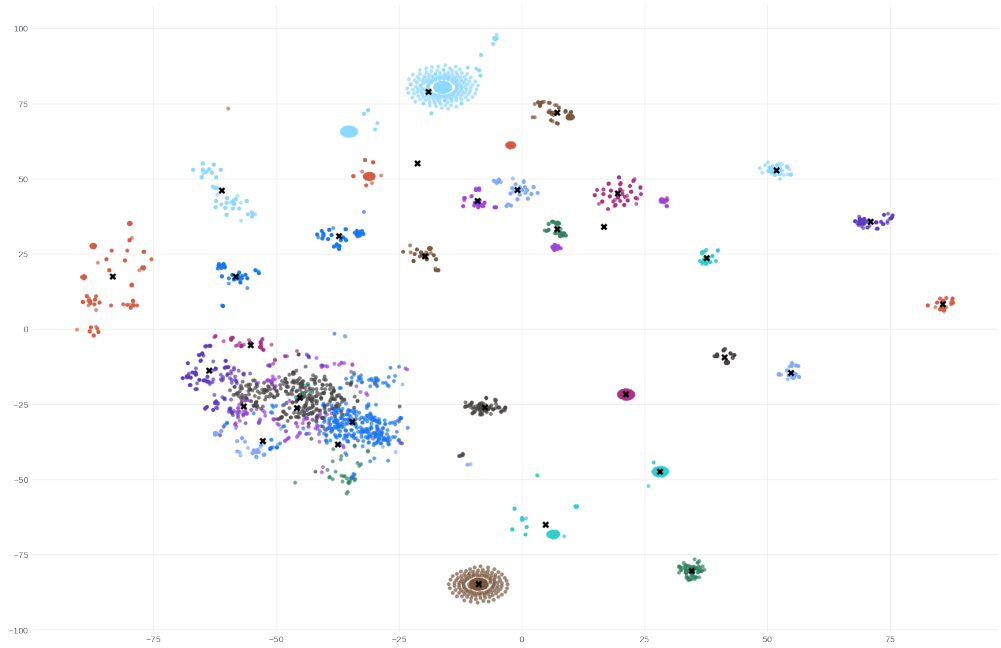

Figure 5.7: HDVector HAC t=0.4 on Dataset 5

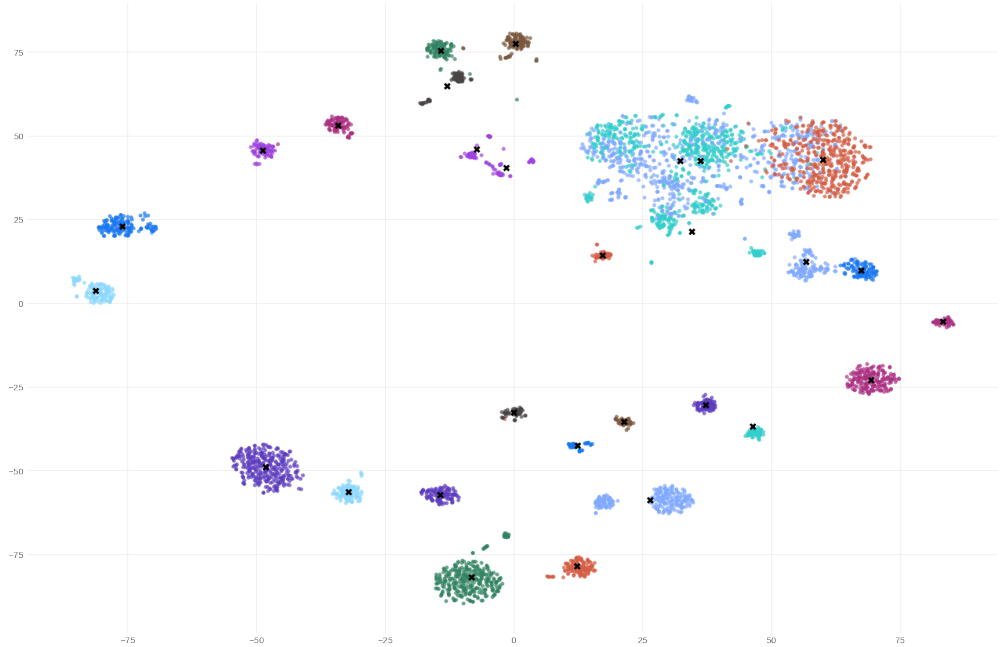

Figure 5.8: Doc2Vec HAC t=0.4 on Dataset 3

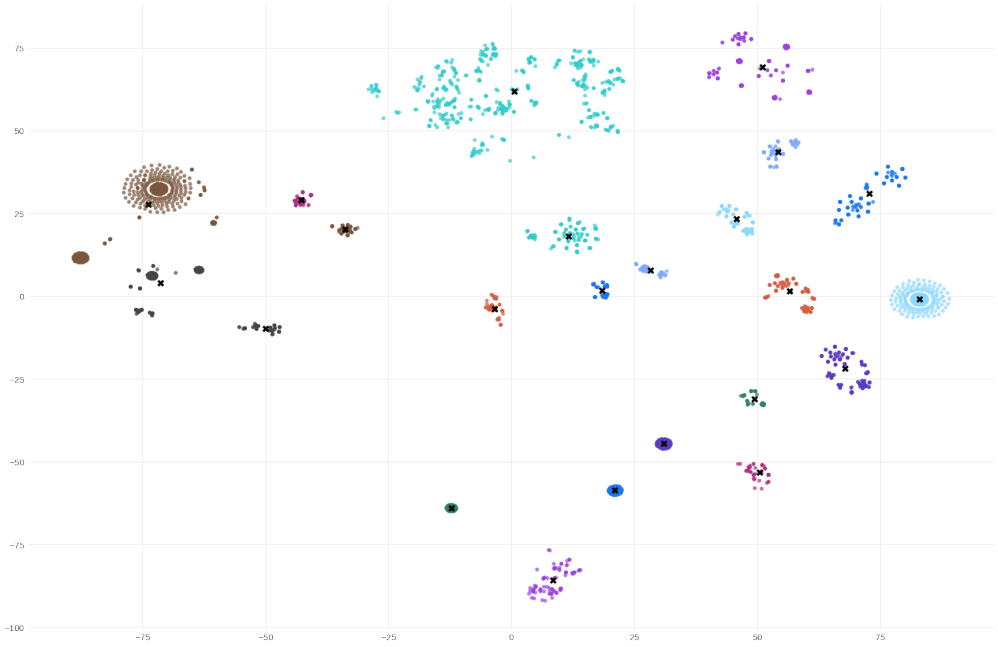

Figure 5.9: Jaccard DBSCAN eps=0.2 on Dataset 5

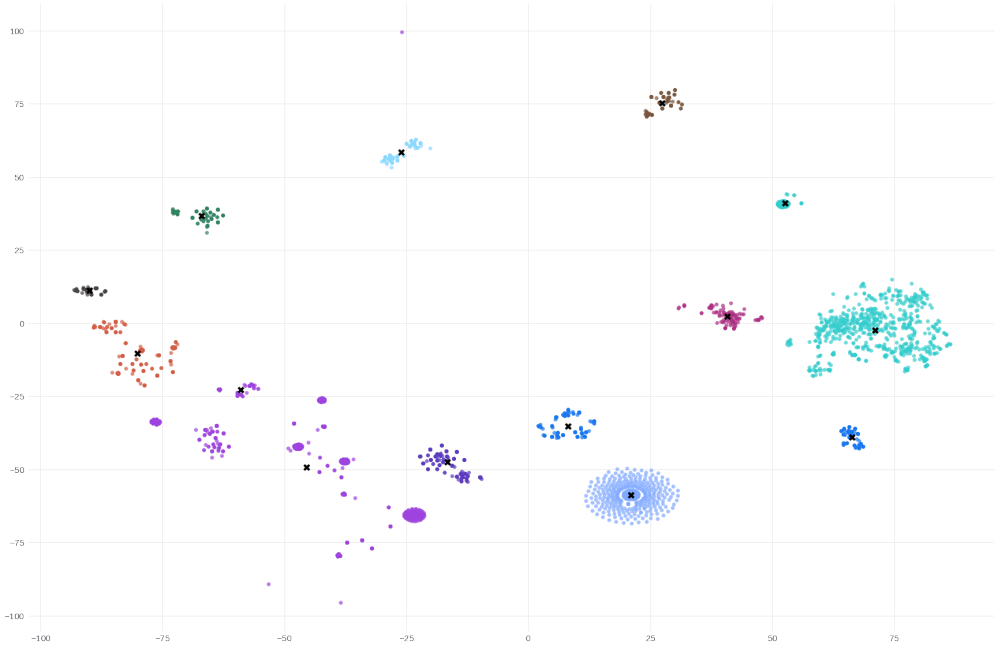

Figure 5.10: HDVector DBSCAN eps=0.1 on Dataset 3

To verify that duplicate clusters aren't just due to 'bad' clustering, one of the runs were further examined based on the error signal each error report contained and presented in table 5.6. The goal was to ensure clusters that are built for the same error signal have other reasonably distinct features to be considered unique. This was done by manually examining the error reports and their metadata in the clusters by using an internal tool for error reports.

Table 5.6: Manual examination of duplicate error signal clusters.

Error Signal	FB Accuracy	Cluster ID	Common feature
Unknown	0.9974619289340102	2196	Gen 2 Microserver
Unknown	1.0	1298	Gen 1 Microserver
Signal 1	1.0	122	Model Type 1
Signal 1	1.0	692	Model Type 10
Signal 1	1.0	87	Model Type 1
Signal 2	1.0	1608	Model Type 1 Manufacturer A
Signal 2	1.0	552	Model Type 1 Manufacturer B
Signal 2	1.0	853	Model Type 10
Signal 3	1.0	830	Model Type 1 Manufacturer A
Signal 3	1.0	935	Model Type 1 Manufacturer B

For all duplicate error signals (including the 'unknown' signal), a distinct common feature that validated the cluster's existence was found. Additionally all clusters but one had an accuracy of 1.0 which generally corresponds to it being a known error with an automated diagnosis. The only duplication that could not easily be explained was Signal 1 on Model Type 1, which had two clusters. There are however some features in the datasets that are not easily surfaced without digging through several internal tools, so it was not further examined due to limitation in time.

Chapter 6

Discussion

In this chapter a reflection of the results and the evaluation method is presented, and the research questions mentioned in chapter 1 will be evaluated. Suggestions and improvements for future work are also presented.

6.1 Results Evaluation

Evaluating if a clustering algorithm builds good cluster on a dataset is a fairly hard task, especially since the notion of a good cluster is subjective to the person or process that gets to interpret and work with the cluster afterwards. By setting a reasonable constraint such as delimitation **D1** specified in section 1.4 that could then be translated into an accuracy metric in section 4.4.2, the overall evaluation process was greatly simplified as it gives a hard number that can be compared between algorithms that produce the same output. It also served as a counterweight to the silhouette score metric that is otherwise often used when determining the quality of a cluster. While it provides a good indication of if the clustering run should be considered at all, it also wasn't reliable on its own as it could be skewed when different algorithms handled things like noise differently.

Another consideration that isn't always done is going to lengths to examine the data in built clusters to actually verify that each and every cluster 'makes sense' to a person who has an expectation of what the cluster should contain. While this expectation is as mentioned earlier subjective, getting and recording that feedback atleast gives an indication of the real-world value of applying the clustering algorithm on a specific problem.

Looking at the results presented in chapter 5, and section 5.1 there is strong evidence that the data recorded from the state transitions has grammatical properties that can be captured by n-grams and by doing such it enhances the clustering results. That $n = 3$ is a good value can be inferred partly by looking at the structure of the FSA log example in section 3.1.2. With $n = 3$, the context captured from a raw string of the

form "input state input state input state ..." will always be of one of the two forms a) "input state input" or b) "state input state". Since a finite-state automata's next state is only affected by it's current state and current input, b) gives you a representation of this relation for all items in the log. And since the grammar of the regular language is never more complex than the relation between two adjacent inputs, a) provides this representation to the data.

For the comparison of clustering algorithm, data representation and parameter value, there is good evidence that the data in the datasets can be clustered using several of the tested algorithms and data representations and provide value. It should however be noted that the choice of parameters must be carefully tested depending on the number and different types of features that is used as input data before converting it to a certain data representation. One example is the distance threshold t for the Hierarchical Agglomerative Clustering algorithm. When coupled with Jaccard Similarity, the quality of the threshold is directly determined by the number of features contained in each set. For example if the dataset contains 5 unique features, having one feature of a difference between two sets creates a distance difference of 0.2, which can easily throw an item over a threshold value.

Using hypervectors for data representation mitigates this a bit as a more complex similarity metric is used, but since each distinct feature represented as a hypervector is approximately orthogonal from any other feature it can still cause big jumps in distance by adding or removing just one feature if the overal feature count is low.

For Doc2Vec we are somewhat in the blind trying to understand what happens as the the model is trained, and it is only by manually examining the data and comparing what is similar to what that you get somewhat of an understanding. It also does not function well at all if the feature count or overall dataset size becomes too low, which is likely an issue in our case with only about 10000 items in each dataset. It is however a good contender, and would be a clear choice if the number of features are extended with more rich log data.

Lastly, there will be no apparent 'winner' picked as it was not one of the goals of the thesis. Instead, the results will be used as motivation for answering the research questions below.

6.2 Research Questions

In chapter 1 several research questions were established as part of the problem description. These will now be answered based on the theory, methods and results presented in previous chapters.

Q1: How well can common clustering methods be used to cluster server error reports in an unsupervised manner?

Based on the results presented in sections 5.2 and 5.3, together with the accuracy metric described in section 4.4.2, common clustering methods can serve **very well** in real-world scenarios to reduce manual work when assessing systematic issues in large-scale data center fleets. Since the accuracy metric was based on error reports having the same desired outcome, and the overall input data for clustering resembling what engineers were manually looking at today to achieve this very same goal, the results show that this step in the process can be automated.

Q2: Can you cluster on key-value data and state transition data together effectively?

Yes, by combining data representations such as sets as in section 4.3.3, or random indexing via hypervectors as in section 4.3.4 both data structures can be represented as one entity that can represent items in datasets. Further, by augmenting input data for state transitions with n-grams by utilizing the underlying grammatical properties of the state transitions as noted in section 3.1.2, the data representation is further enhanced to accurately depict the state transition input data as proven in section 5.1. This is further discussed in 6.1 by evaluating why trigrams does well at representing the FSA log data.

Q3: How does different data representations affect the ability to build good clusters?

As evident by the results presented in tables 5.2-5.5, data representation and their corresponding similarity metrics highly influence the ability to build good clusters. For simple cases like sets + Jaccard Similarity, where you essentially have no fuzziness involved as a feature in a set either exists or it doesn't, it is a bit more cautious building clusters as there needs to be enough full matches to build up to the desired threshold, whereas for something like random indexing and doc2vec with cosine similarity will have a degree of fuzziness as the vector representations are built and mangled together. Parts of the answer to Q2 are also applicable here, as finding ways to accurately represent the syntactic and semantic structure of the data also plays into how well the data is represented.

6.3 Future Work

Looking at the results that this thesis presents, there several things that could be done by future work to both further explore the area of clustering error reports in this hyperscale data center operations setting, but also in the area of clustering on mixed data types.

One fairly straight forward thing would be to actually start building clusters based on production data, and give it to engineers that actually work with this type of data manually today, to get feedback on how valuable it is for them in their daily work and possibly reduce the amount of hours needed. This would provide more real-world signal than trying to represent it solely with the metric in section 4.4.2. Another thing would be to flip the delimitation D4 around and instead of optimizing for finding large clusters, to instead find small clusters that contain more unique and complex error reports and server history. This could be good to help reduce the so called ”long tail” that occurs when you get complex hardware and software failures that normal systems cannot diagnose.

For clustering on different data types it would be interested to use something like the hybrid clustering method that was looked at in chapter 2, to assign different weights to different types of features for an item. This would allow to, for example, treat certain key-value metrics that could too easily affect clustering outcome as less valuable. With the current implementation every feature has an equal weight to affect the outcome of the clusters.

If more data could be gathered for each server, properly testing something like doc2vec as data representation would be very interesting to pursue further. This would however require some thought to wheter all data should be represented as one plan-text document as is done in this thesis, or if this solution also should implement something like hybrid clustering to better represent the semantic differences between raw text data and key-value data.

Chapter 7

Conclusion

In this master's thesis the usage of unsupervised clustering methods has been proposed and tested as a way to speed up and enhance the process of finding and root causing systematic issues when working with server error reports in a hyperscale data center operations environment. With a mix of server properties as key-value data, and historical data as a log of events, several data representations and clustering algorithms were evaluated. Results show that using data representations that can represent syntactic and semantic structures of historical events modelled as finite-state automatas improves clustering accuracy in real-world scenarios.

By leveraging the relationship between regular languages and finite-state automatas, the application of n-grams on sentences that conform to a certain regular grammar can capture the inherent grammatical properties and improve the accuracy when comparing similarities between such sentences. This is not limited to the area of hyperscale data center operations, but can be applied to other problems where event logs can be modelled as finite-state automatas.

Comparing clustering solutions for a particular problem can be really complex as the definition of good and bad clusters must be established in a measurable way beforehand. Additionally it might not be enough to use just one metric during comparison, as there are multiple aspects that needs to be taken into account when clusters are generated such as examining contents and verifying items in clusters make sense with each other.

From the results, both the simpler data representation using sets and Jaccard Similarity as well as bipolar hypervectors with cosine similarity worked well to achieve high accuracy and well-defined clusters for the defined clusters. There were noticeable differences for the different similarity metrics that affected how many clusters were found or how many items each cluster contained that needs to be considered on a case-by-case basis.

With the results from this thesis, the long-term plan is to build a proof-of-concept that can be applied in a production environment to further evaluate the different methods,

and enhance the way Facebook works with hyperscale data center operations.

Bibliography

[1] Dashveenjit Kaur. *The future of data centers is hyperscale*. URL: https://techwireasia.com/2021/03/the-future-of-data-centers-is-hyperscale/. (accessed: 2021-09-22).

[2] Matt Kapko. *How (and Why) Facebook Excels at Data Center Efficiency*. URL: https://www.cio.com/article/2854720/how-and-why-facebook-excels-at-data-center-efficiency.html. (accessed: 2021-09-22).

[3] F Lin et al. "Hardware Remediation At Scale". In: *International Conference on Dependable Systems and Networks (DSN)* (2020). URL: https://research.fb.com/publications/hardware-remediation-at-scale/.

[4] F Lin, H Dattatraya Dixit, and S Sankar. *How Facebook keeps its large-scale infrastructure hardware up and running*. URL: https://engineering.fb.com/2020/12/09/data-center-engineering/how-facebook-keeps-its-large-scale-infrastructure-hardware-up-and-running/. (accessed: 2021-09-22).

[5] *Open Compute Project*. URL: https://www.opencompute.org/. (accessed: 2021-09-23).

[6] Alina Sîrbu and Ozalp Babaoglu. "Towards Data-Driven Autonomics in Data Centers". In: *2015 International Conference on Cloud and Autonomic Computing*. 2015, pp. 45–56. DOI: 10.1109/ICCAC.2015.19.

[7] Fred Lin et al. "Predicting Remediations for Hardware Failures in Large-Scale Datacenters". In: June 2020, pp. 13–16. DOI: 10.1109/DSN-S50200.2020.00016.

[8] Fred Lin et al. "Fast Dimensional Analysis for Root Cause Investigation in a Large-Scale Service Environment". In: *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 4.2 (June 2020), pp. 1–23. ISSN: 2476-1249. DOI: 10.1145/3392149. URL: http://dx.doi.org/10.1145/3392149.

[9] Sourabh Jain et al. "Extracting the textual and temporal structure of supercomputing logs". In: *2009 International Conference on High Performance Computing (HiPC)*. 2009, pp. 254–263. DOI: 10.1109/HIPC.2009.5433202.

[10] O. Maimon and L. Rokach. *Data Mining and Knowledge Discovery Handbook*. Springer US, 2006. ISBN: 9780387254654. URL: https://books.google.se/books?id=S-XvEQWABeUC.

[11] Tung Khuat, Nguyen Hung, and Le Thi My Hanh. "A Comparison of Algorithms used to measure the Similarity between two documents". In: *International Journal of Advanced Research in Computer Engineering Technology (IJARCET)* 4 (Apr. 2015), pp. 1117–1121.

[12] David Combe et al. "Combining Relations and Text in Scientific Network Clustering". In: *2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. 2012, pp. 1248–1253. DOI: 10.1109/ASONAM.2012.215.

[13] John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition)*. USA: Addison-Wesley Longman Publishing Co., Inc., 2006. ISBN: 0321455363.

[14] Dmitry Pashchenko et al. "Search for a substring of characters using the theory of non-deterministic finite automata and vector-character architecture". In: *Bulletin of Electrical Engineering and Informatics* 9 (June 2020). DOI: 10.11591/eei.v9i3.1720.

[15] N. Chomsky. "Three models for the description of language". In: *IRE Transactions on Information Theory* 2.3 (1956), pp. 113–124. DOI: 10.1109/TIT.1956.1056813.

[16] Fernando Pereira. "Finite-State Approximations of Grammars". In: *Proceedings of the Workshop on Speech and Natural Language*. HLT '90. Hidden Valley, Pennsylvania: Association for Computational Linguistics, 1990, pp. 20–25. DOI: 10.3115/116580.116592. URL: https://doi.org/10.3115/116580.116592.

[17] William B Cavnar, John M Trenkle, et al. "N-gram-based text categorization". In: *Proceedings of SDAIR-94, 3rd annual symposium on document analysis and information retrieval*. Vol. 161175. Citeseer. 1994.

[18] Andrija Tomović, Predrag Janičić, and Vlado Kešelj. "n-Gram-based classification and unsupervised hierarchical clustering of genome sequences". In: *Computer Methods and Programs in Biomedicine* 81.2 (2006), pp. 137–153. ISSN: 0169-2607. DOI: https://doi.org/10.1016/j.cmpb.2005.11.007. URL: https://www.sciencedirect.com/science/article/pii/S0169260705002361.

[19] Abhinav Sethy and Bhuvana Ramabhadran. "Bag-of-word normalized n-gram models". In: *Ninth Annual Conference of the International Speech Communication Association*. 2008.

[20] Pentti Kanerva. "Hyperdimensional Computing: An Introduction to Computing in Distributed Representation with High-Dimensional Random Vectors". In: *Cognitive Computation* 1 (June 2009). DOI: 10.1007/s12559-009-9009-8.

[21] Tomas Mikolov et al. *Efficient Estimation of Word Representations in Vector Space*. 2013. arXiv: 1301.3781 [cs.CL].

[22] Quoc V. Le and Tomas Mikolov. *Distributed Representations of Sentences and Documents*. 2014. arXiv: 1405.4053 [cs.CL].

[23] Jasmine Irani, Nitin Pise, and Madhura Phatak. "Clustering Techniques and the Similarity Measures used in Clustering: A Survey". In: *International Journal of Computer Applications* 134 (Jan. 2016), pp. 9–14. DOI: 10.5120/ijca2016907841.

[24] T. S. Jayram, Ravi Kumar, and D. Sivakumar. "The One-Way Communication Complexity of Hamming Distance". In: *Theory of Computing* 4.6 (2008), pp. 129–135. DOI: 10.4086/toc.2008.v004a006. URL: http://www.theoryofcomputing.org/articles/v004a006.

[25] Laurens van der Maaten and Geoffrey Hinton. "Viualizing data using t-SNE". In: *Journal of Machine Learning Research* 9 (Nov. 2008), pp. 2579–2605.

[26] Raghav Sethi et al. "Presto: SQL on Everything". In: *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 2019, pp. 1802–1813. DOI: 10.1109/ICDE.2019.00196.

[27] Thomas Kluyver et al. "Jupyter Notebooks - a publishing format for reproducible computational workflows". In: *Positioning and Power in Academic Publishing: Players, Agents and Agendas*. Ed. by Fernando Loizides and Birgit Scmidt. Netherlands: IOS Press, 2016, pp. 87–90. URL: https://eprints.soton.ac.uk/403913/.

[28] The pandas development team. *pandas-dev/pandas: Pandas*. Version latest. Feb. 2020. DOI: 10.5281/zenodo.3509134. URL: https://doi.org/10.5281/zenodo.3509134.

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

[30] Peter J. Rousseeuw. "Silhouettes: A graphical aid to the interpretation and validation of cluster analysis". In: *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53–65. ISSN: 0377-0427. DOI: https://doi.org/10.1016/0377-0427(87)90125-7. URL: https://www.sciencedirect.com/science/article/pii/0377042787901257.