POLITECNICO DI BARI

Dipartimento di Ingegneria Elettrica e dell'Informazione – DEI

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

ADVANCED SOFTWARE ENGINEERING
BUSINNES REPORT

AUTOMATIC LOG CLUSTERING USING MACHINE LEARNING

Professor:
Prof. Marina Mongiello

Students:
Pietro SPALLUTO
Luca FISCHETTI

Anno Accademico 2021-2022

Contents

List of Figures

Introduction

With the growing scale and complexity of the IT systems, debugging and testing become more difficult. As it is very difficult for the verification engineer to have an insight in every piece of work that the developers do, the test results based on the automated test scripts are not completely reliable. The logs generated by the systems, in another point of view, reveal the states of a running system and contain a wealth of information produced by the system to support diagnosis, hence, the log analysis is a vital method of detecting failures or problems in a large system. It is, however, not realistic and practical to analyze huge amounts of logs manually. In a Big Data world, using machine learning techniques to do automated log analysis becomes interesting. This project aims to extract effective features from the free text functional test loop logs, detect the abnormal logs automatically through machine learning techniques and investigate the effectiveness of different solutions.

Chapter 1

Domain of Interest

1.1 Testing results cannot be trusted all the time

In a complex system, software and hardware are developed by multiple developers, updated and changed frequently. Generally, the verification engineers are not as familiar with all the system details as the developers. Furthermore, no one can foresee all problems. Many unknown problems will occur with a growing and continuous updated system. It is hard for the verification engineers to create perfect automated test scripts that can detect all potential problems.

1.2 Manual log analysis is hard for large systems

In reality, analyzing the logs manually plays an indispensable role when automated testing cannot figure out the problems. However, a complex system usually involves a wide variety of techniques which need multiple experts of different fields to cooperate.

With the CI (Continuous Integration) process introduced, a large-scale system usually dumps tons of logs every day and the majority of them are not interesting for the testers. It costs too much and is time consuming if we train a lot of experts to identify all interesting logs correctly and look into each of them manually. Therefore, that is done by testers and usually, testers select some suspicious log files instead of looking into all logs and they use simple tools or commands (e.g. grep) to search some predefined keywords (e.g. error, failed, abort) which may help identify problems based on personal experiences.

In order to detect problems through analyzing logs manually, the verification engineer must understand the behaviours of the system and the logs it produces. But the number of existing software and growing new software is quite large and the logs of different software are very different, for this reason it is hard to find such an expert having all necessary knowledge. Therefore, the inefficiency and difficulties of manual log analysis make automated log analysis desirable.

1.3 Using explicit programming is hard

A large-scale system is usually composed of various parts, and those parts may be developed with different programming languages. Despite of the noble efforts of the BSD syslog standard and other standards, the contents of different components' logs can be greatly different. There is no output structure standard for all log files, and it is, therefore, hard to implement one single log analyzer for the development of a large-scale system through explicit programming. Even if a specific log analyzer can be implemented for a specific software, it cannot be commonly used in a system under development without frequent updating. Additionally, as mentioned in the previous paragraph, even as for human beings, it is hard to detect problems from tons of logs correctly, and thus it is nearly impossible to implement a perfect log analyzer through explicit programming by human beings.

1.4 Machine learning might help

Training many people to become such an expert mentioned as previously costs a lot and is inefficient, but people may consider to train a computer system to learn by itself, which is exactly the concerning topics of the machine learning techniques. Machine learning techniques can automatically handle large scale data processing and generalize experiences from learning patterns on a set of training data and apply them on a new data set slightly different from the training data without retraining. Many machine learning techniques have been successfully leveraged in some complex problems, ranging from learning to detect fraudulent credit card transactions, learning users' reading preferences. Machine learning algorithms can automate the process to do log analysis and analyze all logs much faster than humans.

Additionally, machine learning does not rely on explicit programming. It does not need to know all the truth about the problems. In machine learning, unsupervised learning can learn from the structures of the training data without given true answers. Therefore, it is possible to find hidden problems in the logs of different formats and contents by machine learning. Furthermore, it is not a necessary condition that the developer of a machine learning system has any expert knowledge in log analysis but is a plus in finding effective features [1].

Chapter 2

Adopted Techniques

2.1 Parsing

Parsing, syntax analysis, or syntactic analysis is the process of analyzing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar.

2.1.1 Parser

A parser is a software component that takes input data (frequently text) and builds a data structure – often some kind of parse tree, abstract syntax tree or other hierarchical structure, giving a structural representation of the input while checking for correct syntax. The parsing may be preceded or followed by other steps, or these may be combined into a single step. The parser is often preceded by a separate lexical analyzer, which creates tokens from the sequence of input characters; alternatively, these can be combined in scannerless parsing. Parsers may be programmed by hand or may be automatically or semi-automatically generated by a parser generator. Parsing is complementary to templating, which produces formatted output. These may be applied to different domains, but often appear together, such as the scanf/printf pair, or the input (front end parsing) and output (back-end code generation) stages of a compiler.

The task of the parser is essentially to determine if and how the input can be derived from the start symbol of the grammar. This can be done in essentially two ways:

- **Top-down parsing**: it can be viewed as an attempt to find left-most derivations of an input-stream by searching for parse trees using a top-down expansion of the given formal grammar rules. Tokens are consumed from left to right. Inclusive choice is used to accommodate ambiguity by expanding all alternative right-hand-sides of grammar rules. This is known as the primordial soup approach. Very similar to sentence diagramming, primordial soup breaks down the constituencies of sentences.

- **Bottom-up parsing**: a parser can start with the input and attempt to rewrite it to the start symbol. Intuitively, the parser attempts to locate the most basic elements, then the elements containing these, and so on. LR parsers are examples of bottom-up parsers. Another term used for this type of parser is Shift-Reduce parsing [2].

2.2 PCA

Principal Component Analysis (PCA) is a statistical procedure that extracts the most important features of a dataset. Considering a set of 2D points as it is shown

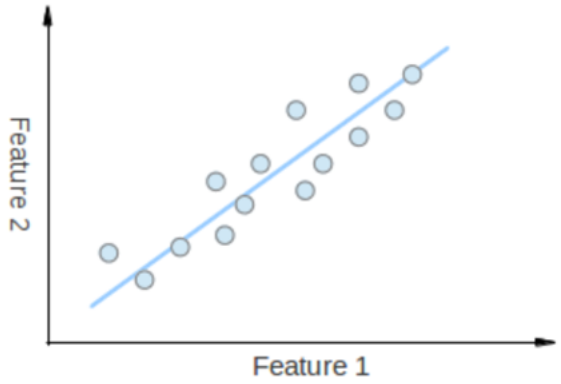


Figure 2.1: Example of PCA using two features.

in Figure 2.1. Each dimension corresponds to a feature you are interested in. Here some could argue that the points are set in a random order. Instead, can be seen that there is a linear pattern (indicated by the blue line) which is hard to dismiss. A key point of PCA is the Dimensionality Reduction. Dimensionality Reduction is the process of reducing the number of the dimensions of the given dataset. For example, in the above case it is possible to approximate the set of points to a single line and therefore, reduce the dimensionality of the given points from 2D to 1D.

Moreover, could be also seen that the points vary the most along the blue line, more than they vary along the Feature 1 or Feature 2 axes. This means that if is known the position of a point along the blue line can be possible to extract more information about the point than if you only knew where it was on Feature 1 axis or Feature 2 axis.

Hence, PCA allows to find the direction along which data varies the most. In fact, the result of running PCA on the set of points in the diagram consist of 2 vectors called eigenvectors (shown inf Figure 2.2) which are the principal components of the data set. The size of each eigenvector is encoded in the corresponding eigenvalue and indicates how much the data vary along the principal component. The beginning of the eigenvectors is the center of all points in the data set. Applying PCA to N-dimensional data set yields N N-dimensional eigenvectors, N eigenvalues and 1 N-dimensional center point [3].

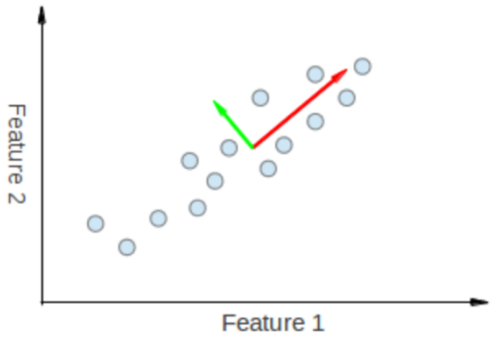


Figure 2.2: Eigenvectors that represent the principal components.

2.2.1 How are the eigenvectors and eigenvalues computed?

The goal is to transform a given data set $\mathbf{X}$ of dimension p to an alternative data set $\mathbf{Y}$ of smaller dimension L. Equivalently, we are seeking to find the matrix $\mathbf{Y}$, where $\mathbf{Y}$ is the Karhunen–Loève Transform (KLT) of matrix $\mathbf{X}$:

$$\mathbf{Y} = \mathbf{KLT}\{X\} \tag{2.1}$$

Organize the data set

Suppose to have data comprising a set of observations of p variables, and the aim is to reduce the data so that each observation can be described with only L variables, $L < p$. Suppose further, that the data are arranged as a set of n data vectors $\mathrm{x}_1 ... \mathrm{x}_\mathrm{n}$ with each x_i representing a single grouped observation of the p variables.

- Write $\mathrm{x}_1 ... \mathrm{x}_\mathrm{n}$ as row vectors, each of which has p columns.
- Place the row vectors into a single matrix $\mathbf{X}$ of dimensions $n \times p$.

Calculate the empirical mean

- Find the empirical mean along each dimension j=1,...,p
- Place the calculated mean values into an empirical mean vector u of dimensions $n \times 1$.

$$u\left[j\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbf{X}\left[i,j\right] \tag{2.2}$$

Calculate the deviations from the mean

Mean subtraction is an integral part of the solution towards finding a principal component basis that minimizes the mean square error of approximating the data. Hence, can be proceeded centering the data as follows:

- Subtract the empirical mean vector u from each row of the data matrix $\mathbf{X}$.
- Store mean-subtracted data in the $n \times p$ matrix $\mathbf{X}$.

$$\mathbf{B} = \mathbf{X} - hu^t \tag{2.3}$$

Where h is a $n \times 1$ column vector of all 1.

Find the covariance matrix

Find the $p \times p$ empirical covariance matrix $\mathbf{C}$ from the outer product of matrix $\mathbf{B}$ with itself:

$$\mathbf{C} = \frac{1}{n-1}\mathbf{B}^* * \mathbf{B} \tag{2.4}$$

Where * is the conjugate transpose operator. Note that if $\mathbf{B}$ consists entirely of real numbers, which is the case in many applications, the "conjugate transpose" is the same as the regular transpose.

Find the eigenvectors and eigenvalues of the covariance matrix

- Compute the matrix $\mathbf{V}$ of eigenvectors which diagonalizes the covariance matrix $\mathbf{C}$:

$$\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D} \tag{2.5}$$

  Where $\mathbf{D}$ is the diagonal matrix of eigenvalues of $\mathbf{C}$.

- Matrix $\mathbf{D}$ will take the form of an $p \times p$ diagonal matrix:

$$\mathbf{D}\left[k, l\right] = \begin{cases} 0, k \neq l \\ \lambda_k, k = l \end{cases} \tag{2.6}$$

- Matrix $\mathbf{V}$, also of dimension $p \times p$, contains p column vectors, each of length p, which represent the p eigenvectors of the covariance matrix $\mathbf{C}$.
- The eigenvalues and eigenvectors are ordered and paired. The j^{th} eigenvalue corresponds to the j^{th} eigenvector.

2.3 K-Means

Clustering represents a set of unsupervised machine learning algorithms belonging to different categories such as prototype-based clustering, hierarchical clustering, density-based clustering etc. K-means is one of the most popular clustering algorithm belonging to prototype-based clustering category. The idea is to create K clusters of data where data in each of the K clusters have greater similarity with other data in the same cluster. The different clustering algorithms sets out rules based on how the data needs to be clustered together. Here (in Figure 2.3) is a diagram representing creation of clusters using K-means algorithms.

It can be noted that:

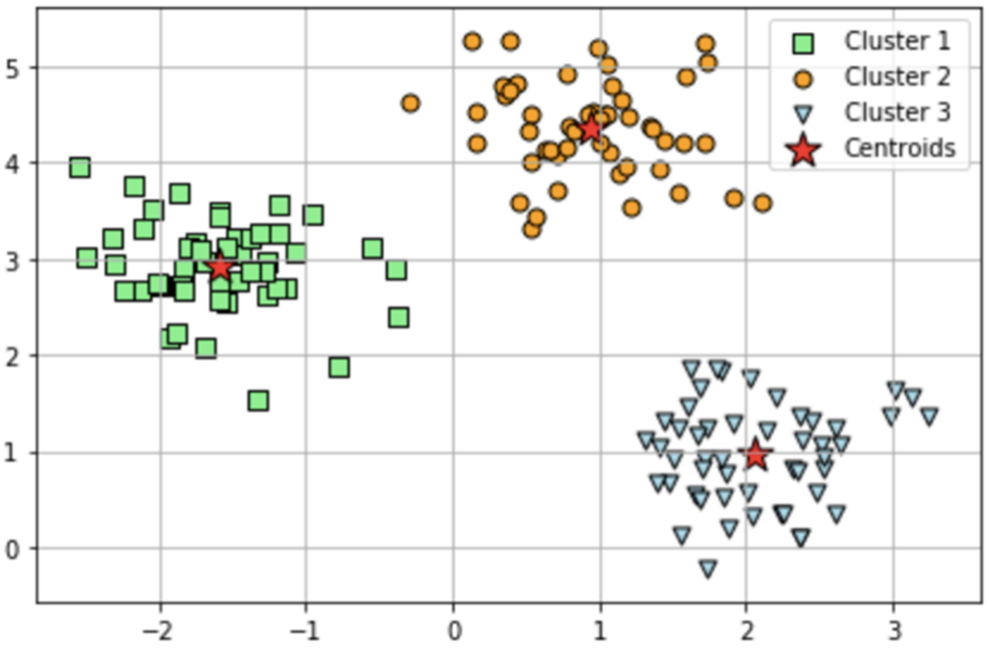


Figure 2.3: Example of K-Means clustering. Three clusters are automatically created once the centroid are selected.

- There are three different clusters represented by green, orange and blue color.
- Each cluster is created around a central point called cluster centroid or cluster center.

As previously exposed, K-means is a prototype-based clustering algorithm, and this is based on one of the following:

- **Centroid-based clusters**: Each cluster built around a point which is termed as the centroid (average) of similar points with continuous features. K-means algorithm results in creation of centroid-based clusters.
- **Medoid-based clusters**: Each cluster built around a point, defined as the medoid which represents the point that minimizes the distance to all other points that belong to a particular cluster.

2.3.1 Key steps of K-Means clustering

The following represents the key steps of K-means clustering algorithm:

- Define number of clusters, K, which need to be found out. Randomly select K cluster data points (cluster centers) or cluster centroids. The goal is to optimise the position of the K centroids.
- For each observation, find out the euclidean distance between the observation and all the K cluster centers. Of all distances, find the nearest distance

between the observation and one of the K cluster centroids (cluster centers) and assign the observation to that cluster.

- Move the K-centroids to the center of the points assigned to it.
- Repeat the above two steps until there is no change in the cluster centroids or maximum number of iterations or user-defined tolerance is reached.

2.3.2 Objective function to optimize

K-means clustering algorithm is an optimization problem where the goal is to minimize the within-cluster Sum of Squared Errors (SSE). At times, SSE is also termed as cluster inertia. SSE is the sum of the squared differences between each observation and the cluster centroid. At each stage of cluster analysis, the total SSE is minimised with $SSE_{total} = SSE_1 + SSE_2 + SSE_3 + SSE_4 + ... + SSE_n$. The below represents the objective function which needs to be minimized:

$$SSE = \sum_{i=1}^{n} \sum_{j=1}^{k} \mathbf{w}^{(i,j)} \left\| \mathbf{x}^{(i)} - \mu^{(j)} \right\|_2^2 \tag{2.7}$$

2.3.3 How to find optimal value of K?

The technique used to find the optimal value of K is to draw a reduction in variation vs number of clusters (K) plot. Alternatively, one could draw the squared sum of error (SSE) vs number of clusters (K) plot (Figure 2.4). Here is the diagram representing the plot of SSE vs K (no. of clusters). In the diagram below, the point representing the optimal number of clusters can be also called as elbow point. The elbow point can be seen as the point after which the distortion/cluster inertia/SSE start decreasing in a linear fashion [4].

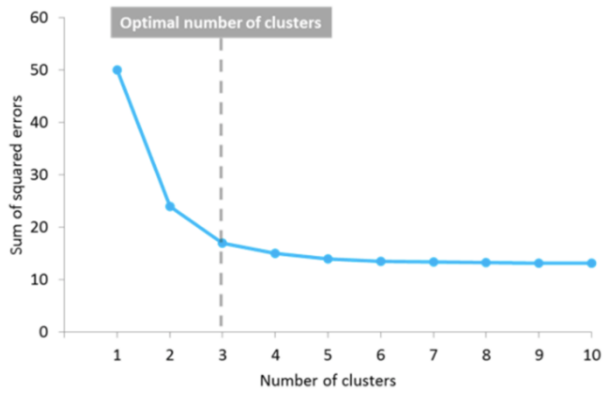


Figure 2.4: The SSE is used to find the optimal number of cluster.

2.3.4 Silhouette Score

Another technique to determine the optimal number of clusters for an unsupervised learning technique is the silhouette algorithm.

Assuming that the data has already been clustered into k clusters by a clustering technique, then for each data point, we define the following:

- *C(i)*: the cluster assigned to the ith data point.
- $|C(i)|$: the number of data points in the cluster assigned to the ith data point.
- *a(i)*: average intra-cluster distance i.e the average distance between each point within a cluster.
- *b(i)*: average inter-cluster distance i.e the average distance between all clusters.

Silhouette coefficient or silhouette score is a metric used to calculate the goodness of a clustering technique. Its value ranges from -1 to 1.

- 1: means clusters are well apart from each other and clearly distinguished.
- 0: means clusters are indifferent, or we can say that the distance between clusters is not significant
- -1: means clusters are assigned in the wrong way.

The silhouette score is computed using the following formula:

$$s(i) = \frac{b(i) - a(i)}{max[a(i), b(i)]} \tag{2.8}$$

The average silhouette is determined for each value of k. For the value of k, which has the maximum value of *s(i)*, it is considered the optimal number of clusters for the unsupervised learning algorithm [5].

2.4 DBSCAN

DBSCAN is a clustering algorithm that defines clusters as continuous regions of high density and works well if all the clusters are dense enough and well separated by low-density regions.

In the case of DBSCAN, rather than guessing the number of clusters, two hyperparameters are defined: epsilon and minPoints:

- **epsilon (ε)**: a distance measure that will be used to locate the points/to check the density in the neighborhood of any point.
- **minPoints (n)**: the minimum number of points (a threshold) clustered together for a region to be considered dense.

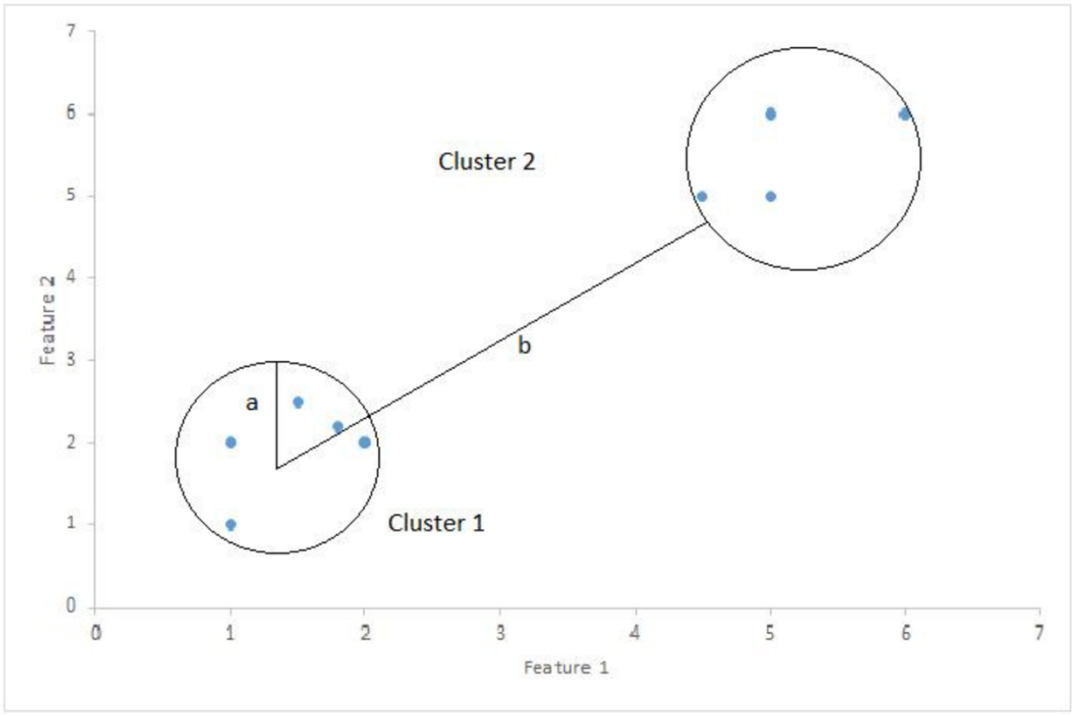


Figure 2.5: Example of clustering in which the clusters are clearly distinguished.

2.4.1 How does the DBSCAN Algorithm create Clusters?

Algorithms start by picking a point (one record) **x** from the dataset at random and assign it to a cluster 1. Then it counts how many points are located within the ε (epsilon) distance from **x**. If this quantity is greater than or equal to minPoints (n), then considers it as core point, then it will pull out all these ε-neighbours to the same cluster 1. It will then examine each member of cluster 1 and find their respective ε-neighbours. If some member of cluster 1 has n or more ε-neighbours, it will expand cluster 1 by putting those ε-neighbours to the cluster. It will continue expanding cluster 1 until there are no more examples to put in it.

In the latter case, it will pick another point from the dataset not belonging to any cluster and put it to cluster 2. It will continue like this until all examples either belong to some cluster or are marked as outliers.

Three different instances/points, as shown in Figure 2.6, are part of DBSCAN clustering:

- **Core Point(x)**: data point that has at least minPoints (n) within epsilon (ε) distance.
- **Border Point(y)**: data point that has at least one core point within epsilon (ε) distance and lower than minPoints (n) within epsilon (ε) distance from it.
- **Noise Point(z)**: data point that has no core points within epsilon (ε) distance.

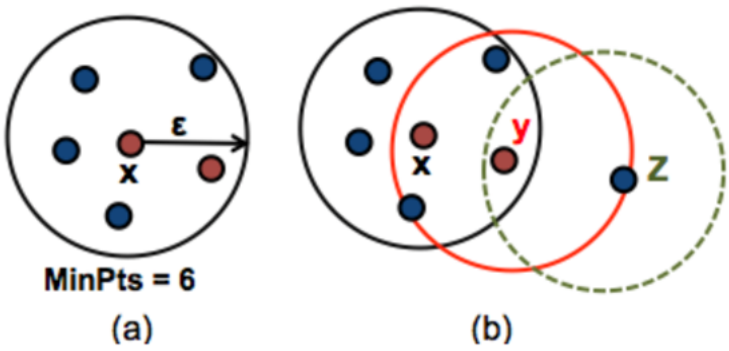


Figure 2.6: Example of the three types of data points in the DBSCAN algorithm.

2.4.2 Parameter selection

DBSCAN is very sensitive to the values of *epsilon* and *minPoints*. Therefore, it is important to understand how to select their values. A slight variation in these values can significantly change the results produced by the DBSCAN algorithm.

- **minPoints (n)**: As a starting point, a minimum n can be derived from the number of dimensions D in the data set, as $n \geq D + 1$. For data sets with noise, larger values are usually better and will yield more significant clusters. Hence, $n = 2D$ can be evaluated, but it may even be necessary to choose larger values for very large data.

- **epsilon (ε)**: If a small epsilon is chosen, a large part of the data will not be clustered. Whereas, for a too high value of ε, clusters will merge and the majority of objects will be in the same cluster. Hence, the value for ε can then be chosen by using a k-graph, plotting the distance to the $k = minPoints - 1$ nearest neighbor ordered from the largest to the smallest value. Good values of ε are where this plot shows an "elbow".

- **Distance function**: By default, DBSCAN uses Euclidean distance, although other methods can also be used (like great circle distance for geographical data). The choice of distance function is tightly linked to the choice of epsilon (ε) value and has a major impact on the outcomes. Hence, the distance function needs to be chosen appropriately based on the nature of the data set [6].

2.4.3 DBSCAN Vs K-means Clustering

K-means Clustering	DBSCAN
Distance based clustering	Density based clustering
Every observation becomes a part of some cluster eventually	Clearly separates outliers and clusters observations in high density areas
Build clusters that have a shape of a hypersphere	Build clusters that have an arbitrary shape or clusters within clusters.
Sensitive to outliers	Robust to outliers
Require no. of clusters as input	Doesn't require no. of clusters as input

Figure 2.7: Differences between K-Means and DBSCAN.

2.5 Feature selection

In order to perform a clustering on the case study considered, thus an analysis on commands, the algorithms need to have features. In the Natural Language Processing (NLP) the words, or the keywords in the case of commands, can be seen as a feature. But, usually they are not enough to have good results at the end of the clustering process.

The words can be used to create more features and improve the performance. A simple feature is the bag of word. A text is represented as the bag of its words, disregarding grammar and even word order, but keeping multiplicity of words. Another feature are the n-grams. A n-gram is a contiguous sequence of n items from a given sample of text or speech. For log commands the n-grams are formed by the keywords, the parameters and by removing special characters. Another popular feature in the NLP field is the TF-IDF (term frequency–inverse document frequency) which is used to reflect how important a word, or a n-gram, is to a document in a collection (corpus) [7]. The TF-IDF value increases proportionally to the number of times a word appears in the document and is offset by the number of documents in the corpus that contain it, which helps to adjust for the fact that some words appear more frequently in general. For logs the corpus is the set of commands, a document is a single line and the term is a keyword, a parameter or a n-gram. The term frequency (TF) of the term t in the document d is obtained by using the formula:

$$tf\,(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.9}$$

Where:

- $f_{t,d}$: raw count of t in d.
- $f_{t',d}$: number of words in d different from t.

The inverse document frequency (IDF) measures how much information a word provides in the corpus and its formula is:

$$idf\,(t, D) = log\frac{N}{|\{d \in D : t \in d\}|} \tag{2.10}$$

Where:

- N: total number of documents in the corpus, $N{=}|D|$.
- $\{d \in D : t \in d\}$: number of documents where the term t appears. If the term is not in the corpus the division by zero is avoided by adjusting the denominator to $1 + \{d \in D : t \in d\}$.

Finally the TF-IDF is:

$$tfidf(t, d, D) = tf(t, d) \cdot idf(t, D) \tag{2.11}$$

Other features can be used in machine learning algorithm for NLP which can be combined to create new ones according to the situation.

Chapter 3

Platforms and Technologies

The programming language used to reach the required aim is Python, which is usually note for its use in the machine learning field. Furthermore, has been used different libraries such as:

- scikit-learn;
- matplotlib;
- seaborn;
- pandas;
- numPy.

These are the main libraries used in the project. The log files used have all this following structure:

```
{'status': 'PASS',
'timestamp': '2021.05.01-12:40:39.726',
'enviroment': 'SIMULATION',
'profile_name': 'CoreSwerrCheck',
'user': 'side_art',
'batch_name': 'NBG_SVT_QUICK_PSS12X_SIM_EM_BATCH',
'response': '\nexit\n',
'sw_version': '',
'ne': None,
'testcase_step': 'Cleanup',
'cmd': 'exit',
'system': 'SIM_sa1010',
'tc_name': 'infra_CoreSwerr_Checker_Testcase-1',
'other': None,
'root': None}
```

The key fields considered relevant for the clustering are `profile_name`, `tc_name` and `cmd`. Obviously other commands would have been useful to improve the final result but have been chosen just some of them to reach the right tradeoff between weight and speed of execution.

To parse the log a .json file has been used in which is defined the hierarchical structure followed to build a command.

Thanks to the .json file it is possible to distinguish the keywords from the values representing the parameters.

In order to have a tidy work has been preferred to have four different python scripts:

- **clustering.py**: this is the usual "main" in which parameters are defined, dictionaries containing already parsed command are loaded and the clustering process is executed. The results are saved in a file after the clustering ends.
- **read_log.py**: it parses the log file and serializes the dictionaries in a file (.pkl format) avoiding the reading every time the clustering is executed.
- **load_workspace.py**: clustering results can be loaded to be further processed. This script also makes various plots to visualize the data.
- **functions.py**: it contains basically all the functions used in the other scripts.

3.1 Parameters

To set up the parameters, shown in Figure 3.1, used by the clustering algorithm the following variable are defined:

- `CLUSTERING_ALG`: this variable allows to choose the clustering algorithm. Currently the K-Means and the DBSCAN are implemented.
- `MIN_VALUE`: represents the minimum number of clusters for K-Means or the minimum ε for DBSCAN.
- `MAX_VALUE`: represents the maximum number of clusters for K-Means or the maximum ε for DBSCAN.
- `STEP`: is the step between a value of k or ε and the next one.
- `MIN_NGRAMS, MAX_NGRAMS`: minimum and maximum number of items in a contiguous series. In this case an item is represented by a keyword or a parameter of a command.
- `EVAL_METHOD`: the evaluation method used to measure the performance of the algorithm. The main techniques are the silhouette score and the elbow method.
- `FILES`: the files that will be analyzed by the algorithm.
- `FIELDS`: lines' fields in the log file that will be used as feature to group the commands of the log.

- `DIM_RED_METHOD`: this variable is used to set the dimensionality reduction method to reduce the number of features and to avoid strongly correlated features. The method used are PCA, SparsePCA and NMF.

```
[ ]  1 # Parameters
     2 CLUSTERING_ALG = 'K-Means'
     3 if CLUSTERING_ALG == 'K-Means':
     4     MIN_VALUE = 2
     5     MAX_VALUE = 10
     6     STEP = 1
     7 elif CLUSTERING_ALG == 'DBSCAN':
     8     MIN_VALUE = 0.1
     9     MAX_VALUE = 1
    10     STEP = 0.1
    11 MIN_NGRAMS = 1
    12 MAX_NGRAMS = 5
    13 EVAL_METHOD = 'Elbow Method'
    14 FILES = ['2021-05-01', '2021-06-28', '2022-01-03']
    15 FIELDS = ['profilename', 'testcase', 'profilename_testcase']
    16 DIM_RED_METHOD = 'PCA'
```

Figure 3.1: Parameters used to set up the clustering algorithm.

3.2 Dictionaries

The dictionaries contain the already parsed commands grouped according to the fields `profile_name`, `tc_name` or a combination of both. These dictionaries are created after the parsing of a log file and stored in a PKL file. Reading an already structured file is much faster than reading the entire log and it's convenient if many executions of the clustering algorithm are necessary. The dictionaries have two fields:

- **Key**: is the name of the field.
- **Value**: contains a list of commands.

3.3 Results

At the end of the clustering process the results are saved in a file for each dictionary's key. This organization allows to perform an analysis of results for the entire log or for a single field. The files are loaded as a Pandas `DataFrame` that contains all the necessary information to process and plot the data:

- **Input data**: the original parsed commands organized in a Pandas `Series`.
- **TF-IDF data**: matrix of TF-IDF values obtained from the input data.
- **Features**: features for the transformation.
- **Dimensionality reduction**: contains an object that represents the dimensionality reduction method used and its parameters.
- **Transformed components**: the transformed components are the new variables obtained by applying a dimensionality reduction method to the TF-IDF data. This is done to reduce the computational complexity and to consent a visualization of the clusters in a reduced dimensional space.
- **Labels**: is an array containing the label of the cluster for each command.
- **Best ML model**: is an object that contains the trained model considered as the best ML model for a particular field.
- **Other models**: any other model used during the clustering.
- **Silhouette score**: every model has an associated silhouette score. If the silhouette score computation is skipped the value of this column is `None`.
- **Variance**: every model has also a variance used to find the knee in the elbow method. For the DBSCAN the elbow method does not use the variance, it uses the k-nearest neighbors to find a knee. In the second case the Variance column is `None`.
- **Knee**: an object containing the knee's characteristics. If a plot has no knee this object has value `None`.

Everything is organized into a workspace as can be seen in Figure 3.2.

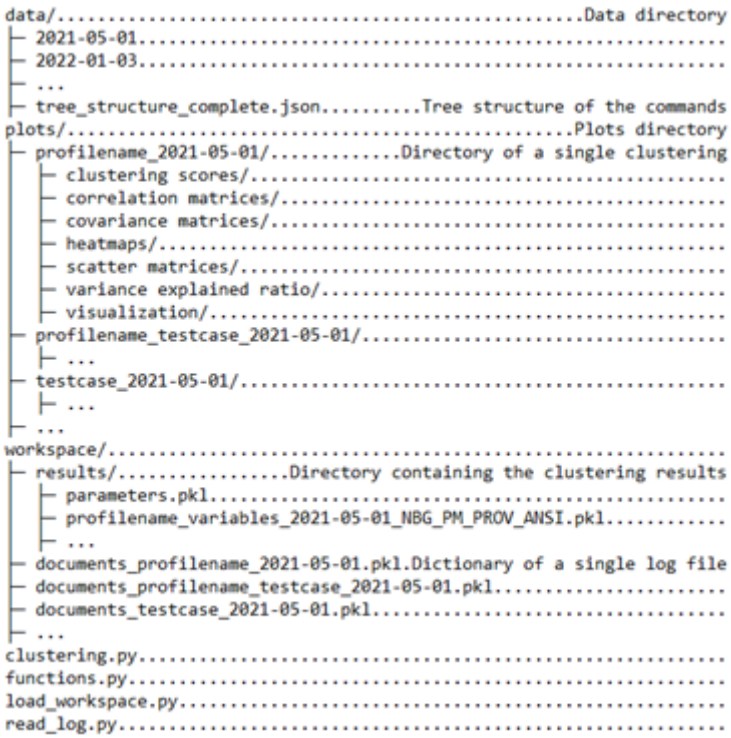


Figure 3.2: Workspace used.

An organized workspace allows to keep separated data and results coming from different configurations of the algorithm.

Chapter 4

Proposed Solution

Data pre-processing is a fundamental phase to implement before the execution of the clustering algorithm. Firstly, parsing of the log file is needed. During this step the field to analyze is chosen, and a tree, containing the structure of the commands, is used to distinguish keywords from parameters. Parameter's values can be replaced by a '*' or by their own name. System commands are trivial, so they are removed during the parsing.

The value of K used has been chosen considering two different methods previously mentioned (elbow method, silhouette score). Firstly, to choose the K value, elbow method is applied, while the second method is used just to give a score to clustering. In these cases, K varies between 2 and 10. Has been chosen this range to avoid the creation of too many clusters and because has been seen nice behavior in this interval.

Number of features has been increased using n-grams, where the range considered goes from 1 to 5.

This method has been used on three different fields:

- `profile_name`
- `tc_name`
- both

4.1 Field `profile_name`

Dataset is divided into smaller sets according to the name of the profile. The clustering is executed for every set. Here are presented the charecteristics of the biggest set.

In Figure 4.1 can been seen, a part of the main diagonal, that several features are strongly correlated. This situation increases algorithm's computational complexity and, at the same time, decreases its performance. To avoid this issue, PCA is used to reduce dataset's dimensionality.

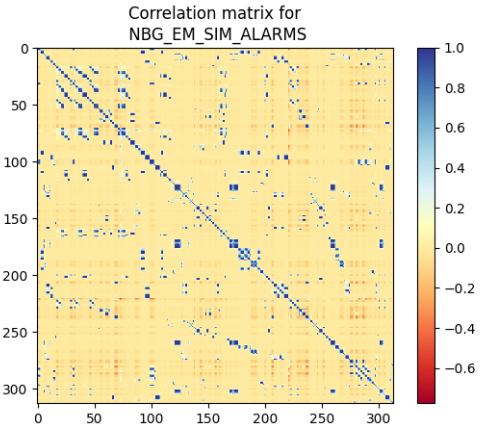


Figure 4.1: Heatmap representing the correlation between the features of the profile `NBG_EM_SIM_ALARMS`.

The number of principal components is computed using PCA such that the amount of variance that needs to be explained is greater than 95%. The correlation between the original features and the principal components is shown in a heatmap which proves that not all features are important for the clustering.

Before using PCA there are more than 300 features. With PCA, the principal components are computed, and a heatmap (Figure 4.2) is used to visualize the correlation between each component and the original features. This is useful to understand which feature influences the most a component.

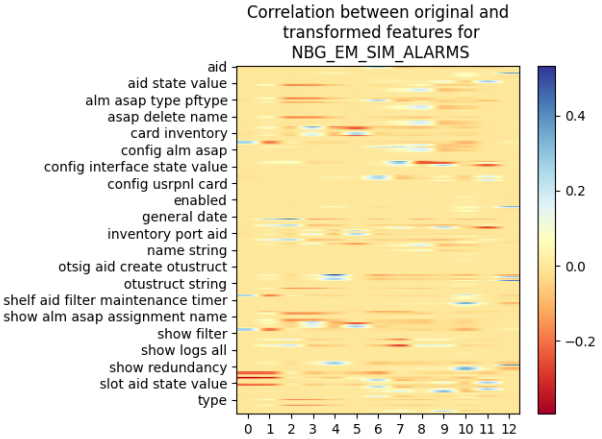


Figure 4.2: Only a few original features are shown. It can be seen that a small part of the 300 features is important.

The new correlation matrix in Figure 4.3 is computed to show the correlation between new components. In this case there is no strong correlation.

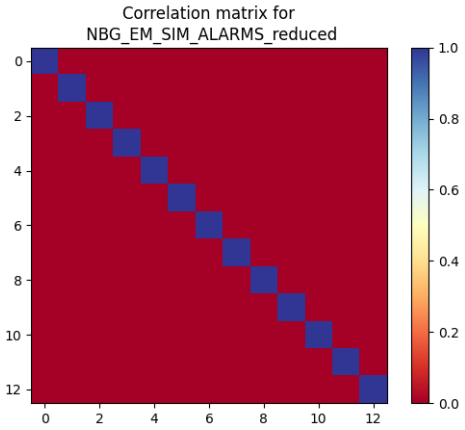


Figure 4.3: Correlation matrix for the principal components.

Figure 4.4 shows how much each principal component explains the entire dataset (blue bars) and, in addition, the cumulative sum of the variance (red line).

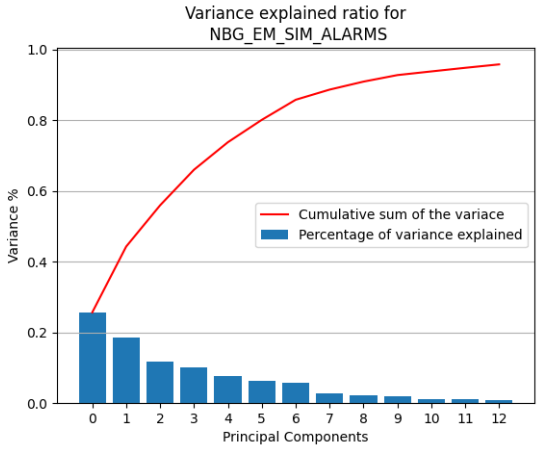


Figure 4.4: Variance explained ratio for every principal component.

Scatter matrices can be used to visualize clusters in more than 3 dimensions; in most cases there are too many dimensions to be represented, even after the reduction of the dimensionality. For the issue previously explained they are omitted due to their size.

4.2 Field `tc_name`

The same procedure can be applied to `tc_name` field. By doing this, the clustering is executed on the same dataset, but with a different organization.

This testcase has more features than the profile analyzed before, as can be seen in the correlation matrix (Figure 4.5). With the use of PCA, 18 principal components out of more than 350 remain.

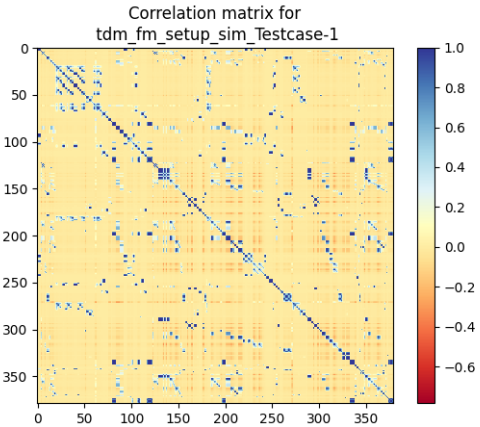


Figure 4.5: Heatmap representing the correlation between the features of the `profile tdm_fm_setup_sim_Testcase-1`.

The heatmap in Figure 4.6 shows a situation that is similar to the previous one

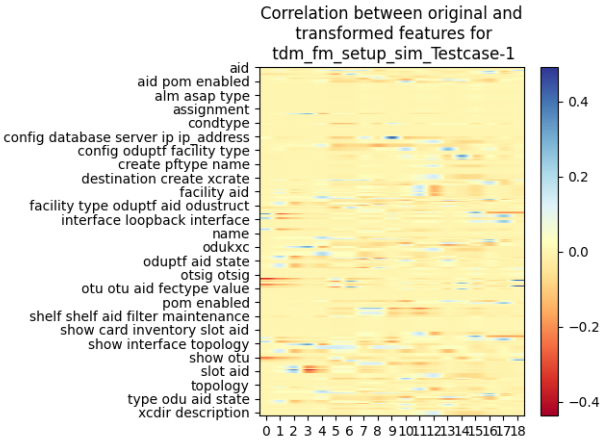
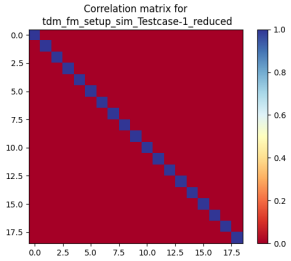
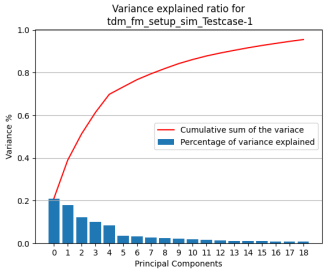


Figure 4.6: Also in this case a few features are relevant for the clustering.

The new correlation matrix and the percentage of variance explained also give similar results shown in Figure 4.7. In this situation more components are necessary to explain at least the 95% of the variance.



(a) Correlation between features and principal components.



(b) Variance explained ratio.

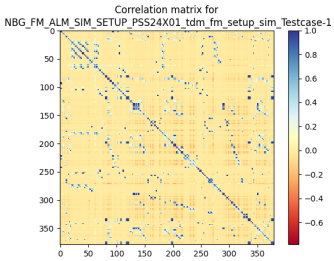
Figure 4.7: Result of the PCA.

4.3 Both fields merged

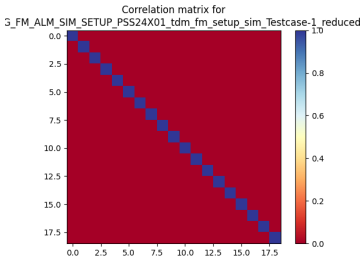
Merging the fields `profile_name` and `tc_name`, the dataset can be analyzed in a more specific way because smaller sets are obtained. The commands contained in a set are shared by a profile and a test case.

It can be noted that, usually, for every profile, there are more testcases, and each testcase contains a particular category of commands. The plots below refer to the combination between the profile `NBG_FM_ALM_SIM_SETUP_PSS24X01` and the testcase `tdm_fm_setup_sim_Testcase-1`.

First of all, in Figure 4.8, there is a comparison between the two correlation matrices. As usual, the first one contains a lot of features and most of them are strongly correlated. A reduction of dimensionality can solve the problem and lead to the situation described by the second heatmap.



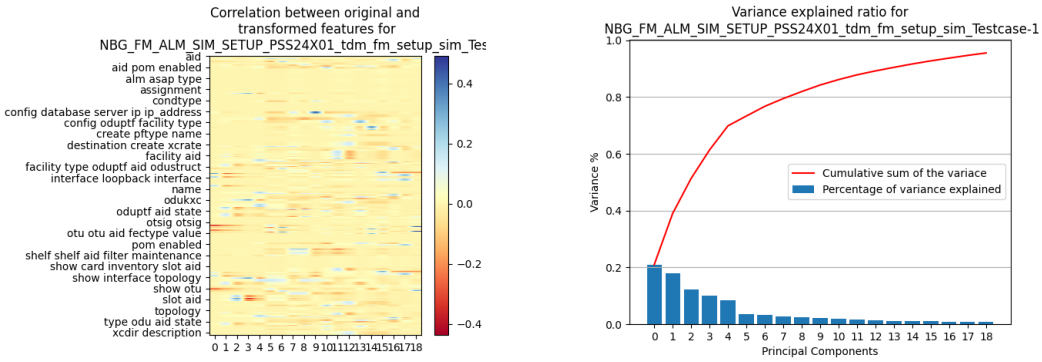
(a) Correlation between features.



(b) Correlation between the principal components after the PCA.

Figure 4.8: Comparison between the two correlation matrices.

The PCA returns 18 principal components and again, not every original feature is useful to describe the entire dataset (Figure 4.9).

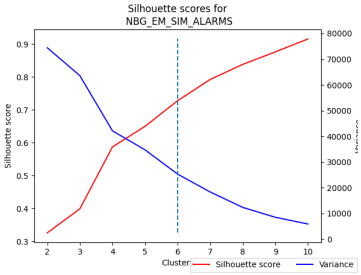


(a) Correlation between features and principal components.

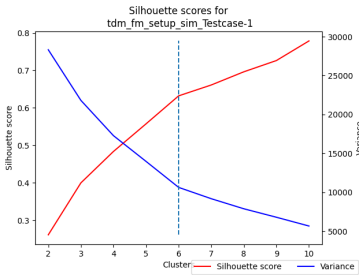
(b) Variance explained by the 18 principal components.

Figure 4.9: Comparison between the two correlation matrices.

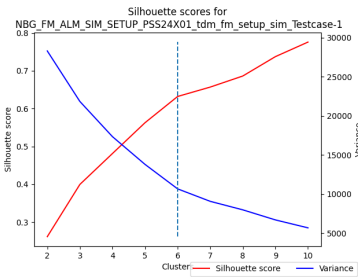
After this preliminary analysis, the K-Means can be executed. For every subset, different values of K are used to find the best one. The silhouette score alone is a good metric, but it increases during every iteration so there could be the problem of having too many clusters. To avoid this, the elbow method is used to select a reasonable number of clusters.



(a) Profile `NBG_EM_SIM_ALARMS`.



(b) Testcase `tdm_fm_setup_sim_Testcase-1`.



(c) Merging of profile `NBG_FM_ALM_SIM_SETUP_PSS24X01` and testcase `tdm_fm_setup_sim_Testcase-1`.

Figure 4.10: The red line represents the silhouette score, meanwhile, the blue line is the value of the variance. The dashed line is the elbow of the plot of the variance.

In Figure 4.10 there are plots relative to the three cases previously analized. The y axis represent the silhouette score and the variance.

Using the K computed by the elbow method, the silhouette score for the three cases is between 0.6 and 0.7. A bar chart (Figure 4.11) is used to represent the commands in a cluster. Using this kind of visual representation it is easy to see that clusters are very unbalanced. In `NBG_EM_SIM_ALARMS` profile, cluster 2 contains more than 25000 samples, but `show slot <slot-aid>` command is the only one in it. The same is true for cluster 1, which contains only two different commands.

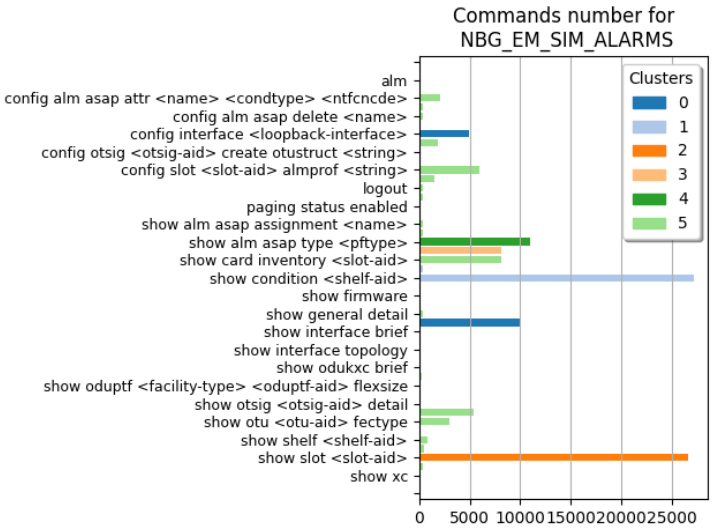
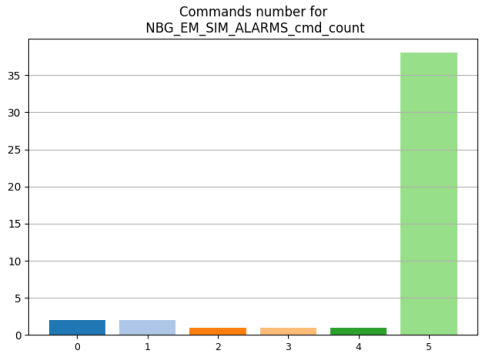


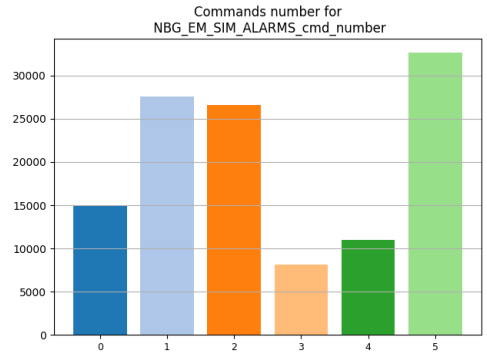
Figure 4.11: Bar chart representing the distribution of the commands in every cluster of the profile `NBG_EM_SIM_ALARMS`.

By using histograms in Figure 4.12, the distribution of commands in the different clusters is more clear. With this clustering method the commands are unevenly divided. There is a cluster with more commands than the other ones and some clusters with just one type of command. In clusters with more than one command, the algorithm groups the samples according to their syntactic similarity.

The same happens for the testcase and the combination of a profile name and a test case. Bar charts in Figure 4.13 and 4.14 show that the test case `tdm_fm_setup_sim_Testcases-1` and the union of the latter with the profile `NBG_FM_ALM_SIM_SETUP_PSS24X01` share all the samples. In fact, the profile and the testcase coincide, so analysis will be done just for one of them.



(a) Number of samples per cluster.



(b) Number of commands per cluster.

Figure 4.12: The histograms represent the number of commands per cluster and the number of different types of commands per cluster.

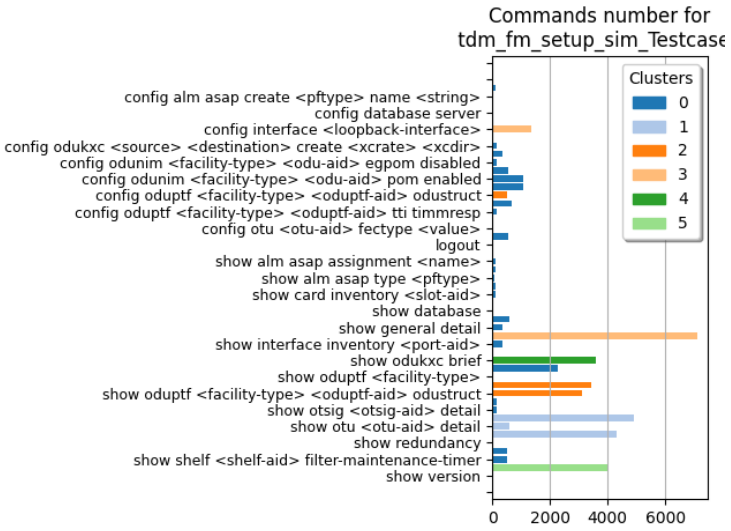


Figure 4.13: Bar chart representing the distribution of the commands in every cluster of the testcase `tdm_fm_setup_sim_Testcase-1`.

Considering the testcase `tdm_fm_setup_sim_Testcase-1`, the same problem as before arises, as can be seen in Figure 4.15.

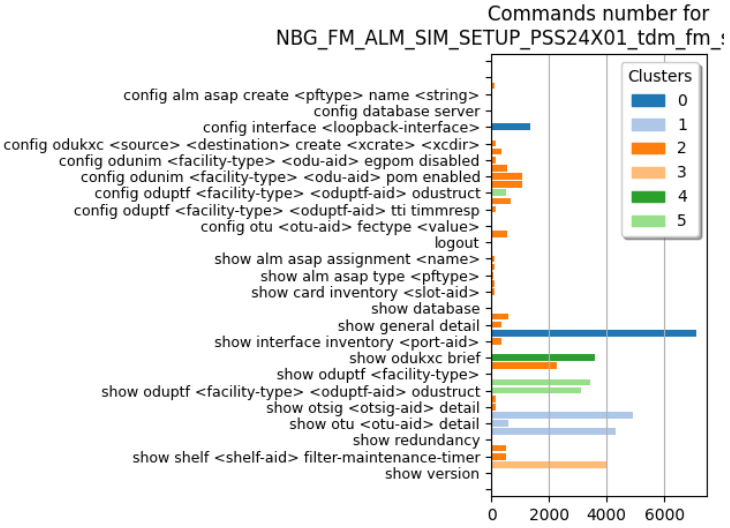
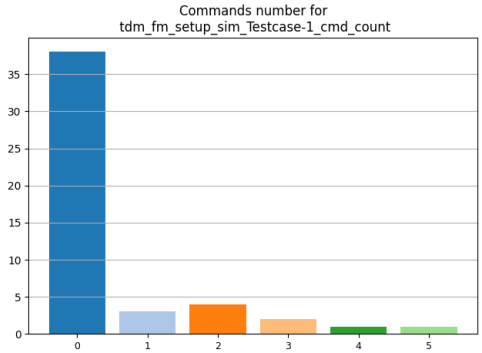
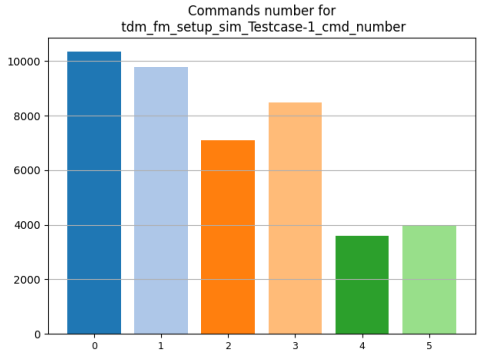


Figure 4.14: Representation of the clusters in the union of a profile and a testcase.



(a) Number of samples per cluster.

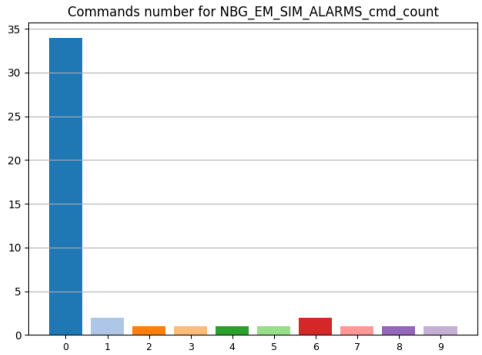


(b) Number of commands per cluster.

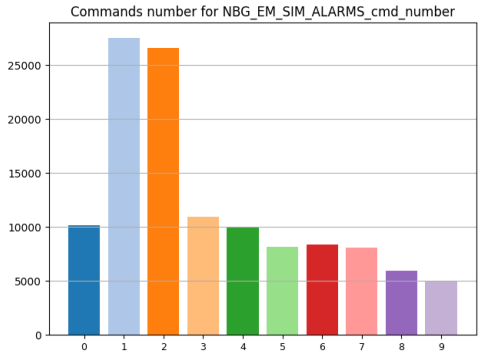
Figure 4.15: This second clustering leads to a similar situation as before.

Changing parameters, silhouette score can be increased, and, with k=10, it reaches the value of 0.9. By doing this on the same profile as before, the clustering is still unbalanced and it is done by similarity.

Changing dataset, clustering method or dimensionality reduction method leads to similar results (Figure 4.17). Other clustering methods (OPTICS, Restricted Boltzmann Machines (RBM)) or decomposition methods (Singular Value Decomposition (SVD), Independent Component Analysis (ICA)) could be applied, but most of them require too much time or memory. The same true is for datasets with more than 400000 samples, so the analisys of the entire log file is not currently possible.

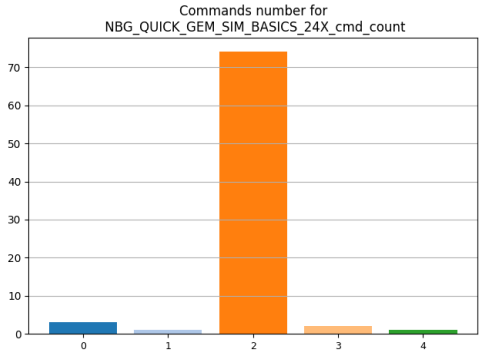


(a) Number of samples per cluster.

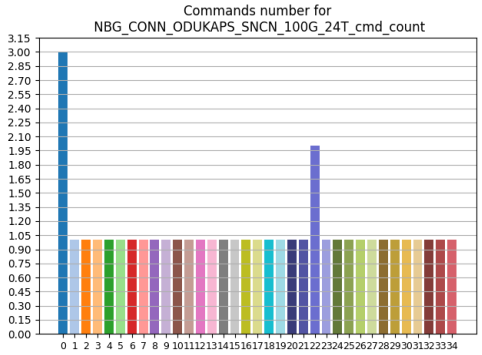


(b) Number of commands per cluster.

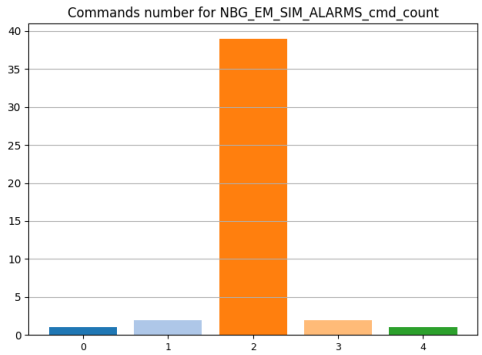
Figure 4.16: For k=10 the problem is not solved.



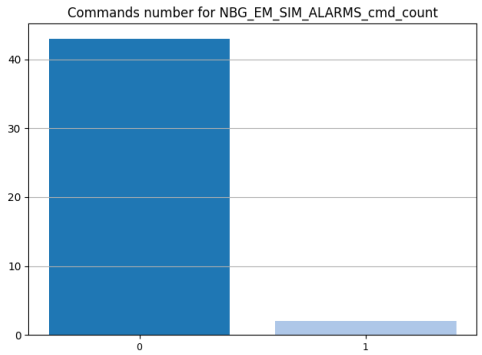
(a) Number of samples per cluster using a different dataset.



(b) Number of commands per cluster using DBSCAN.



(c) Number of samples per cluster using SparsePCA as decomposition method.



(d) Number of samples per cluster using NMF as decomposition method.

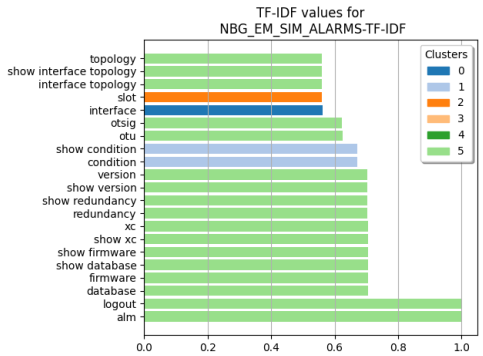
Figure 4.17: Using different method to execute the clustering or different dataset gives results that are similar to the previous ones.

Clustering results can be improved by adding more features that are not present in the provided datasets. For instance, the description of commands could be useful to cluster the log according, not only to the syntactic similarity, but also to the

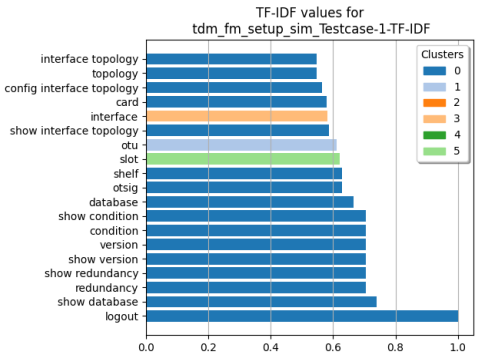
semantic similarity (for example `exit` and `quit` have the same semantic meaning so they will be part of the same cluster).

4.4 TF-IDF in different cases

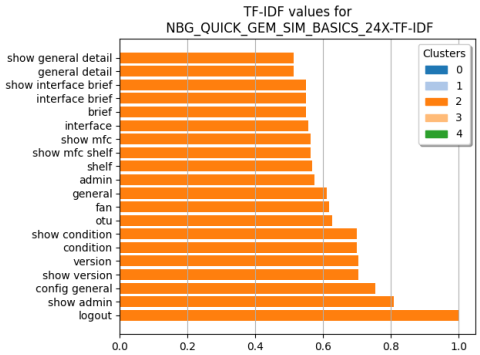
The last value to analyze is the TF-IDF. In Figure 4.18 are represented the 20 n-grams with the higher TF-IDF in different situations. They all, or almost all, belong to one cluster as expected, because the majority of the samples are all in one cluster. Using these histograms, and other obtained from other profiles or testcases, it can be noted that some keywords appear less frequently across the database. For example `logout`, `show`, `condition` and `otu` are very common in bar charts.



(a) Profile `NBG_EM_SIM_ALARMS`



(b) Testcase `tdm_fm_setup_sim_Testcase-1`



(c) Different dataset.

Figure 4.18: Values of TF-IDF in different cases.

By adding other features to the dataset, such as the semantic meaning relative to keywords or parameters, a very accurate clustering can be done and, in addition, every profile name or testcase can be described automatically and in less time.

Conclusion and Future Development

Machine learning can help with the automatic log analysis. A properly setup for clustering algorithms can provide good results in a relatively small time, even with big files. The advantage of such an algorithm is the possibility of clustering different log files, with different structures, only by changing parsing and the model's hyperparameters. The algorithm presented is a starting point for log commands clustering and many additions can be done to refine it:

- Implement new clustering methods.
- Implement new dimensionality reduction methods.
- Use new datasets to train models.
- Use commands' documentation to apply NLP techniques and improve performance.
- Apply algorithms to the entire dataset (due to lack of memory it was not possible).

Bibliography

[1] Weixiang Li. Automatic log analysis using machine learning : Awesome automatic log analysis version 2.0. 2013.

[2] Wikipedia contributors. Parsing — Wikipedia, the free encyclopedia, 2022. [Online; accessed 30-January-2022].

[3] OpenCV. Introduction to principal component analysis (pca), 2022. [Online; accessed 30-January-2022].

[4] K-means clustering explained with python example, 2022. [Online; accessed 30-January-2022].

[5] Silhouette coefficient - validating clustering techniques, 2022. [Online; accessed 30-January-2022].

[6] Dbscan algorithm | how does it work?, 2022. [Online; accessed 30-January-2022].

[7] Wikipedia contributors. Tf–idf — Wikipedia, the free encyclopedia, 2022. [Online; accessed 30-January-2022].