Final Year Project Report

**Full Unit – Final Report**

# Clustering Android Application Behaviour

Ghadah Alshehri

A report submitted in part fulfilment of the degree of

**BSc (Hons) in Computer Science**

**Supervisor:** Lorenzo Cavallaro


Department of Computer Science
Royal Holloway, University of London

March 28, 2018

# Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been acknowledged.

Word Count: **14316**

Student Name: Ghadah Alshehri

Date of Submission: 28/3/2018

Signature: GhadaShehri

# Table of Contents

# Abstract

Android platforms are evolving more widely every year. With such an expansion, the number of applications in their Marketplace (Google Play) is accordingly increasing. As a result, Cybercriminals are more concentrated on constructing malicious applications that would affect the user. In this paper, using behavioural traces produced by CopperDroid System, as a starting point in analysing malicious behaviour. Extracting features from these profiles, and using that, as our input for the clustering algorithm, will allow us to create clusters of malicious behaviour. Each cluster, will represent a malware family, that shares a common set of behaviours among its members.

The structure of this paper is as follows, in section 1, we Included a brief introduction on Malware and its notable presence nowadays, especially in Android Marketplace. In section 2, we focused on Android's structure and clarified different approaches in which Android's Malware gets installed and activated, we also, highlighted CopperDroid's mechanism in producing behavioural profiles, and the several methods we applied in creating our feature sets. In section 3, we discussed the difference between supervised and unsupervised learning methods, we demonstrated Principal Components Analysis, and the way it functions. In section 4, we introduced clustering as a common technique in unsupervised learning, we also elaborated on 3 popular clustering algorithms, explaining the advantages and disadvantages of each algorithm. In section 5, we illustrated the different experiments we carried out on the extracted features, in order to evaluate the quality and performance of DBSCAN algorithm. In section 6, we discussed the limitations and difficulties we encountered, and the distinct methods we used to resolve them. In section 7, we elaborated on plagiarism as a professional issue, and how serious it can be. Finally, a included a brief section on future work and areas of improvement.

# Project Specification

**Aims**

To implement clustering for recorded behaviour traces of Android applications towards distinguishing benign from malicious behaviour (malware) [38].

**Background**

Much of the relevant behaviour of an Android application is visible by its interactions with the operating system and runtime. The CopperDroid Android analyser runs an Android app and records all, such interactions (system calls) and their arguments in a long trace. Different combinations of system calls and arguments correspond to different types of behaviour. With a suitable model, clustering these behaviours should exhibit related classes of behaviour and ideally allow to expose clusters of malicious behaviour [38].

**Early Deliverables**

1. A report describing methods for feature selection and model construction

2. A report giving an overview of clustering and describing three different clustering methods

3. Proof of concept implementation: application of DBSCAN with a set of simple features of behaviour traces [38]

**Final Deliverables**

1. Program for extracting complex features from traces and clustering the resulting data

2. Describe and justify the features and clustering methods used

3. Critically compare and visualise experimental results with different features and different clustering algorithms

4. Evaluate accuracy of clustering-based malware detect [38]

**Overall Result**

Extracting significant features from behavioural profiles, and pre-processing data, in order to obtain, feature sets of fine granularities, that entirely represents the initial profiles. Applying clustering algorithm on that feature set, and being able to discover a nearly ideal partitioning, which will obviously allow us to automatically generate signatures, that will be used in detecting Android malware, and distinguishing them from benign applications.

# Chapter 1: **Introduction**

Nowadays, Android has become one of the leading mobile operating system platforms, it recently reached two billion active devices on monthly basis. Having an extremely user-friendly platform, along with several modern technologies and features, lead to this success and popularity [13]. A report from Lookout indicates that, Android marketplace is growing by a factor of three when compared with Apple's app store, which specifies how fast its growing among its competitors [36].

In a recent study conducted by Kaspersky, which demonstrates the evolution of malware in 2016, *"From the beginning of January till the end of December 2016, Kaspersky Lab registered nearly 40 million attacks by malicious mobile software and protected 4,018,234 unique users of Android-based device*s" [33]. Undoubtedly, with such huge growth in the number of active devices, it is apparent that it will attract the attention of several malware authors targeting these devices [2]. Considering, the ultimate technique to affect Android devices, is by inserting well-crafted malicious applications in their marketplace (Google Play), as it is the absolute way in which users can obtain a vast amount of applications [2].

The reasons behind the increased amount of Android malware existing in their marketplace, is due to insufficient amount of checking an application gets before being uploaded to the marketplace [5]. For example, considering the strict approach that apple application store follows before adding any new application, effectively reduces the percentage of malware found in their store [31]. Another reason to consider, is Android permitting users to install applications form third-party sources, which are mainly used to distribute malware [13].

## 1.1 Malware

Malicious software (Malware) is considered to be a significant threat that opposes on the Internet and devices these days. Malware can appear in many extensive forms and variations, such as viruses, worms, Trojan horses, botnets, and rootkits [13]. Its main objective, is to primarily act against the user's interest. Thus, it can perform many harmful activities, including, spam, Denial of Service (DoS), gaining access to user's information. As well as, causing some hardware related problems by reducing the efficiency of the device which can be achieved, for example, by draining the battery [13]. Moreover, Malware can spread itself by exploiting vulnerabilities or social engineering techniques that deceive both the user and operating system to perform previously mentioned activities, for example, downloading a malicious application and running it, can eventually lead to compromising user's login information, and committing fraud, or it can infect every name in the user's phonebook [5,22].

Analysing malware samples, is a crucial step, to determine the actions performed and how much these actions affect the underlying system. In addition, the resulting information is valuable, since it eases the process of writing removal procedures and producing signatures for each sample [5].

# Chapter 2: **Android**

Our aims are to be able to accurately cluster Android malware samples into different families, so that each sample contained in a certain family, shares a set of common behaviours with it ancestors. In order to achieve such a result, a number of steps must be undertaken. Starting with dynamically analysing the behaviour of the program and creating a corresponding behavioural profile, that reflects the actions performed by a certain malware sample. Therefore, we will be using CopperDroid to preform dynamic analysis and generate accurate behavioural profiles, Thus the 1st and 2nd steps are achieved by CopperDroid. Afterwards, parsing those profiles to determine a set of common features, that exists in all the samples in our dataset, and using those features in our last step to preform clustering, which can be described as follows (see Figure 1):

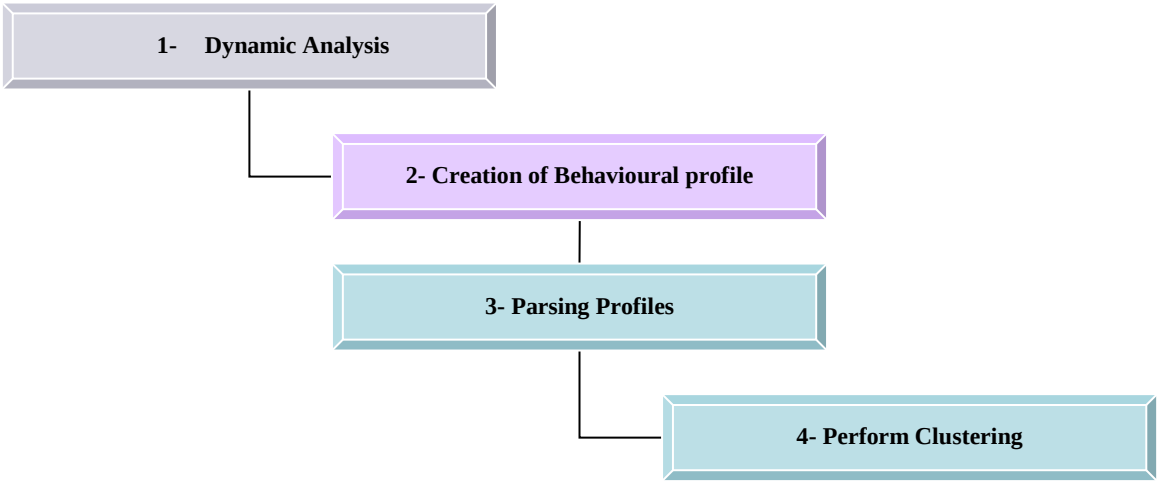


*Figure 1. The overall process of Clustering Android Malware.*

In the next section, we explained Android's underlying structure, and its security model, we also highlighted the distinct techniques in which Android's malware gets installed and activated. And finally, we introduced CopperDroid's main functionality and its distinct approach in analysing samples, and how accurate it could be, followed by a section on feature extraction procedures we performed on our dataset, to contain valuable information that best describes a malware behaviour.

## 2.1 Android's Structure

Initially, Android applications are written in java language, and then deployed to Android Packages archives, known as APKs'. Each APK contains several components, which can be described as a program in a whole. Each component is designed to accomplish a certain task, for example, user-interface interactions. In every APK file there must be a manifest included, which contains a list of components, permissions, software, and hardware features an application has and probably will use. Interestingly, the applications manifest can reveal some information about whether the application is malicious or not [26].

Each android application runs in a separate userspace process, which is assigned a unique user and group ID. Thus, each application has an isolated environment. However, applications can communicate thoroughly using well-defined application programming interface (API) [26]. Inter-process communication (IPC) and remote procedure calls (RPC), are two main mechanisms in which processes interact. And by using Binder protocol, a process is allowed to invoke several methods of another object (i.e. services), through synchronous calls [26].

As for Android's security model, it extremely relies on permission-based techniques. Their security model consists of 130 permissions, which control the access to different resources. For example, when installing a new application, the user will be given the chance to approve or reject the set of permissions an application requests. Since the users may not be fully aware of the consequences of approving a certain permission, they might make incorrect decisions when allowing certain permissions to unreliable applications [36].

Static and dynamic analysis are the two approaches followed in order to analyse any type of malware. Considering the static and dynamic information the applications contain, is not the only information that can be found helpful. As suggested by Wu et al. [36] inspecting the author information of an application, can provide an overview on whether the repackaged application is the updated version of the genuine application, or it is the misleading application that contains malicious patterns. Another aspect to consider, is the information provided on the Android's marketplace by inspecting the application's description, number of downloads, and category, can help in constructing patterns [36].

In order to get a rough idea on the differences between benign and malicious applications, we included a table, which consists of the number of times a certain component is included in the application Manifest. This comparison is made over a set of benign and malicious applications [36].

**Component Average Usage**

| Component Name | Benign Apps | Android malware |
|---|---|---|
| Permission | 4.67 | 11.27 |
| Activity | 9.97 | 3.60 |
| Service | 0.49 | 1.10 |
| Receiver | 0.68 | 1.73 |
| Provider | 0.10 | 0.17 |

*Figure 2. Comparison between benign and malicious applications.*

However, as seen on the table above, that overall malicious applications include higher results in each of the permission, service, and receiver components, unlike benign applications [36].

## 2.2 Malware Installation

Android's malware primarily adopts three different ways to get installed onto users' devices, which are repackaging, update attack and drive-by download. Yet, these are not the only techniques a certain malware adopts, in fact, different variants can use other techniques to lure users. Malware authors mainly use repackaging technique to attach malicious payloads into legitimate applications [37]. By downloading popular applications, malicious authors disassemble those apps, insert malicious payloads, and then re-assemble them and finally they submit them to either official or unofficial marketplaces. Malware authors are likely to use benign and trusted class-file names, to hide their attached malicious payloads, for example, the first version of **DroidKungFu** uses **com.google.ssearch** to cover up as Google search module, whereas, their later versions use **com.google.update**, to disguise as Google update [37].

As for update attack, which is a technique widely used among malware authors. Instead of attaching the whole malicious payloads into legitimate applications, this technique attaches an update component into such applications. The update component will download the malicious payloads at runtime, which decreases the possibility of detection. Hence, a static scan on such

applications, can possibly fail to catch the malicious payloads [37]. **DroidKungFuUpdate** malware family, is one of the malware families that adopts that technique. Moreover, when running **DroidKungFuUpdate malware,** an update notification will be displayed, it indicates that a recent version is available to download, and the user is given the chance to accept or refuse updating the application. This notification is provided by a third-party library, that matches the official notification functionality. When the user accepts, the new version will be remotely downloaded over network. Another example for update attack, is carried by **AnserverBot** malware, which does not require the user's approval to download a new version. **AnserverBot** updates a certain component within the legitimate application, and since it is not the whole application, the user's approval is not required [37].

Drive-by download is the third technique used to install malware, it performs drive-by download attacks on mobile platforms, but rather than using the traditional method of exploiting mobile browsers, it tricks users into downloading attractive applications that claims to perform certain tasks [37]. For example, **GGTracker** malware functions by providing a few advertisement links, which when clicked on, gets directed to a malicious website. That website deludes the user by claiming that it examines the usage of the mobile battery, it then redirects the user to illegitimate marketplace, to download an application that pretends to improve the battery performance. Instead, the user ends up with a malware that subscribes to a premium-rate service stealthily [37].

Finally, as we mentioned earlier that, the previous techniques are the most popular among malware authors, however, several standalone applications perform malicious behaviours, such as, spying on users' locations, stealing users' credentials, subscribing to premium-rate services and leveraging root exploits [37].

## 2.3 CopperDroid

CopperDroid is VMI-based System, where a user can upload an APK file to get analysed, and will produce a JavaScript object notation file (.json extension), and .syscalls file. These generated files, include highly detailed behavioural profile on the submitted application. The analysis that CopperDroid outputs, consist of operating system specific (low-level), and Android specific (high-level) behaviours. An example on low-level behaviours, is writing to a file or executing a program, and high-level behaviours, making a phone call or sending an SMS. Furthermore, considering the fact that CopperDroid, depends on tracking system calls made, when analysing programs, allows it to be resistant, when there're changes made in the Android environment [26]. Another advantage of using CopperDroid, that it can explicitly indicate where the malicious application will initiate its behaviour from, either from java, or native code execution [32].

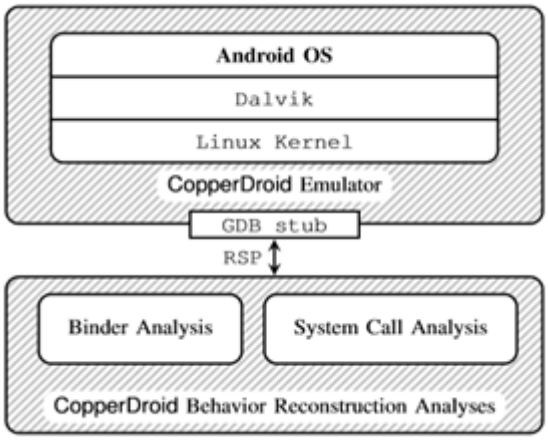


*Figure 3. CopperDroid's Architecture*

The structure of CopperDroid (see Figurer 3), is as follows, the Android system is built on top of CopperDroid's emulator, which is built on top of QEMU. All The analysis takes place on the other side of CopperDroid's emulator, and it distinguishes Binder analysis from system call tracking. Additionally, to obtain a subtle communication between host and emulator, remote serial protocol (RSP) of the GNU debugger is used along with GDB stub provided by Android's emulator [26].

Moreover, CopperDroid uses two mechanisms that helps in analysing applications, one way is by tracking system calls invocations. Unlike other systems, it focuses on intercepting transitions logged in **cpsr** register, between supervisor and user modes. Allowing it to recover all system calls made whether it returned values or not (e.g., **exit**, **execve**) [26].

Dissecting Inter-process communication (IPC) and remote procedure calls (RPC), is the second mechanism CopperDroid adopts. The Binder kernel driver, handles **ioctl** system calls, which allows two components of an application to communicate [32]. By examining communication that occurs over these channels, CopperDroid is able to distinguish the characteristics of a malicious application [26].

Additionally, several dynamic analysis approaches follow a simple install-then-execute method, which discards many important behaviours. CopperDroid, however, uses a technique to stimulate a running application. By examining the application Manifest, and injecting a number of artificial events, such as receiving a phone call, SMS, which is helpful to the analyses [32]. Considering the Android's permission system and its underlaying security, which doesn't provide CopperDroid of much freedom. Alternatively, CopperDroid extracts the events and permissions an application requires, in order to stimulate its behaviour [32].

CopperDroid's behavioural profiles are enhanced by the capability of converting a sequence of related low-level events, into a high-level behaviours. Also, by being able of recreating resources, that are used by a certain application, such as files and network communications [32].

Overall, by reconstructing Android complex objects, tracking system call invocations, and Binder analysis, CopperDroid can distinguish 6 classes of behaviours. Each class consists of one or more behavioural models, which is attained by a number of actions (see Figure 4). The complexity highly varies among different elements, some elements can be characterized by a single system call, such as **execve**. Whereas, other elements, for example, **'SMS Send'**, are characterized by a number of transactions of the Binder protocol [32].


*Figure 4. Hierarchal map of observed behaviours by CopperDroid*

### 2.3.1 Accuracy of CopperDroid

CopperDroid has evaluated over 2900 samples, which are taken from public datasets, such as, Android Malware Genome Project, Contagio and McAfee. It indicates that CopperDroid is capable of behavioural reconstruction at a significant rate. However, more than 60% of the analysed samples had approximately 25% additional behaviours, which prove the system's capabilities by using application stimulation technique [32].

### 2.3.2 Behavioural Profiles

After the Dynamic analysis process is completed, transforming the captured execution trace and reconstructing it as a behavioural profile is what comes next [2]. Considering the high accuracy of CopperDroid's Analysis, the behavioural profiles it produces, perfectly reflects the malicious behaviour performed by a certain malware. Thus, those profiles were used in assembling the dataset. Figure 5 below, is a sample of a behavioural profile produced by CopperDroid, which belongs to the **BASE_BRIDGE** malware family. This Figure contains **two** distinct behaviours (class), **FS ACCSESS** and **ACCSESS PERSONAL INFO**, and **one** sub-behaviour (subclass), **PHONE**. There also are **3** system call invocations (Binder transactions), **open()**, and **write()** to the underlined file names. Along with, a Binder transaction, **GET_CONENT_PROVIDER_TRANSACTION().**


*Figure 5. Sample behavioural profile from CopperDroid.*

## 2.4 Feature Set

Features resemble behavioural characteristics of a given sample. So, by examining each sample and extracting these characteristics that forms the entries in our feature set. The features extracted however, represents our input to the clustering algorithm [2]. When selecting features, it is crucial that unique characteristics are discarded, since it does not appear in all profiles. Therefore, when clustering, a sample that has some unique characteristics, using these features will not ease the process of finding other samples that behaves similarly [2].

### 2.4.1 Parsing. json files

Files produced by CopperDroid, .json files contain a detailed profile on each application malicious behaviour. Starting with extracting features from **.json** files, in a suitable format, such as vectors, that can easily be fed to our algorithm, and considering that our dataset contains multiple Android malware families; each family contains several samples. Thus, the values of each family should be separated from others, by using the family name as the key.

Feature vectors are implemented in several forms, such as Bit-vectors, frequency-vectors, n-grams, …etc. When implementing frequency vectors, the number of times each feature exists in a sample, will increment its corresponding count value. Whereas, in Bit-Vectors, a vector of all methods in

dataset is created, and after iterating through the contents of a sample profile, a vector of 0s and 1s, illustrating whether a certain feature is present (1) or not (0). For example, when having the following constructed behaviours (X),

```
X = { open(), read(), write(), write(), close(), open(), close() }
```

```
Set of distinct methods = { open(), read(), write(), close() }
```

we include each distinct entry (set(x)), and produce the following:

```
Frequency-Vector = {2, 1, 2, 2}
Bit-Vector = {1, 1, 1, 1}
```

Also, using Bi-grams to form our feature set, allowed us to significantly increase the number of dimensions we have from **43** up to **180** dimensions. The way Bi-grams work, for example, if were provided of the set of

```
X = { open(), read(), write(), close() }
```

Our algorithm will loop through the set of system calls, and add each one of them to the list, however, it will consider the series of calls that were made after one another.

```
Bi-grams = { open(), read(), write(), close(), open()read(), read()write,
write()close() }
```

And after we created the sequence of methods, we can either apply frequency or Bit vectors to represent Bi-grams. Thus, it allowed us to cover the basic behaviours and enriched our feature set of the sequence of method calls made.

Overall, the parsing programs included high-level behaviours, which is the class and subclass information, also the binder transactions made, such as, Binder, Intent, and Syscall and used them as entries in our feature vectors.

# Chapter 3: **Machine Learning**

Statistical learning, is the whole set of tools, that helps in understanding data. These tools can be categorized to supervised, and unsupervised.

In supervised learning, by acquiring data inputs, we can estimate (predict) their possible outputs (label). This method is broadly used; however, in this section, our primarily focus will be on unsupervised statistical learning, which is recognised as having variety of inputs, without having their corresponding supervised outputs (no labels provided) [8].

By means of unsupervised learning, inspecting each element characteristics, can guide us to discover a way, which eases the process of grouping data. Using a similarity measure, applying it to a certain dataset, can result in gathering data into distinct groups. Its fundamental goal, is to discover the underlaying structure of the observations provided. Thus, Clustering, is a technique, that simplifies the process of determining subgroups in dataset [8].

When working with large datasets, that consist of a large number of dimensions, and several samples, Principle Components Analysis, which is a technique of unsupervised learning, is considered to be a useful tool that aids in visualising, and pre-processing data, as its reduces the number of dimensions [8].

In the next section, we introduce principal Components Analysis, and the way it works, we also elaborate on the effects of normalizing data, and how to properly determine the number of components needed.

## 3.1 Unsupervised Learning

### 3.1.1 Principal Components Analysis (PCA)

PCA Is a widely known technique, that is used for reducing dimensionality, and decomposing large datasets that have a vast number of dimensions to a lower dimensional space. The data points will be displayed, as orthogonal components, which describes the maximum amount of the variance.

In general, instead of using all dimensions included in our dataset, performing PCA, will result in using, for example, two dimensions at most that will provide an accurate representation of data [8].

A number of benefits arise when using PCA, which can be summarised in the following:

- It reduces the capacity and memory consumption, when working on large datasets.
- It decreases the complexity of a system, by excluding redundant attributes.
- It is less sensitive to noise than other algorithms.

However, it can be argued that, evaluation of the covariance matrix result in less accuracy. Which can be viewed as a disadvantage of PCA [12].

**Effects of Normalization and Standardization on PCA**


*Figure 6. Effect of standardization on PCA*

Standardization plays a significant role, as it is a requirement in several machine learning algorithms. Scaling variables is of a significant importance since; different variables can be measured using different units, for example, a dataset could include a feature called **Alcohol,** which is measured by volume percent, and another feature called **Malic acid,** which is measured by gram per litre and when combining these two features, it can result in one of the features ruling out the others [8]. Also, considering that in clustering analysis a certain distance measure is used in comparing similarities between features, such as, Euclidian distance, therefore, standardizing values is a key step. Also, since we are curious about the components which maximize the variance [25]. The figures (see Figure 6) above illustrate the effect of standardization on PCA, which clearly imply, that standardizing data, will result in more apparent distribution of data.

**Determining Number of Components**

When working with principal components analysis, choosing the number of components needed is a vital part. Since the number of components is essentially a way of accurately describing data but with less number of dimensions. However, by examining the cumulative *explained variance ratio* as a function, we can roughly determine the number of components we need to efficiently describe our data [34].

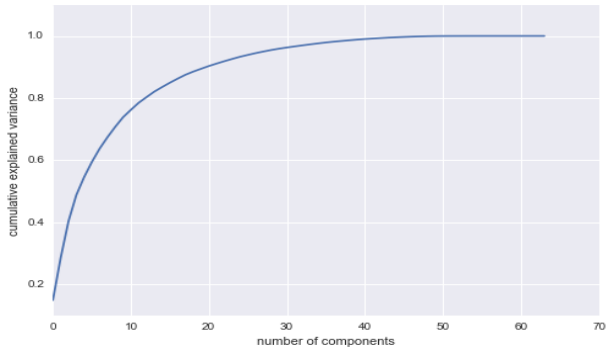


*Figure 7. explained variance ratio*

In the graph above (see Figure 7), the curve demonstrates how much of the variance is genuinely contained within the first Nearest Neighbours (NN) components. Taking 20 components, for

example, yields approximately 90% of the variance, and in order to describe 100% of the variance, the number of components needed should be in the range of **43 to 50** [34].

By using the same approach on our dataset, we managed to determine the number of components required. When working with **43** dimensions, which was produced by feature extraction class (frequency vector), and applying the previous approach, the suggested number of components as seen on Figure 8 below (left), is between **22 and 25**. Likewise, when using another feature extraction class (Bi-grams), that produces up to **180** dimensions, the suggested number of components ranges from **79 to 81**, in order to fully describe the variance. However, using this approach provides a way of discovering the level of redundancy in our dataset.

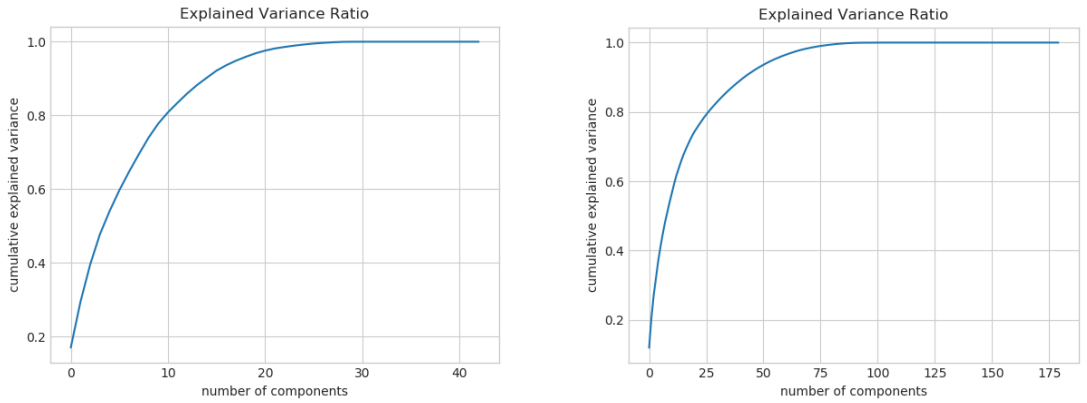


*Figure 8. Explained Variance Ratio for dataset (Frequency Vector- **left**, Bi-grams- **right**)*

# Chapter 4: **Clustering**

Clustering can be recognized as the set of algorithms, which are used in order to find similar subgroups within datasets. Each cluster, contains number of observations in which they're found similar to each other, whereas observations located in other groups are found dissimilar. A partitioning rule, is used to determine what observations can be included at each exact cluster, in other words, the characteristics that makes two items similar or different [8]. Nonetheless, Clustering is unsupervised problem, because it's applied to determine different groups, can be thought of as assigning labels to a set of unlabelled data.

There are quite a few requirements that each clustering algorithm, should satisfy, which includes, ability to deal with noise and outliers, high dimensionality, insensitivity to order of data provided and scalability [35].

Issues may arise when clustering, such as, the difficulty of specifying a proper distance measurement, that is used to indicate where the partition should be between clusters. Time complexity is also considered a problem, since datasets may be of high dimensionality, and the results of clustering algorithms can be viewed in different ways [8].

To better understand clustering, vectors, distance measures concepts should be addressed appropriately. A dataset consists of a series of points, where each point represents an object. Each object can be thought of as a vector of numbers, where the length of that vector resembles the number of dimensions in our space [24]. Moreover, the components of a vector(object) are called coordinates, which is used by our distance measures to determine the similarity or difference between two objects [24].

In general, there are two forms of clustering, which are soft and hard clustering. In hard clustering, each element is assigned to only one cluster. Whereas, in soft clustering, an element is decided to belong to a cluster based on the probability of belonging to either one [35].

However, since every clustering algorithm follows a different set of rules for determining the similarity measure between observations. The following sections will discuss, 3 of the most popular algorithms used nowadays, which are, K-means, Hierarchal and Density-based Algorithms. In each section, the way each algorithm functions will be explained along with advantages and disadvantages of using it.

## 4.1 K-means Clustering Algorithm

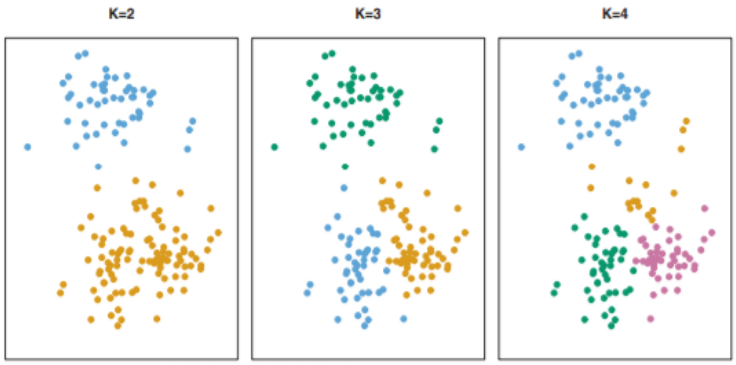


*Figure 9. applying K-means clustering with different values of K*

Partitional algorithms depends on specifying the number of clusters, the data should be divided to, beforehand. K-means belongs to partitional algorithms category, since K-means algorithm uses a pre-defined value of K that specifies the number of distinct clusters the dataset should be divided into (see Figure 9) [17]. The algorithm assigns points to clusters, by checking which cluster centroid is found nearest. The centroid value is computed as the average of all the points in that cluster (see Figure 10) [17]. However, choosing the correct value of K is a crucial part, since the algorithm depends on that value when assigning observations [8].

Moreover, K-means successfully satisfies two main conditions, that are, each data point should belong to one cluster, and there is no overlapping found in clusters.

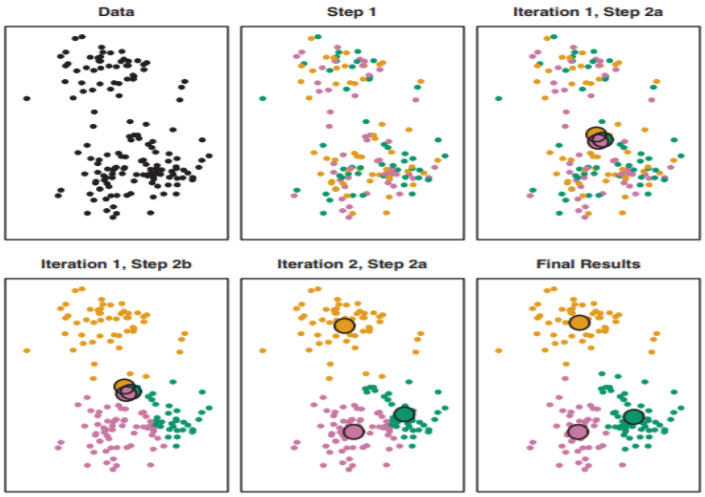


*Figure 10. The progress of K-means Algorithm, K=3*

An important point that should be as well considered is, minimizing the within-cluster variation. The within-cluster variation, measures how tightly grouped the clusters are and can in fact be determined using several options, such as, Euclidian's or Manhattan's distance [8].

An abstract way to describe K-means clustering algorithms, used *from Introduction to statistical learning* book [8], is as follows (see Figure 10):

1. For each observation in dataset, assign a random number from 1 to K.
2. Iterate until clusters, stop changing.
    a) For each cluster K, compute cluster centroid.

b) Allocate each observation to the cluster, that has the nearest centroid.

**Estimate Number of Clusters**

Since the process of estimating the number of clusters a dataset should be divided into, is a considerably complicated process. Using Silhouette coefficient plot, can noticeably help in computing the number of clusters visually. We can use silhouette analysis to understand the separation distance between resulting clusters [10]. The silhouette plot computes how close, data points in a cluster, are to other data points in neighbouring clusters. The values Silhouette coefficients range between are **[-1, 1],** as **0** indicating that a data point is on or close to the decision margin, and **1** is the exact opposite, hence, the data point is considered to be far from neighbouring clusters. Whereas, any negative value will indicate that samples are assigned to wrong clusters [10].

For example, the following Silhouette plots (see Figure 11) shows that, choosing **k** equal to **3** is a bad choice, particularly for this dataset, whereas, choosing the value of **4** would yield better results.

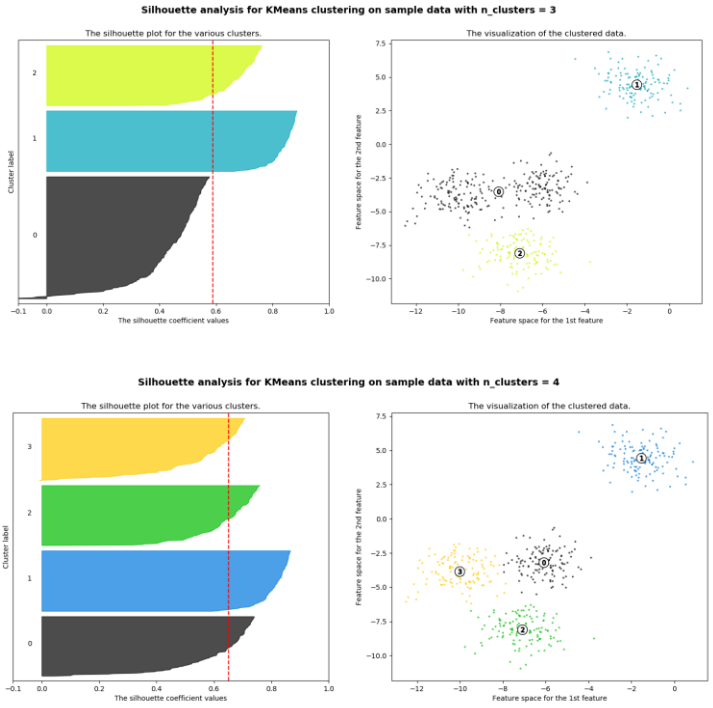


*Figure 11. Silhouette plots, with k= 3(top), and k=4(bottom)*

Therefore, by using the value **4**, all the clusters are found to be of equivalent size [10].

### 4.1.1 Advantages

Using K-means is found to be efficient with datasets of high dimensionality, it is also considered to have an easy implementation, and has a high performance(fast), when the k value is small [11]. Also, when compared with hierarchal algorithms, K-means is able to produce tighter clusters [11].

### 4.1.2 Disadvantages

There are several difficulties encountered when using K-means, which include, choosing the value of K (number of clusters) beforehand, portioning data into K clusters while keeping the within-cluster variation minimized, since there are $K^n$ different ways to portion data [4, 19].

Also, if the dataset contains two highly overlapping classes, K-means algorithms will not be able to cluster them accurately, as it will probably assume they belong the same cluster [11].

## 4.2 Hierarchal Clustering Algorithm

In Hierarchal algorithms, the clusters are found based on previously discovered clusters [17]. And it is an algorithm that doesn't require the user's prior knowledge in specifying the number of clusters beforehand. Instead, it creates a tree-based representation of data observations (builds hierarchy of clusters), which is known as a dendrogram. The algorithm produces dendrograms of two types, which are Agglomerative, and Divisive [8].

The Agglomerative type, which is also known as "bottom-up", functions in a way that each observation initiates its own cluster, and as we move upwards (towards the top of hierarchy), pairs of clusters gets merged together. Whereas, in the Divisive "top-down" approach, all the observations start at the same cluster, and the algorithm will start splitting them up to clusters, while moving down the hierarchy. However, using Agglomerative approach is found to be less problematic, since the Divisive approach needs to consider all possible divisions the data can be placed into [35]. However, using the Divisive approach, in some cases, is more efficient, especially when working on a certain number of top levels, which means we don't want to generate a complete hierarchy [17].

The height in the dendrogram at which two clusters are merged represents the distance between two clusters in the data space. In addition, observations that mainly fuse near the bottom in a dendrogram, tend to be similar, while, observations that fuse near the top of tree are quite different [8].

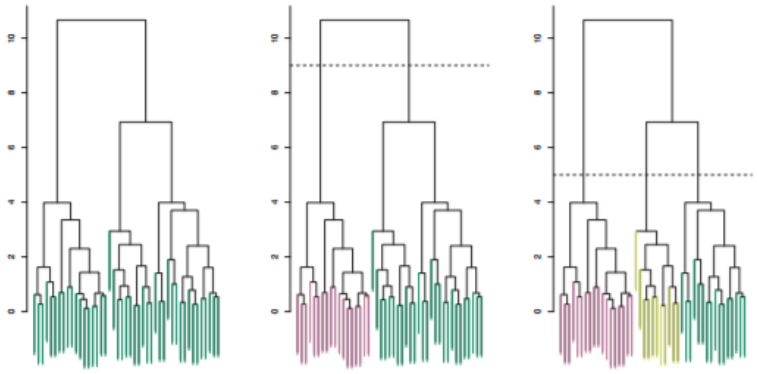

*Figure 12. dendrograms obtained from hierarchical clustering*

To discover number of clusters a dendrogram contains, a horizontal cut at a relevant depth can determine that. And depending on where the horizontal line is placed, the major branches found below that line, represent the different clusters (see Figure 12) [4,17]. In general, the horizontal line that cuts through a dendrogram in hierarchal clustering, is equivalent to the K value in K-means, which both specify the number of clusters that the dataset can be divided into [17].

Different values of where the horizontal line cuts the dendrogram, would indicate a different number of clusters, i.e. in Figure 12, Centre- the horizontal line is at height **= 9**, and divides dataset to **2** distinct clusters, while on the left, the line is at height**=5**, thus, number of clusters is set to **3** in that case.

### 4.2.1 Distance measures

Hierarchal algorithm works with several distance measures, that is used to compute the distance between elements. For example, we can use the Euclidean distance, that considers the distance between two data points and compute it as the square root of the sum of the squared differences of distance between two variables. Another example, is correlation-based distance, which computes the correlation between features of two observations to be high, if they're found similar [17]. Also, the Manhattan distance, is considered one of the simple measures to be integrated with the algorithm, which is equal to *"the sum of absolute distances for each variable"* [17].

### 4.2.2 Linkage

Describing the dissimilarity between two sets of data points (two groups of observations), by using one of the following four different linkage types to achieve (see Figure 13), the required output:

- Single linkage: measures the closest pair of points.
- Complete linkage: measures the farthest pair of points
- Average linkage: measures the average dissimilarity over all pairs
- Centroid linkage, measures the distance between the group centroids (i.e., group averages) [8].

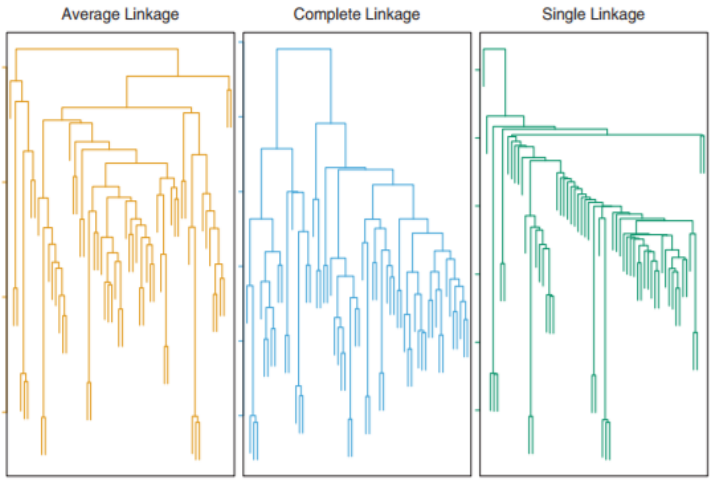


*Figure 13. Shows different Linkage methods used*

According to James et al., a way to describe Agglomerative hierarchal clustering algorithms, is as follows:

1. Starting with **n** observations, and a dissimilarity measure, such as Euclidean distance, from **n** to **2 = n (n − 1)/2** pairwise dissimilarities, where each data point is a cluster.
2. Iterate from **i = n** to **2**:
    a) Examine all pairwise inter-cluster dissimilarities among the **i** clusters and identify the pair of clusters that are least dissimilar (that is, most similar). Fuse these two clusters. The dissimilarity between these two clusters indicates the height in the dendrogram at which the fusion should be placed.
    b) Compute the new pairwise inter-cluster dissimilarities among the **i − 1** remaining clusters

However, in order to improve the quality of hierarchical clustering, there are two approaches that can guarantee that,

- By performing a thorough analysis of objects linkage, emphasising on each hierarchical partitioning, such in CURE and Chameleon [17].

- By integrating hierarchical agglomeration and reformulating results, with iterative relocation, such in BIRCH [17].

### 4.2.3 Advantages

It does not require the user to specify the value of K before starting the algorithm. When using Agglomerative approach, there is some flexibility given to the user, to choose the number of clusters. And by using Divisive clustering, the result is considered, the best possible solution, since the user can access all the data [19].

### 4.2.4 Disadvantages

Specifying the terminating condition, which determines when the algorithm (division or merge process) needs to stop, in either one of the types, can be found challenging, as it needs to be small enough to separate the clusters and large enough to prevent splitting the same cluster into two or more clusters [9]. Moreover, determining where the horizontal line should be placed, counts as a weakness, as it's in many cases, may seem ambiguous.

One more disadvantage arise when using Divisive approach, since several computational difficulties emerge particularly when splitting the clusters [19].

Also using Centroid linkage, can cause an inversion, which will result in more difficulties, especially in visualising and analysing data [8].

However, hierarchal algorithms sometimes can lead to less accuracy when compared with K-means [8]. Depending on the distance metric used, the results would vary, and if the data grouping/splitting is incorrectly made in earlier stages, the results cannot be altered after [19].

## 4.3 Density-Based Spatial Clustering of Applications with Noise (DBSCAN)

According to Kreigel et el. explanation of Density-based clustering*," in density-based clustering, a cluster is a set of data objects spread in the dataspace over a contiguous region of high density of objects. Density-based clusters are separated from each other by contiguous regions of low density of objects.".* The resulting clusters, unlike previously mentioned algorithms, would have an arbitrary-shape, that's determined by high density areas, where number of data points exceeds threshold value (see Figure 14). Moreover, data points that exist in low-density areas, are considered either noise, or outliers [9].

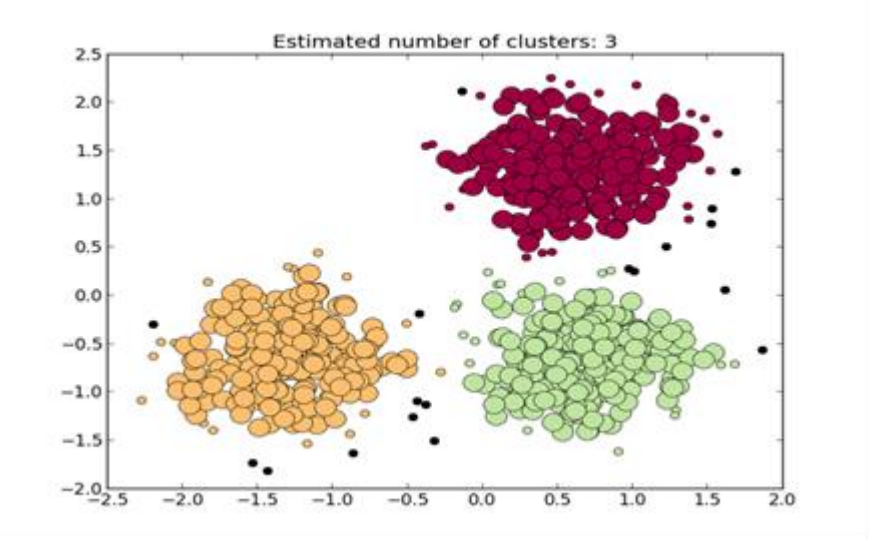


*Figure 14. Plot of DBSCAN performed on dataset*

The DBSCAN Algorithm needs two parameters to preform clustering, which are:

1. **Eps (ε)**, that describes the radius of neighbourhood for a given point; serves as distance threshold.

2. **MinPts**, which defines, the minimum number of points that must exist in eps radius to form a cluster, serves as density threshold.

Nevertheless, each observation could be categorized, as a core point, border point or an outlier (noise point), using value of Eps threshold parameter [14]. Considering Figure 14 as an example, points that are located on the coordinated **(7.5, -1),** resembles a core point, whereas the point at **(0, 0),** represents a border point. And all the black points on the graph, are considered noise.

There are some significant definitions made by Shah, et al. [9], by the definition of density-reachability, which states that in order for two data points, a core and border point, p and q, to be density-reachable, the following criteria should be met:

1. $\boldsymbol{p \in Eps(q)}$ , point p is within Eps threshold of q

2. $\boldsymbol{|Eps(q)| \geq MinPts}$, the number of points in q's neighbourhood is greater than or equal to MinPts (Core point condition).

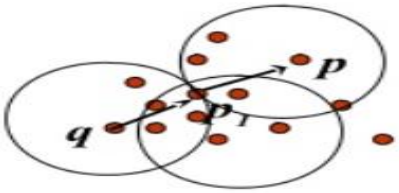


*Figure 15. Density-reachability*

This relation, is guaranteed to be transitive (see Figure 15) [9].

Density-Connectivity, is another definition, that implies:

For points p, q and o, to determine if p is density-connected to q, the following condition must apply:

1. There exists a point o, where p and q are both density-reachable from o, with respect to Eps and MinPts. (if they have a distance of less than r).

Density-Connectivity is a symmetric relation, and for density-reachable point its considered reflexive as well (see Figure 16) [9].

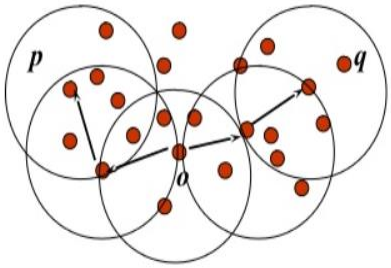


*Figure 16. Density-Connectivity*

Yet, there are several distance measures that can be associated with DBSCAN, such as, Euclidian, Manhattan measures, however, Euclidean distance is more commonly used. [17].

In order to compute the distance between two n-dimensional vectors, the Euclidean distance between $\mathbf{x_p}$, and $\mathbf{x_q}$ is computed by (see Figure 17):

$$d^2(x_p, x_q) = \left\| \sum_{k=1}^{n} x_k^p - x_k^q \right\|^2$$

*Figure 17. Euclidean distance formula.*

Where **n** is the number of dimensions for each data point, $\mathbf{x_K^p}$, is the $k^{th}$ dimension of $\mathbf{x_p}$.

and $\mathbf{x_K^q}$, is the $k^{th}$ dimension of $\mathbf{x_q}$. [1].

We also used the Manhattan distance with our clustering algorithm, which can be defined as *"distance between two points in Euclidean space with fixed Cartesian coordinate system"* [30].

$$d(i,j) = \sum_{k=1}^{n} |x_k(i) - x_k(j)|$$

*Figure 18. Manhattan distance formula.*

By following the formula above (see Figure 18), we can compute the distance between **i$^{th}$** and **j$^{th}$** object. [30].

Moreover, when working with a dataset, that doesn't contain any sort of correlation between its attributes, using Euclidean distance would produce sufficient results for the analysis. However, when working with different datasets, which may contain different normalization conditions, using Euclidean distance might not provide satisfactory results. So, experimenting with different measures, would lead to finding the best measures [30].

**Algorithm:** DBSCAN

**Input:** i) Set of $m$ data points $X=\{x_1, \ldots, x_m\}$,
ii) ε (epsilon), the neighborhood distance and
iii) $\eta$, the minimum number of data points required to form a cluster.

**Output:** Set of clusters $C = \{c_1, \ldots, c_k\}$,

**Steps:**

1) $C = \emptyset.; i = 0;$
2) **for each** $x_p \in X$ *and* $x_p.visited = false$
3) **begin**
4) $x_p.visited = true$
5) $N_p = N_\varepsilon(x_p)$ using (13)
6) **if** $|N_\varepsilon(x_p)| < \eta$ **then**
7) $x_p.noice = true$
8) **else**
9) $i = i + 1$
10) $C = C \cup c_i$
11) $c_i = c_i \cup x_p$
12) **for each** $x_q \in N$
13) **begin**
14) **if** $x_q.visited = false$ **then**
15) $x_q.visited = true$
16) $N_q = N_\varepsilon(x_q)$
17) **if** $|N_\varepsilon(x_q)| < \eta$ **then**
18) $N_p = N_p \cup N_q$
19) **if** $x_q \notin c_j \; \forall j = 1 \le j \le i$ **then**
20) $c_i = c_i \cup x_q$
21) **endif**
22) **endif**
23) **endif**
24) **end**
25) **endif**
26) **end**

*Figure 19. DBSCAN Algorithm.*

The following is an illustration of DBSCAN Algorithm (see Figure 19), in a more simplified manner [28]:

1. Compute neighbours of each point and identify core points
2. Join neighbouring core points into clusters
3. For each non-core point do:
    a) Add to a neighbouring core point if possible // Assign border points
    b) Otherwise, add to noise

Using the definitions of density-reachability and density-connectivity, if two clusters A and B, of different densities, are found close to each other, a distance method is used to compute the distance between all data points in cluster A, and B. Based on distance computed, two clusters remain separated if the distance between their points is larger than Eps [9].

Generally, two properties must be satisfied by every cluster. One, all the data points inside a cluster, are density-connected. Two, if any data point is density connected to another data point of a cluster, this point should be a part of the cluster too [1].

### 4.3.1 Determining values of MinPts and Eps

Determining the values of **MinPts** and **Eps** parameters are essential for DBSCAN. Starting with **MinPts** parameter, depending on whether dataset contains a lot of noise or not. Sander et al. [9] suggest setting it to twice the dataset dimensionality, i.e., **MinPts = 2 × dim(dataset).** However, when dataset is large or consists of a large number of dimensions, or contains noise, it is preferred to increase **MinPts** value [28]. Another herustic, suggested by Birant et al. [3], indicates that using the natural logarthim (**ln**), can of a great help in determining the value of **MinPts**. By computing the value of **ln(n)**, where **n** represnets the size of our dataset (number of samples), it can provide us with a good estimation of minumn number of samples. In general, domain knowledge can ease the process of estimating both parameters.

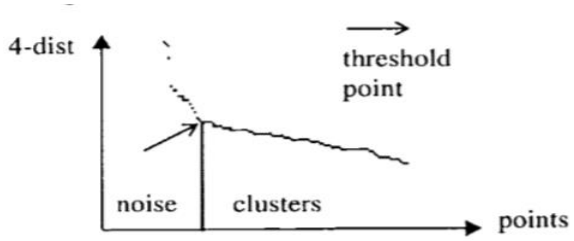


*Figure 20. An example of k-dist. graph*

Moreover, setting value of **Eps** is considered harder than **MinPts**, an appropriate way to set **Eps** value, is using k-distance graph. Ester et al. [9] provide a heuristic to compute its value, that is, *"When sorting the points of the database in descending order of their k-dist values, the graph of this function gives some hints concerning the density distribution in the database. We call this graph the sorted k-dist graph. If we choose an arbitrary point p, set the parameter Eps to k-dist(p) and set the parameter MinPts to k, all points with an equal or smaller k-dist value will be core points. If we could find a threshold point with the maximal k-dist value in the "thinnest" cluster of D we would have the desired parameter values. The threshold point is the first point in the first "valley" of the sorted k-dist graph. All points with a higher k-dist value (left of the threshold) are*

*considered to be noise, all other points (right of the threshold) are assigned to some cluster."* (see Figure 20) [9].

Moreover, when dealing with high dimensional data, the process of estimating **epsilon** value, becomes even harder, since distances lose their contrast with higher dimensionality. Such difficulties can be avoided, by using OPTICS or HDBSCAN, which don't require **epsilon** value [28].

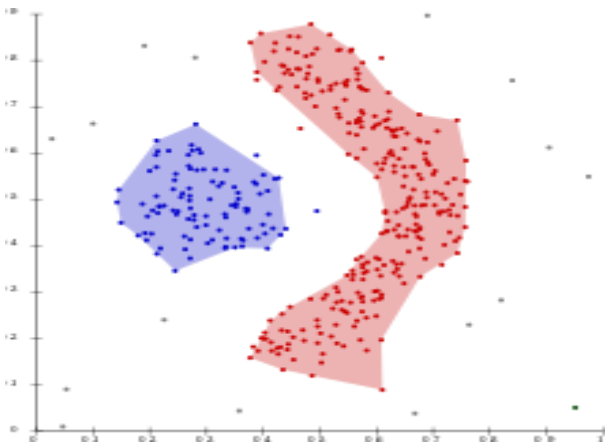

*Figure 21. DBSCAN result, that has one cluster surrounded by another*

### 4.3.2 Advantages

In this approach, a lot of advantages arise, such as, efficiency when used on high dimensional datasets, user is not required to determine number of clusters in advance, Moreover, the algorithm is more effective in defining clusters of arbitrary-shape, and the algorithm is considered to be resistant to noise (has the ability to distinguish noise points) [28].

The algorithm uses two parameters, Eps and MinPts, which are not affected of the order in which data is provided and if computed appropriately, will result in high performance [9]. It can even find a cluster completely surrounded by (but not connected to) a different cluster (see Figure 21).

### 4.3.3 Disadvantages

On the contrary, the algorithm has some drawbacks when clustering datasets, that has a great difference in densities, since MinPts and Eps parameters cannot possibly be computed correctly for all clusters [28]. Another point to consider as well, is the order in which data is processed, sometimes may result in different changes in the outcome of clustering algorithm [28]. In the case of ambiguous datasets (that aren't understood in terms of data and scale used.), the process of computing Eps parameter can be found difficult [29].

# Chapter 5: **Evaluation and Results**

The intention of this chapter, is to verify our clustering algorithm effectiveness, and performance in generating clusters. In order to produce a reliable output, we assessed our approach on real malware samples that formed our dataset.

In the first section, we introduce different measures, both internal and external, we also explain some terms, such as reference cluster, precision and recall, Silhouette coefficient. Followed by a section that contains the different experiments performed on our dataset, using different feature sets and the technique we used in setting our parameters, we also measure the quality of our approach in clustering each of the feature sets provided, as well as, comparing the results obtained from our experiments.

## 5.1 Algorithm Evaluation

For the success of any clustering algorithm, evaluating its correctness and the quality of its produced clusters, is an essential factor that should be considered [16]. There are three main clustering validation categories, which are Internal, external and relative measures. Mainly the external measures are used on supervised data, where it precisely compares a clustering against prior or expert-specified knowledge, such as, class labels (ground truth), hence, obtaining its value from external information [6]. In external measures, there are four crucial standards that should be met, those include homogeneity, completeness, distinguishing noise and preserving small clusters. On the other hand, Internal measures are fundamentally applied on unsupervised data, thus, the results achieved are inherited from the data itself [6]. It evaluates the quality of a clustering by determining how well, clusters are separated and how compact clusters are. As for relative measures, it is accomplished by comparing different clustering results, which are the result of using different parameter settings, to each other.

However, there are several cluster validation measures, in our case, we evaluated our clustering using external measures, such as, precision, recall, F-score, homogeneity, completeness, V-measure, and Rand Index. As for internal measures, we used Silhouette Coefficient [16].

### 5.1.1 External Validity

We previously mentioned four standards that external measures should meet. Firstly, we have homogeneity, which examines that a cluster is pure, so it only contains data of the same category, class. Secondly, cluster completeness, which checks that all the data points of a certain label in our ground truth, are assigned to the same cluster.

However, both these standards are in opposition that means, increasing homogeneity of a clustering, usually yields a decrease in its completeness, and vice versa [27]. And they can be both used as measures to precisely determine the performance of the clustering offered [27].

Thirdly, handling varied objects, indicates that noise points should not be assigned into pure clusters, as it should be allocated separately, hence, outside any cluster. And finally, small cluster preservation, the effect of splitting up small clusters will result in more noise; however, it would be preferred if larger clusters are split up instead [16].

Since the process of discovering the flaws in the clusters our algorithm produce, is a fairly complicated process, we propose to measure that difference between the original clustering and our results, by using V-measure [27]. The V-measure, is a representation of the harmonic mean of both homogeneity and completeness scores. It uses a similar approach, such as F-measure, which combines both precision and recall, and favour both their contributions. The V-measure, thus,

combines the values of homogeneity and completeness. Additionally, computing these three values, is independent of the size of our dataset, the number of classes or clusters it contains, and the algorithm it is combined with [27].

In the following subheadings, we explain the usage and computation of ground truth, precision and recall, F-score and Rand index.

**Ground Truth**

The process of estimating the accuracy of a clustering algorithm, is not particularly an easy task to accomplish. Clearly, picking several randomly chosen clusters and checking if the samples each cluster contains are similar to each other, by using some similarity measure. The ideal way is fulfilled by comparing results with an existing Reference Cluster [2]. In our case, we were provided with some sort of a ground truth, since the dataset provided consisted of several subdirectories, each representing a malware family.

Withal, our dataset contained 49 malware families, each with different number of samples. When reading each subdirectory, a label (ground truth) was assigned to all samples it includes (numbers from 0 to 48). Thus, when determining the correctness of our clustering algorithm, we compared the resulting labels, produced by our algorithm, with our ground truth, and computed the overall accuracy as follows (see Figure 22).

$$\boldsymbol{Acc} = \frac{correct\ result}{number\ of\ samples} \times 100$$

*Figure 22. Accuracy equation used.*

Morover, If a certain class (label) is split up into different clusters, or if different calss labels are found in one cluster, these findings will particluary decrease the accuracy of an algorithm's output [16].

**Precision and Recall**

In the process of assessing the quality of our approach, two metrics are additionally used to assure that we are obtaining the desired output. We will start by defining each term, and specifying the method (rule) that helps in computing it. Firstly, Precision, aims to measure how adequate our algorithm is, in differentiating samples, by checking whether different samples are clustered into different groups [2]. Whereas, Recall measures how adequate our clustering algorithm in recognizing similar samples, by checking that all the samples in a cluster are of the same type [2]. A cluster precision can be computed by (see Figure 23):

$$P_j = max(|C_j \cap T_1|, |C_j \cap T_2|, .., |C_j \cap T_t|)$$

*Figure 23. Cluster percision rule*

Where $\mathbf{C_j}$ represents clusters produced by our algorithm. And $\mathbf{T_j}$ represents a cluster from the reference cluster. Moreover, to compute the overall precision (see Figure 24):

$$P = \frac{(P_1 + P_2 + .. + P_c)}{n}$$

*Figure 24. Overall precision cluster formula*

On the other hand, recall can be computed by using the method below (see Figure 25):

$$R_j = max(|C_1 \cap T_j|, |C_2 \cap T_j|, .., |C_c \cap T_j|)$$

*Figure 25. Cluster recall formula*

Where $\mathbf{C_c}$ represents clusters produced by our algorithm. And $\mathbf{T_j}$ represents a cluster from the reference cluster [2]. Moreover, to compute the overall recall value, the following method is used (see Figure 26):

$$R = \frac{(R_1 + R_2 + .. + R_r)}{n}$$

*Figure 26. Overall cluster recall formula*

An efficient algorithm should maintain both high percision and recall values, such that it can distinguish between differnet samples, and can detect similarity between samples, and assign them accordingly.

**F-score:**

Both percision and recall are used together to determine the value of F-score, which represenets the harmonic mean of both their values [16]. The F-score is computed as following (see Figure 27):

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

*Figure 27. The formula of F-score.*

**Adjusted Rand Index (ARI)**

As we already obtained the ground truth, we can asses the quality of our algorithm. By using Adjusted Rand Index, which is a function that measures the similarity between our ground truth and the predited labels, it will ignore permutations with chance normalization [10].

This method can be performed by using, Sickitlearn's metrics library. The value it produces is between **0.0**, and **1.0**, as zero being very bad labeling, and one otherwise [10]

### 5.1.2 Internal Vlaidity

On the grounds of clustering real-world datasets, the according labels (ground truth) are not included, as it represents an unsupervised data problem, so instead, we need to evaluate the quality of our algorithm assuming that no ground truth was provided.

**Silhouette coefficient**

One approach to evaluate the quality of our clustering algorithm on unsupervised data, is by using Silhouette Coefficient. The way this measure work is by, finding distinction between the average distance of data points in the same cluster and the average distance of data points in other clusters. Mainly it uses the model itself [10].

Silhouette coefficient composes of two scores which are:

- **a**: The average distance between a sample and all other points in the same cluster.
- **b**: The average distance between a sample and all other points in the next nearest cluster.

Moreover, the silhouette coefficient can be considered a relative measure as well as internal. Its value ranges between **0.0**, and **1.0**, where one indicates that the data is well clustered, and zero otherwise [10].

## 5.2 Experiments

In this section, we outline the different experiments that we carried out, indicating the results we obtained. We conducted **three** different experiments to asses DBSCAN's algorithm quality. In the **1st** and **2nd** experiments, we applied our algorithm on different feature sets, with various number of dimensions. Noting that, these two experiments were carried out on approximately **1200** samples, that correspond to **49** malware families.

After carrying out both experiments **1** and **2** on different feature sets, we used the one that provided the best results, as our feature set in the **3rd** experiment, hence, frequency vectors. Nonetheless, in our **3rd** experiment, rather than using the Euclidian distance as our metric function we used the Manhattan distance, to determine whether it will improve the accuracy of our algorithm or won't.

Overall, in all our experiments, it will be clearly indicated whether the feature set used in each experiment had been normalized, and if we performed PCA or not.

### 5.2.1 Filtering Feature Set

After parsing all the behavioural profiles in our dataset, we discovered that some malware families, contain samples that doesn't consist of any constructed behaviours. Using such information clearly will not improve the quality of our clusters, instead it will lead to an increase in the noise quantity our dataset contains. Taking **BeanBot** malware family as an example, some of its samples, does not contain any information on its own behaviour, which led us to remove it from our feature set.

Moreover, to obtain the best possible results, we performed some sort of filtering process on our dataset, starting with **1230** samples, excluding up to **150** samples since their behavioural profiles were basically empty (vectors of zeros). After excluding such vectors, we ended up with a total number of samples, equivalent to **1078**.

### 5.2.2 Setting Eps and MinPts values

We started with setting up MinPts parameter, considering Ester et al. [14] concept, which suggests setting it to **2 × number of dimensions**. However, setting MinPts parameter to double the number

of dimensions is not concrete. Instead, it requires some domain knowledge in order to set it properly. Moreover, another approach, that was suggested in section **4.3.1**, is using natural logarithm function (**ln**) on the number of samples.

By exploring the dataset, and carrying a few experiments using different **MinPts** values, when following the first heuristic, for example, having **180** dimensions, and setting **MinPts** to **360**, which is considered to be an unreasonable value, since by examining our dataset none of the malware families actually contain that number of samples.

Therefore, we concluded that using large values, such as **16**, **20**, **43**, **180**, **86** and **360**, don't deliver viable solutions, which led us to set **MinPts** equivalent to **8, and that value was used in all three expiremnts**.

Similarly, when setting **Eps** value, as stated by Ester et al. [14], in section **4.3.1**, to ease the process of estimating **eps** value, a k-distance graph can provide a rough estimate on the range of suitable **epsilon** value. The heuristic suggests, that in order to get the correct range of values, we should look for **K = (2 × number of dimensions) -1** or **K = MinPts**, neighbour. In this case, we implemented a K-Distance class, to assist us in choosing the appropriate **Eps** value. By using Sicikitlearn implementation of Nearest Neighbour, we applied it on our normalised dataset, we searched for the **$k^{th}$** neighbour, which is equivalent to **8**.

After obtaining the distances, we plotted them, and searched for an elbow in each graph (see Figure 28).

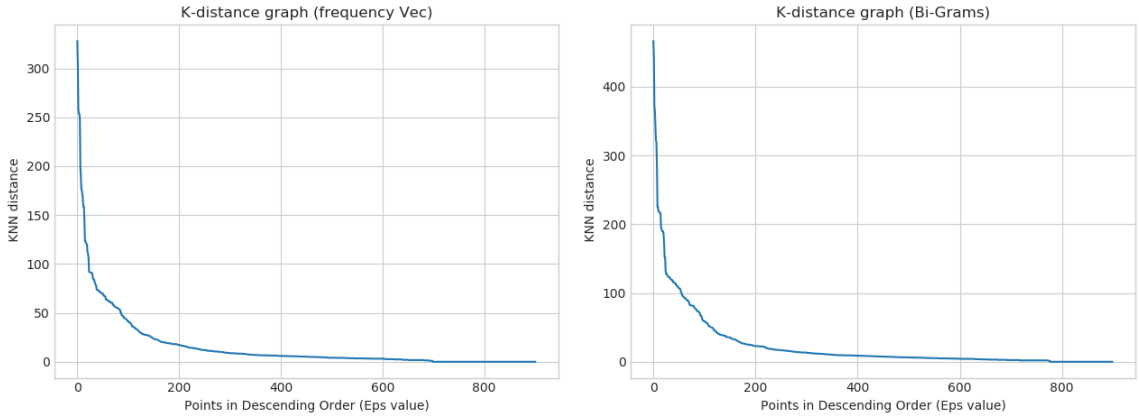


*Figure 28. K-distance graphs used to estimate value of Epsilon (Frequency Vector (Left), Bi-grams (right))*

Closing up on where the elbow is found, will demonstrate the range of appropriate **eps** values. For frequency feature set (see Figure 28 (left)), **epsilon** should be in the range of **[5.5, 8.5].** As for Bi-grams feature set, the best value of **epsilon** is shown (see Figure 28 (right)), between **[10, 21].**

### Experiment 1

As stated above this experiment was conducted on our dataset which consist of **1230** malware samples that represent **49** malware families. As mentioned previously, we performed some sort of a filtering process on our feature set, which indicates that our new feature set only includes **1078** samples, however, in order to reduce the amount of noise our algorithm would predict, we only included behavioural profiles with **6** and more constructed behaviours, hence, reducing the number of samples up to **902** files.

As a starting point, we used frequency vectors to form our feature set, which contains **43** dimensions each representing a certain behaviour. Next, we set the values of **MinPts**, and **Eps**, to **8**, **6.89**, respectively. We did not perform normalization on data, to illustrate its effect on data, and to check whether it will lead us to better results.

After that, we performed DBSCAN on our input, we ended up with **14** clusters. And since we were provided with the ground truth (labels), we managed to compute the following internal and external measures, to assess the validity of our clustering algorithm.

| measure | Accuracy | ARI | Silhouette | Precision | Recall | F-measure | Homogeneity | Completeness | V-measure |
|---|---|---|---|---|---|---|---|---|---|
| **value** | 0.145 | 0.156 | 0.116 | 0.145 | 0.145 | 0.145 | 0.415 | 0.547 | 0.472 |

By looking at the previous table, we can determine that our clusters, by using internal methods, are of low quality, however, by examining the values of homogeneity, completeness and V-measure, they indicate that our algorithm quality, is close to average.

Since the values we obtained, indicate that our clustering lacks quality, Therefore, we normalised our feature set.

The next step was to properly set our parameters, since after normalizing we need to search for the best values that works properly with our feature set. Thus, **Eps** was set to **1.15**, and **MinPts** remained the same. We ran the algorithm again, and obtained **22** clusters, and computed the same validity measures,

| measure | Accuracy | ARI | Silhouette | Precision | Recall | F-measure | Homogeneity | Completeness | V-measure |
|---|---|---|---|---|---|---|---|---|---|
| **value** | 0.134 | 0.081 | 0.200 | 0.134 | 0.134 | 0.134 | 0.474 | 0.536 | 0.503 |

Apparently Normalizing the values in our feature set, led to an improvement in the overall results, especially in, homogeneity, V-measure and silhouette coefficient.

In the next step, we performed Principle Component Analysis on our feature set. However as indicated in section **3.1.1**, we used Explained Variance Ratio, to estimate the number of components that signifies majority of the variance. The suggested number of components is equivalent to **25**.

After performing PCA, we used the matrix it produced as our input. But considering that the number of dimensions significantly decreased from **43 to 25**, the parameters of DBSCAN should be adjusted accordingly. Therefore, by using the same mechanism to estimate **Eps** value, we set it to **1.3**, and **MinPts** to **8**.

Overall, our algorithm manged to produce **17** clusters. And as our final step in this experiment we evaluated the algorithm using the previously mentioned measures.

| measure | Accuracy | ARI | Silhouette | Precision | Recall | F-measure | Homogeneity | Completeness | V-measure |
|---|---|---|---|---|---|---|---|---|---|
| **value** | 0.118 | 0.142 | 0.191 | 0.118 | 0.118 | 0.118 | 0.450 | 0.570 | 0.500 |

Clearly, using PCA assured that our algorithm remains at same quality level, and in this case, decreased a bit.

Throughout this experiment, we evaluated our algorithm on Frequency vectors of **43** dimensions, each step we performed, however, improved or maintained the same quality measures. Even though, the values we obtained don not specify that our algorithm efficiently cluster data.

### 5.2.3 Experiment 2

In the second experiment, we evaluated our algorithm, by following the same procedure. However, we made a few adjustments to ensure that we obtain better results. Thus, instead of using frequency vectors, we used Bi-grams as our feature set, which consists of **180** dimensions. Considering that Bi-grams provide us with an idea on the sequence of methods that occurred in each behavioural profile, it increases the possibility of identifying samples of the same malware family.

We also performed the filtering process, in which we discarded behavioural profiles with less than **12** constructed behaviours, eventually we ended up with **902** samples, which is similar to what we used in our previous experiment.

Next step was to estimate the value of **epsilon**, as suggested in section **5.2.2**, using the K-distance graph, guided us to set it to **19.95** and **MinPts** to **8.**

After appropriately setting our parameters, we ran our algorithm on the feature set, and obtained **9** clusters. In order to evaluate the quality of our clustering, we assessed it by the following measures:

| measure | Accuracy | ARI | Silhouette | Precision | Recall | F-measure | Homogeneity | Completeness | V-measure |
|---|---|---|---|---|---|---|---|---|---|
| **value** | 0.186 | 0.184 | 0.210 | 0.186 | 0.186 | 0.186 | 0.320 | 0.580 | 0.410 |

The values imply, that the overall clusters are of low quality, as all the values are less than half the minimum required scores, which is less than 50%.

Our next step, was to normalize the feature set, since by normalizing the values, we allow our algorithm to handle all the variables, with the same level of importance. After normalizing data, we estimated **Eps** value to be **6.74,** and **MinPts**, remained the same.

After running the algorithm, our feature set was split into **11** clusters. In order to evaluate the quality of these clusters, we computed the same measures.

| measure | Accuracy | ARI | Silhouette | Precision | Recall | F-measure | Homogeneity | Completeness | V-measure |
|---|---|---|---|---|---|---|---|---|---|
| **value** | 0.11 | 0.176 | 0.135 | 0.11 | 0.11 | 0.11 | 0.314 | 0.520 | 0.392 |

It can be seen from the above results that nearly all the measures scores decreased, in this case, the normalised feature set, did not evidently improve the quality of clustering.

Considering the values, we attained previously, we applied PCA on this feature set, to examine whether using it will help us in improving the overall validity results, in particular, precision, recall, and Silhouette coefficient.

By looking at section **3.1.1**, that describes the number of components required when having **180** dimensions, which is equivalent to **81** components. Thus, our input will consist of **902** rows and **81** columns. Next, we repeated the parameter estimation step, as the number of dimensions visibly changed, hence, **epsilon** value was set to **6.63** and **MinPts** remained the same (equal to 8).

Subsequently, we carried out with our clustering algorithm, and we got up to **11** clusters. And to check whether our algorithm clustered data in an adequate way, we computed the following measures:

| measure | Accuracy | ARI | Silhouette | Precision | Recall | F-measure | Homogeneity | Completeness | V-measure |
|---|---|---|---|---|---|---|---|---|---|
| **value** | 0.11 | 0.178 | 0.138 | 0.11 | 0.11 | 0.11 | 0.320 | 0.521 | 0.393 |

From the results, we concluded that performing PCA on data, maintained the same quality.

### 5.2.4 Comparing experiments 1 and 2

After experimenting with distinct feature sets, which included a considerable variation in the number of dimensions. And, examining the effects of reducing the number of dimensions by performing Principal Components Analysis, on our clustering algorithm results.

It can be concluded that from both experiments, normalising values in general improved the measures scores, however, in one of the cases, when we normalised Bi-gram vectors, the quality of our measures did not increase, instead it slightly decreased.

Nevertheless, all the measures used to evaluate the quality of clusters, did not reach the desirable range of values. Instead, the maximum limit reached was approximately **0.50**, which yields **50% of the required outcome**. In the discussion section, we go over all the possible reasons and limitations that might have caused these inadequate results.

In order to understand, the results we obtained, the following plots were produced for each experiment,

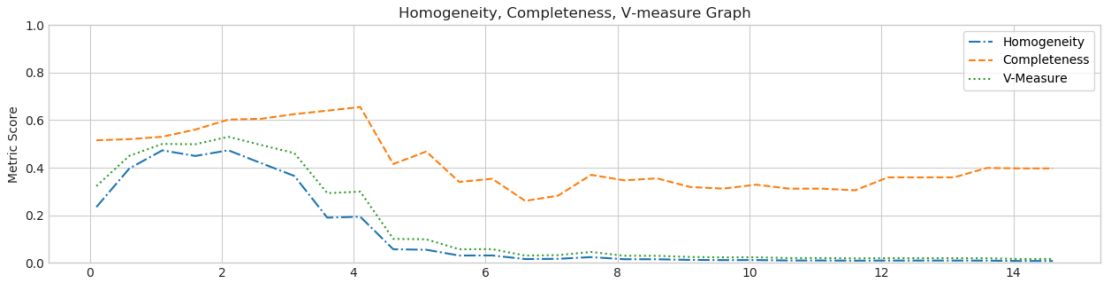


*Figure 29. Experiment 1 Internal validity measures plot*

In the top graph (see Figure 29), we plotted the Homogeneity, Completeness, and V-measure scores, that correspond to each Epsilon value, from that graph we can conclude that the highest value we were able to obtain is approximately when the epsilon value is between **1.10 to 1.15.** The best values are within the range of **[0.48, 0.54].** We can also identify, an indication of the opposite relationship between homogeneity and completeness, considering that when epsilon value is equal to **2** onwards, the completeness score increases to some degree, unlike the homogeneity score, that decreases to **0.06**.

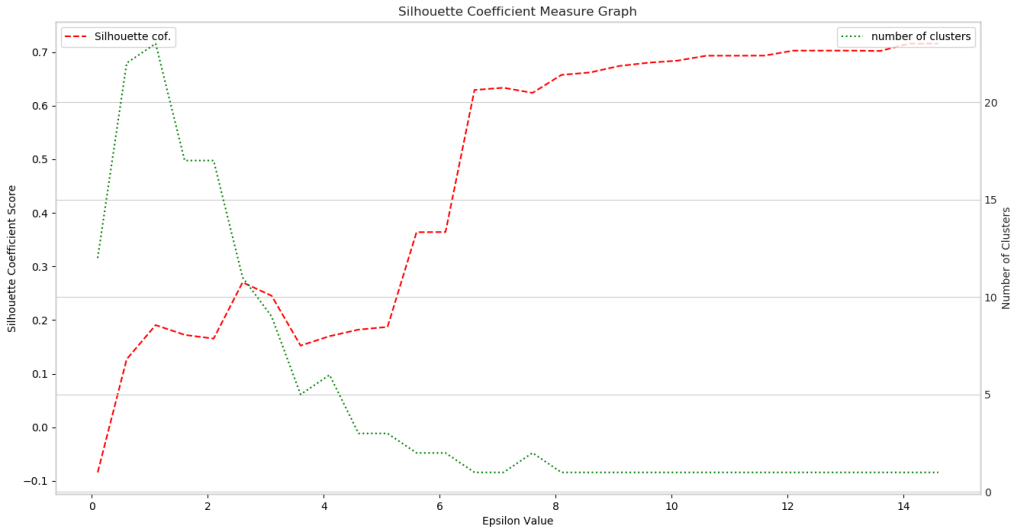


*Figure 30. Experiment 1 external validity measures plot*

Moreover, we used the silhouette coefficient measure, to illustrate that the value we obtained in Experiment **1** is the highest, considering the number of clusters we are looking for. From the graph above (see Figure 30), we can see that high values of Silhouette coefficient, depends on high values of Epsilon. Although, the number of clusters found clearly decreases to **1**, when the values of epsilon rise. Therefore, the best possible value of silhouette coefficient is **0.20**, which divides dataset into **22** clusters.

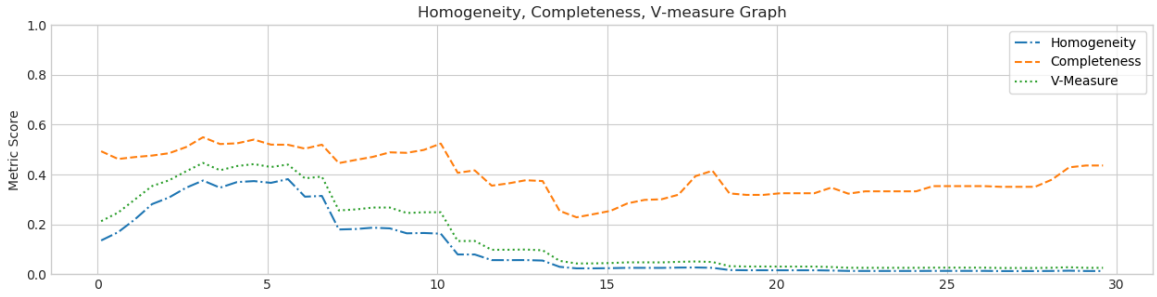


*Figure 31. Experiment 2 Internal validity measures plot*

We evaluated the results we obtained from experiment 2 as well. In the graph above (see Figure 31), we demonstrate the relation between the increasing Epsilon value with, homogeneity, completeness and V-measure scores. This graph was implemented on Bi-Grams vectors (feature set). By inspecting V-measure score, we can determine the best possible values, since it represents the harmonic mean (average) of both homogeneity and completeness scores. However, the most suitable values of all three measures are found, when epsilon value is between **[5.4, 6.6]**, which result in scores that ranges from **0.30 to 0.50.**

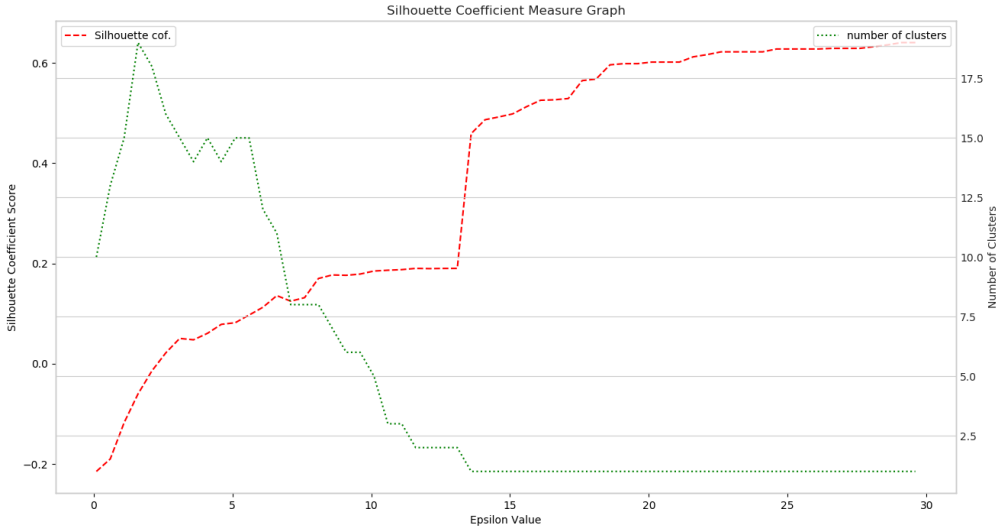


*Figure 32. Experiment 2 external validity measure plot.*

Additionally, we plotted the different Silhouette coefficient produced in experiment 2, against their corresponding Epsilon value (see Figure 32). We also, plotted the number of clusters found, at each epsilon value. Comparing both lines against each other at a certain epsilon value, illustrate the silhouette score and number of clusters produced. The highest value of silhouette coefficient, which is around 0.64, indicates that all our samples, are included in the same cluster (large epsilon value). From the graph we can determine, that the highest silhouette score, that would result in an adequate number of clusters, is found in the range of **[5.5, 6.9],** that produces up to **11** clusters.

Overall, we concluded that, the best results are achieved by using frequency vectors as our feature set, and setting the parameters to these certain values, **Eps = 1.15, and MinPts = 8**. Also, using normalised feature set, led us to the improving our clustering results.

### 5.2.5 Experiment 3

In this experiment, we used frequency vectors, to form our feature vector. The same procedure in experiment 1 is followed in here, except the distance measure.

Sine the clusters we obtained in the previous experiments, are not considered to be of high quality. We combined the Manhattan distance function with our DBSCAN algorithm, in order to discover if our results can be enhanced. At first, we normalized our feature set, since we realised that it improves the clustering results. After normalizing, we started estimating DBSCAN parameters, hence, we used the same value for **MinPts**, which is equivalent to **8**. Whereas, we used the succeeding K-distance graph in order to provide us with a suitable range of values for **Eps parameter** (see Figure 33)**.**

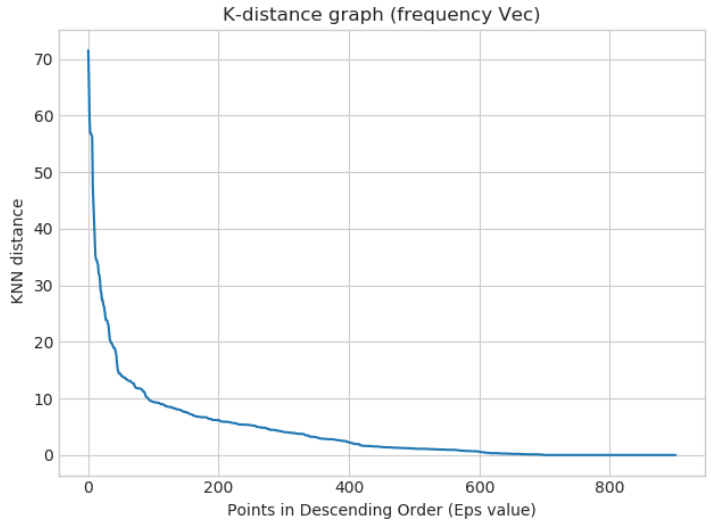


*Figure 33. K-distance graph.*

After reviewing the previous graph, the best **Eps** values are approximately found between **[2.5, 4.5].**

In the next step, we ran the algorithm ensuring that we chose the right metric function. **Eps** value was estimated to be **2.8**. As a result, our algorithm produced **19** clusters, and we evaluated the quality of clusters using, the same measures.

| measure | Accuracy | ARI | Silhouette | Precision | Recall | F-measure | Homogeneity | Completeness | V-measure |
|---|---|---|---|---|---|---|---|---|---|
| **value** | 0.117 | 0.155 | 0.232 | 0.117 | 0.117 | 0.117 | 0.503 | 0.571 | 0.535 |

It is clear that, using Manhattan distance instead of the Euclidean distance slightly improved the overall results in terms of the quality.

Moreover, we plotted the possible Silhouette coefficient scores, against the increasing values of epsilon (see Figure 34).

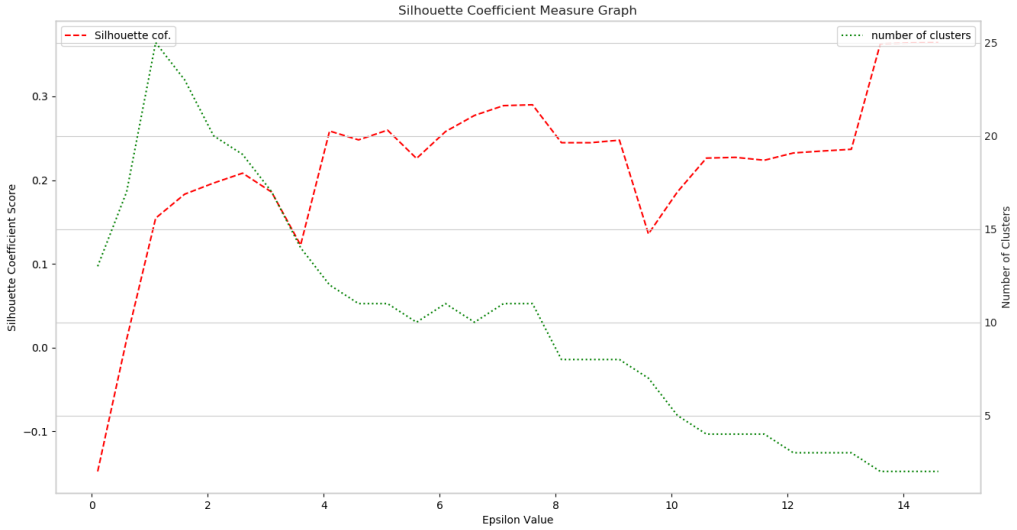


*Figure 34. Experiment 3 external validity measure plot*

In the graph above, the highest silhouette score is around **0.30**, with epsilon roughly equal to **14**. On the other hand, considering figure 30, that demonstrates the silhouette coefficient highest score we obtained in experiment **1** using Euclidean distance, being around **0.70.** Therefore, we can imply that using both distance measures provide us with considerably similar results.

## 5.3 Performance

In this section, we evaluate the performance of all the steps we performed until obtaining the final results. The performance was assessed on all the samples that represent our dataset.

| **Step** | **Time consumed** | |
|---|---|---|
| | **Frequency Vector** | **Bi-Grams** |
| **Extracting features and creating feature set.** | 35.6 s | 32.8 s |
| **DBSCAN clustering** | 0.22 s | 0.60 s |
| **Overall running time** | 35.8 s | 33.4 s |

### 5.3.1 Complexity

We The original paper of DBSCAN [9] suggests that, the average runtime complexity of the algorithm is equivalent to **O(n log n)**, where **n** represents each data point in our dataset.

As for the memory complexity, the original algorithm had a complexity of **O(n),** and since we used Sicikitlearn implementation of DBSCAN as our algorithm, it computes all neighbourhood quires, which increases the memory complexity up to **O(n × d)**, where **d** is the average number of neighbours [10].

# Chapter 6: **Discussion**

With regard to limitations that we encountered while implementing our clustering algorithm, we came across a few of them. Considering the different feature sets we used in representing our input, the problem appeared to emerge from there. Since our behavioural profiles, which were produced by CopperDroid did not include low-level system calls (very detailed) that can be thought of as the basic operations that initiated a certain behaviour. The absence of such information, caused our feature set to be of a coarse granularity.

In general, using larger granules in our feature set would result in including a few valuable information and neglecting several minor details, whereas using smaller granules, would cause our feature set to enclose an excessive amount of information [4]. By eliminating such intricate details, hence, using a coarse granular feature set, our clustering algorithm would be only able to cluster samples that are considered ambiguously similar [4]. On the other hand, when using a feature set of a finer granularity, it could possibly lead our algorithm to precisely function on the feature set provided at hand, but when evaluating different datasets, it may not produce the same level of accuracy. Therefore, adjusting our feature set and allowing it to be of a finer granularity, might be an excellent solution for the low results we obtained from our quality measures [4].

As mentioned in the drawbacks section of DBSCAN Algorithm (section **4.3.3**), which indicates that it is not capable of appropriately handling clusters with varying densities. Mainly because, its definition of core points, cannot figure out the core points when having clusters of varying densities.

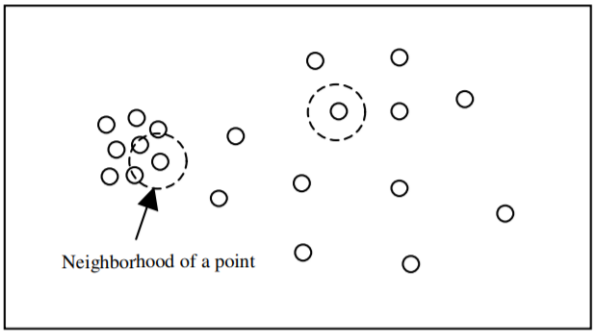


*Figure 35. Density-based Neighbourhood*

Considering the figure above (see Figure 35), if, for example, we pick a value of **Eps** and start searching for the minimum number of points within its radius, either we would treat all the points on the left side, as a whole cluster, while the points on the right side would be considered noise. Or all the points would be placed into the same cluster [15]

In our case, handling clusters of varying densities, significantly affected our algorithm's results. For example, when inspecting our feature vectors, we manged to identify some malware families, in which their behavioural profiles included a large number of behaviours (high density). While, other malware families, only contained a few behaviours within their profiles (low density), hence, these samples could be classified as noise by our algorithm.

Another obstacle that we encountered, was setting the **MinPts** parameter. Following the heuristics to estimate its value, in certain cases resulted in producing unrealistic values, or values that would not correctly collaborate with our dataset. Setting the minimum number of points parameter, for any dataset, requires a prior domain knowledge. However, being provided with the ground truth helped us in determining the best value of **MinPts**. By examining all the malware families and

getting a rough estimate on the number of samples each family contained, guided us to eliminate such impractical values.

For example, when using Bi-grams to represent our feature set, we ended up with **180** dimensions, however, setting the parameter **MinPts,** to be double that value, hence, equal to **360**. Bearing in mind that **MinPts** resembles the minimum number of samples, to signify a cluster. In our data set, neither one of the malware families contained that number of samples. Thus, we concluded that **MinPts** parameter can be set to be equivalent to **8**. Similarly, we tested our algorithm with other values, following a heuristic described in section **4.3.1**, which suggests that **MinPts** can be set to **ln(n),** where **n** is the number of samples in our dataset [3]

We evaluated the algorithm using the value of **MinPts ≈ 7,** and the output was reasonably better. Yet, we cannot claim that this heuristic would always provide accurate values, but a good combination of domain knowledge and that heuristic might yield better results.

Similarly, we tried to apply a different distance measure, to increase the quality of the clusters we obtained. Rather than using Euclidean distance, we applied the Manhattan distance function on our feature set. However, using Manhattan distance function, did not demonstrate a significant effect on the results we obtained. Instead, the clusters' quality remained the same. These low results can be claimed to be an instance of the "curse of dimensionality". Considering that as the number of dimensions grow, the effectiveness of our distance measures decreases. To properly handle such an issue, we used Principal Components Analysis. By estimating the optimal number of components, we manged to reduce number of dimensions from **180** up to **81** dimensions. And that indeed, assisted in slightly increasing the quality of clusters produced e.g. in experiment **2** using the original feature vector with **180** dimensions, provided us with considerably low validity scores, however, after applying PCA on that feature set, the results obtained, enclosed an improvement, on a small scale.

Another approach we followed was to normalise our data set before we performed clustering. Since by normalising all the samples, our dataset would be intact around certain value limits. Also, it is a widely known technique, which is performed in several machine learning problems. Normalising our dataset, in all the experiments we carried out, had a noticeable impact on the validity measures we obtained. However, in experiment 2, normalising data, did not lead to an improvement in the measures' values, instead, it decreased the obtained results.

Moreover, the method we pursued in estimating **Eps** value heavily counted on k-distance graphs produced. Choosing the accurate range of values, is easily computed by inspecting the elbow shown on each graph. However, experimenting with every value in that range, is by some means time consuming, since we also needed to evaluate the clusters' quality produced by every value. In order to solve that issue, we ran our algorithm inside a loop, where it started with **epsilon = 0.1** and it incremented its value by **0.05** in each iteration, all the validity measures' results were computed and added into different lists, and after the loop terminated, we printed all the values obtained, this method overall, helped us in finding the best Epsilon values.

As for the extraction of feature sets, we chose bit-vectors, frequency-vectors and bi-grams to form our algorithm input, however, when experimenting with these different sets, it was clear to us that using frequency-vectors resulted in a higher overall accuracy. Since by using frequency-vectors, we increased the chance of distinguishing similar malware families, as the number of times each system invocation occurred, is accounted for.

Moreover, at the early stages of extracting these features, we unintentionally neglected, the different behaviour classes, and their corresponding subclasses. In the Figure below (see Figure 36), we displayed our incorrect approach, in which we neglected such crucial information. While, we focused on only extracting the Binder transactions found in each sample. Our vectors consisted

of **27** dimensions, which led us to obtain very bad results in mostly all the validity measures, as the number of features, did not provide our clustering algorithm with enough information, in order to distinguish different families from each other.

```
{
  "behaviors":{
    "static":{

    },
    "dynamic":{
      "host":[
        {
          "tid":318,
          "procname":"m.keji.danti413",
          "class":"FS ACCESS",
          "low":[
            {
              "sysname":"open",
              "blob":"{'flags': 131649, 'mode': 384, 'filename': u'/data/data/com.keji.danti413/files/smartmad.ai\\x00'}",
              "type":"SYSCALL",
              "id":2580,
              "ts":0
            },
            {
              "sysname":"write",
              "xref":2580,
              "ts":0,
              "blob":"{'filename': u'/data/data/com.keji.danti413/files/smartmad.ai'}",
              "type":"SYSCALL",
              "id":2582
            }
          ]
        },
        {
          "tid":325,
          "procname":"Thread-9",
          "class":"ACCESS PERSONAL INFO",
          "subclass":"PHONE",
          "low":[
            {
              "method_name":"android.app.IActivityManager.GET_CONTENT_PROVIDER_TRANSACTION()",
              "type":"BINDER",
```


*Figure 36. The wrong approach we used in extracting features*

On the other hand, considering figure 5, that displays our revised approach when extracting features that clearly helped in enriching our feature set, by adding **16** new dimensions, which eventually generated better results, and even higher accuracy. However, even after we obtained 43 dimensions, we realised that by using more dimensions we could improve our algorithm's results, therefore, we implemented Bi-grams, which included **180** dimensions rather than **43**. As by using bi-grams, the series of method a malware sample invoked, can undoubtedly, increase the similarity factor found between samples of the same malware family. And finally, the inability of displaying (plotting) high dimensional data, was troubling at early stages. Considering that primarily, 2-dimesnional and 3-dimensional feature sets are the only dimensional data that can be easily displayed and analysed, however in our case we had more than **40** dimensions, or in other cases more than **170** dimensions. We had to rely on other approaches, for example, when having two-dimensional dataset, after clustering, noise points could be easily identified, also the number of clusters would be clearly visible, however, we had to compute the number of noise points found, by writing a few lines of code and so forth.

In global context, working on this project allowed us to experiment with different approaches, on different levels, and demonstrated the effects of each optimisation we made, on the results we obtained. For example, when using DBSCAN algorithm, adjusting the value of either one of its parameters, could significantly provide us with different set of clusters unlike the ones we are aiming to get, for example, increasing Epsilon value would clearly merge two clusters together if they were placed next to on another.

Another example, is the use of different feature sets, bit-vectors for example, did not provide us with the ability of navigating through the whole data space, since it did not provide precise

information on each dimension. Also, the effect of standardizing the values before clustering considerably enhance our results... etc.

Nevertheless, working throughout the requirements of this project, highlighted the usefulness of Machine learning algorithms, such as PCA, Standardization, Nearest Neighbour. Likewise, different types of clustering proved to be beneficial in different situations and when dealing with different feature sets. By combining all these algorithms, we managed to distinguish malware families, based on the actions found in their behavioural profiles, also, we were able to determine a set of common behaviours that are shared, for example, among two malware families.

# Chapter 7: **Professional Issues**

## 7.1 Plagiarism

Plagiarism, is the act of copying, using, other people's work, ideas, expressions, without identifying their effort. According to Merriam-Webster online dictionary, to plagiarise is described as, "*to present as new and original an idea or product derived from an existing source*" [23]. However, as claimed by O'Conner, when an individual's work contains more than a certain percentage, it is expressed as extreme plagiarism, which she defined as *"Student work that comprises more than 80% unacknowledged, copied material"* [21].

Plagiarism can occur by performing any of the following actions, submitting other people's work as your own, providing incorrect information about sources used, using images or figures from any website or research paper, without acknowledging the source, and many more [23]. In academia, committing such an action is considered to fall in the academic dishonesty or misconduct domain. Additionally, the continuation of committing plagiarism can actually lead to being disqualified [7].

There are some factors that lead to plagiarism, as stated by O'Conner [21], age, gender and cultural background, may have a slight or no bearing on the probability of being engaged in plagiarism. As disregarding cultural generalization(stereotyping) is crucial, different cultural values demonstrate distinct approaches regarding academic writing and plagiarism [21]. Taking the Japanese culture as an example, Japanese philosophers and well-known authors had contributed in writing their words of wisdom. However, these words are treated as absolute facts, and when using them, the need of citing their source is nonessential [21].

To avoid carrying out such an action, individuals should clearly cite every source that helped in forming their ideas. Hence, when presenting an author statement or referring to a certain idea proposed by this author, a clear quotation marks should surround that statement. And a corresponding bibliography entry that contains thorough information about the source should be included. Likewise, when using an author's idea but not his exact wording, his name and information should be declared [39]. As far as how to cite other people's code, essentially, adding a comment indicating that this piece of code is not your own work, and that it's been used, or borrowed from a source. Thus, writing the name of the author along with a working link to the original source, and highlighting where that code begins and ends, should be sufficient citation [20].

Furthermore, plagiarism have a major impact on both ethical and practical aspects. As discussed above, that plagiarism is considered a form of stealing, thus, it clearly effects the person's reputation. Getting caught after plagiarising would surely damage an individual's credibility along with reducing the possibility of him being chosen in contributing in significant projects, for example [18]. It can also damage his relationship with other co-workers, as they would not prefer working with a nonprofessional person [7]. As for practical aspects, when a company, for example, produces a software that compromises the idea of another product in the market, without giving any credit, this company can be legally prosecuted, which will eventually destroy the company's image and reputation among its competitors [7].

If this issue is not addressed properly, it will result in chaos on several aspects, which implies that copyrights will lose their value. Also, the task of distinguishing who wrote what, and whether an information is scientifically proven and does not contain any false assumptions, will be complicated. In academia, a student, for example, would copy another student's work and get high marks while, it wasn't his own work and so on. Likewise, a person could take credit for a software that he did not write, and once bugs or problems with the code arise by different users, who tested the product, the possibility that those issues will be rightfully addressed is quite low.

While working on this project, finding sources was a considerably straightforward task to accomplish, however, the writing process required more time and attention. Different types of resources, books, research papers, blogs, articles, and websites, have been used in writing this paper. Nonetheless, each source has been clearly referenced both in text and in bibliography. Also, when writing programs, there exists some online implementations illustrating how to use certain methods from a public library, such as Sicikitlearn. The intention of writing such programs is to simplify the idea to different users, of different background knowledge. However, when writing DBSCAN algorithm, we reviewed the demo page of the algorithm, to get more familiar with how to use some particular methods and how to display(plot) data. Acknowledging the source of each code fragment used is crucial, since by doing that, we are actually signifying that some of our ideas were borrowed or actually built on top some existing criteria. Additionally, as indicated in the Project booklet [39], the amount of reused code, should not exceed what a student wrote, so highlighting that limit is beneficial to the students.

And it also supports the concept of code reusability, which signifies that some implementations have been produced, and perfected over time, so the idea of re-producing a certain clustering algorithm, for example, will consume both time and effort, and might not cover all different aspects of the algorithm. Instead, using such publicly available implementations, with clear indications to where this data is taken from (references), provides correctness, precision, and consumes less time.

In general, by enforcing standards, and raising awareness, the possibility that people would commit such mistakes will obviously reduce. Moreover, in academia, allowing students to practice good citation, and setting some rules beforehand, would change the way that these students will interpret and use information later in their lives. Also, highlighting the consequences of plagiarising, will alarm people not to carry out such an action.

# Chapter 8: **Conclusion**

In this paper we provided a detailed background on Android Malware, we described CopperDroid tool that is used to Dynamically analyse different malware samples. We also highlighted the differences between supervised and unsupervised machine learning problems. Additionally, we introduced Principal Components Analysis, as a technique that is used to reduce dimensionality, and included a mechanism that helps in determining the number of required components.

We also provided an overview on a number of different clustering algorithms and their underlying structures, implementations, along with their strengths and weaknesses.

In addition, we discussed several validity measures, which were used to evaluate the efficiency and time performance of our algorithm. And finally, we conducted a couple experiments on our dataset, using different feature sets, and distance measures.

We also illustrated how our feature sets were created, and the procedure we followed in order to cluster data. we showed our final results and the accuracy level we achieved. As we argued in our paper, several limitations that affected our overall results, and how we could fix them.

However, in the following section, we suggest some future work that would enhance our algorithm's performance, in terms of results and time consumption.

## 8.1 Future Work

Consdering the obstacles we encountered in the previous chapter, we concluded a number of fixes that, will enhance the performance and quality of our algorithm. Since estimating the paramter values, was a main concern, throughout the process of clustering and when evaluating the results. Eliminating the process of estimating the parameters needed, and implemnting an algorithm that will automatically process the data without the need of explitly setting the parametres. Which can be accomplished by using OPTICS algorithm, for example, since it doesn't require **epsilon** vlaue.

Another area we can work on is the time performacne. It was estimated by, recording the time in which the program satrted proccessing data, until it produced the clusters. It is apearnet that the feature extarctaion part consumes more time compared to any other part of the program, so fixing that issue and trying different approaches to extract features can clearly improve our time performance.

As for our main issue, which was accuarcy, using a feature set of finer granuality, is the ideal way to increase accuacy results.

# Bibliography

[1] Ansari, Z. and Azeem, M.F. and Ahmed, W. and Babu, A. *Quantitative Evaluation of Performance and Validity Indices for Clustering the Web Navigational Sessions*. World of Computer Science and Information Technology Journal. 2011

[2] Bayer, U. and Comparetti, P.M. and Hlauschek, C. and Kirda, E. and Krügel, C. *Scalable, Behavior-Based Malware Clustering*, NDSS, 2009.

[3] Birant, D., Kut, A. *ST-DBSCAN: An algorithm for clustering spatial–temporal data*. 2007.

[4] Ding, S. and Du, M. and Zhu, H. *Survey on granularity clustering*. 2015.

[5] Gupta, A. and Kuppili, P. and Akella, A. and Barford, P. *An Empirical Study of Malware Evolution*. University of Wisconsin-Madison, 2009.

[6] Han, J. *Cluster Analysis in Data Mining*. n. d.

[7] Hirby, J. *What Are Some Consequences of Plagiarism?* Available from: https://thelawdictionary.org/article/what-are-some-consequences-of-plagiarism/

[8] James, G. and Witten, D. and Hastie, T. and Tibshirani, R. *An Introduction to Statistical Learning*, 2013.

[9] Sander, J. and Ester, M. and Kriegel, H.P. and Xu, X. *Density-based clustering in spatial databases: The algorithm GDBSCAN and its applications*. Data Mining and Knowledge Discovery ,1998.

[10] Scikit-learn: Machine Learning in Python, Pedregosa et al., JMLR 12, pp. 2825-2830, 2011.

[11] Kanungo, T. Mount, D. Netanyahu, N. and Piatko, C. and Silverman, R. and Wu, A. *An Efficient k-Means Clustering Algorithm Analysis and Implementation*. 2002.

[12] Karamizadeh, S. and Abdullah, S. and Manaf, A. Zamani, M. and Hooman, A. *An Overview of Principal Component Analysis*. University Technology Malaysia, Malaysia, 2013.

[13] Kikuchi, Y. and Mori, H. and Nakano, H. and Yoshioka, K. and Matsumoto, T. and Van Eeten. *Evaluating Malware Mitigation by Android Market Operators*. Yokohama National University, Japan, 2016.

[14] Kriegel, H.P. and Kröger, P. and Sander, J. and Zimek, A. *Density-based Clustering*. WIREs Data Mining and Knowledge Discovery. 2011.

[15] Levent, E. and Steinbach, M. and Kumar, V. *Finding Clusters of Different Sizes, Shapes, and Densities in Noisy, High Dimensional Data*. 2003.

[16] Liu, Y. and Li, Z. and Xiong, H. and Gao, X. and Wu, J. *Understanding of Internal Clustering Validation Measures*. 2010.

[17] Madhulatha, T.S. *An Overview on Clustering Methods*. Alluri Institute of Management Sciences, Warangal, 2012.

[18] McQuerrey, L. *Why Is Plagiarism an Important Issue in the Workplace?* Available from: https://woman.thenest.com/plagiarism-important-issue-workplace-14333.html

[19] Namratha, M. and Prajwala, T.R. *A Comprehensive Overview of Clustering Algorithms in Pattern Recognition.* Visvesvaraya technological university, India, 2012.

[20] Okoli, C. *How to Cite Code.* Available from: http://chitu.okoli.org/psyche/it/how-to-cite-code/

[21] O'Conner, Z. *Extreme plagiarism: The rise of the e-Idiot?* International Journal of Learning in Higher Education. 2015.

[22] PIERCY, C. *Embedded Devices Next on the Virus Target List*. Electronic Systems and Software, 2005.

[23] Plagiarism.org. *What is Plagiarism.* 2017. Available from: http://www.plagiarism.org/article/what-is-plagiarism.

[24] Rajaraman, A and Ullman, J. D. in Mining of Massive Datasets, Cambridge: Cambridge University Press, 2011, pp. 213–251.

[25] Raschka, S. *About Feature Scaling and Normalization*. 2014. available from: https://sebastianraschka.com/Articles/2014_about_feature_scaling.html

[26] Reina, A. and Fattori, A. and Cavallaro. *L. A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviours*. 2015.

[27] Rosenberg, A. and Hirschberg, J. *V-Measure: A conditional entropy-based external cluster evaluation measure*. 2007.

[28] Schubert, E. and Sander, J. and Ester, M. and Kriegel. H.P, and Xu, X. *DBSCAN Revisited, Revisited: Why and How You Should (Still) Use DBSCAN*. 2017.

[29] Shah, H. and Napanda, K. and D'mello, L. *Density Based Clustering Algorithms.* Dwarkadas J. Sanghvi College of Engineering, India, 2015.

[30] Sujan, D. *Effect of different distance measures in result of cluster analysis*. 2015.

[31] Svajcer, V. *When Malware Goes Mobile.* SophosLabs, Sophos, 2013.

[32] Tam, K. and Khan, S.J. and Fattori, A. and Cavallaro, L. *CopperDroid: Automatic Reconstruction of Android Malware Behaviours*, 2015.

[33] Unuchek, R. *Mobile malware evolution 2016.* https://securelist.com/files/2017/02/Mobile_report_2016.pdf. Accessed: 10-11-2017.

[34] VanderPlas, J. *In Depth: Principal Component Analysis*. n. d.

[35] Walde, S. *Experiments on the Automatic Induction of German Semantic Verb Classes.* University of Stuttgart, Germany, 2003.

[36] Wu, D. and Mao, C. and Wei, T. and Lee, H. and Wu, K. *DroidMat: Android Malware Detection through Manifest and API Calls Tracing*. 2012.

[37] Zhou, Y. and Jiang, X. Dissecting Android Malware: Characterization and Evolution. 2012.

[38]*** Available from project description.**

[39]***Available from Rules and Guidelines 2017-2018.**

# Appendix A (interim)

Structure of submitted directory:

- Documents:
    - ZBVA200.interim (word format)
- Code:
    - DBSCAN
        - DBSCAN.py (doesn't work yet)
    - Proof of Concept:
        - src (all programs are written in python)
            - BitVector.py
            - FrequencyVector.py
            - ParseJson.py
        - progCode.zip (contains all code zipped)

          once unzipped it has Feature Extraction folder, which has src, test and DBSCAN folders
    - samples + (samples.zip)
        - ADRD_genome_stimulated
        - AnserverBot_genome_stimulated

**Diary**

An overview of work accomplished throughout this smester (updated regularly) is included in the diary provided by department; https://pd.cs.rhul.ac.uk/2017-18/blog/author/zbva200/

productive meetings with supervisor, were made every 2 weeks, starting from 28th of Sep, throughout autumn term.

**Proof of Concept Programs**

In general, several program implementations have been written. These programs mainly focused on feature extraction procedures, to obtain variables that can be used as input for clustering algorithm. All of which, can be found on https://github.com/ghadashehri/Final-Year-Project17-18/tree/master

Following software engineering approach when writing each of the programs, helped in reducing the number of bugs the code contains, which was achieved by writing a test case for each class, that checks the correctness of results obtained, also using checkstyle (pep8) to ensure that naming variables, methods, classes are all done appropriately. Additionally, commits were made regularly to GitHub repository.

# Appendix B (final)

The subdirectory structure is as follows:

- Reports
    - ZBVA200.final.pdf
- Programs
    - DBSCAN

      Includes the following python scripts:

        - BiGrams Vectors
        - Bit-Vectors
        - Frequency Vectors
        - Dict Vectorizer

      Also,

        - 4 K-Distance classes
        - 2 PCA classes
        - 4 DBSCAN classes
        - 2 DBSCAN Evaluation classes

***REQUIRED LIBRARIES:**

**Numpy, matplotlib. sklearn, pandas**

**All the programs are written in python.**

**The dataset is also submitted with the files (samples.zip)**

**NOTE: All proof of concept programs, include variable called (directory is str), that represents the location of the dataset, hence, should be changed accordingly.**