Final Year Project Report

Full Unit – Interim 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

December 01, 2017

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 Where other published and unpublished source materials have been used, these have been acknowledged.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: 5034 (excluding Appendix and List of Figures)

Student Name: Ghadah Alshehri

Date of Submission: 1/12/2017

Signature: GhadaShehri

Table of Contents

List of Figures:

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.

Project Specification

Aims: To implement clustering for recorded behaviour traces of Android applications towards distinguishing benign from malicious behaviour (malware) [18].

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 [18].

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 [18]

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 [18]

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 new technologies and features, lead to this success and popularity [6].

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" [16]. 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 [1]. 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 [1].

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 [2]. 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 [14]. Another reason to consider, is Android permitting users to install applications form third-party sources, which are mainly used to distribute malware [7].

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 [6]. 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 [6]. 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 [2,10].

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 [2].

Chapter 2: **Android**

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 [11].

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) [11]. 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 [11].

2.1 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 [11]. 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 [15].

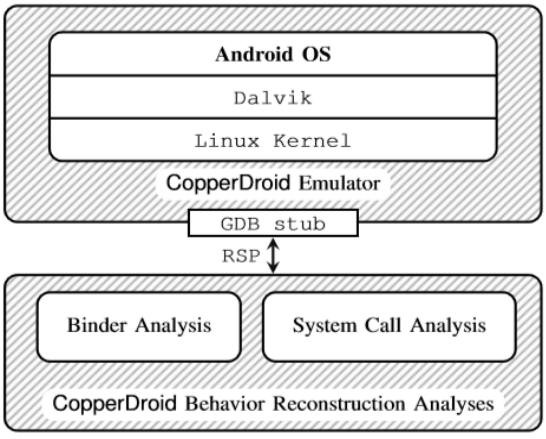

Figure 1. CopperDroid's Architecture

The structure of CopperDroid (see Figurer 1.), 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 [11].

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**) [11].

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 [15]. By examining communication that occurs over these channels, CopperDroid is able to distinguish the characteristics of a malicious application [11].

2.1.1 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 [15].

2.2 Feature Extraction

2.2.1 Parsing. json files

Files produced by CopperDroid, .json files contain a detailed profile on each application malicious behaviour. Extracting Features from .json files, in a suitable format, such as vectors, which will represent the input to our clustering algorithm and since the dataset contains multiple Android malware families; each family have several samples. Thus, the values of each family should be separated from others.

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, describes whether a certain feature is present (1) or not (0). Moreover, the parsing programs inspects Binder, Intent, and Syscall, and will use them as entries in our feature vector.

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. Supervised learning, acquire data inputs, and provide estimation (prediction) of their output (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) [3].

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 assembling data to distinct groups. Its fundamental goal, is to discover significant aspects about the observations provided. Thus, Clustering, is a technique, that simplifies the process of determining subgroups in dataset [3].

In case of large datasets (High dimensionality), Principle Components Analysis, which is a technique of unsupervised learning. Is considered to be a useful tool that aids in visualising data, and pre-processing it, as its reduces the number of dimensions [3].

3.1 Unsupervised Learning:

3.1.1 Principal Components Analysis (PCA)

PCA Is a widely known technique, that's used for reducing dimensionality, and decomposing a large dataset, that has a huge 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 parameters of a dataset, using PCA will result in using, for example, 2 values at most that will eventually provide a precise representation of data [3].

Some of the benefits of using PCA can be summarised in few points,

- It clearly reduces the capacity and memory used.
- It decreases the complexity of a system.
- 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 [5].

Requirements to perform PCA:

1. Scaling the variables. Before performing PCA on observations, all data, should be scaled (using a certain method, such as standard deviation = 1). Scaling is important since; different variables are measured in different units, so when calculating the variance of each variable, it will result in first principal component loading vector to have, for example, large loading, since it had the largest variance. Mostly, each variable is scaled to have unit standard deviation when the possibility of them having different units is high [3]. Hence, scaling has a substantial impact on obtained results, and should be performed beforehand.

2. The minimum number of Principal Components to use is calculated via, **min(n-1, p),** where **n** is the number of observations and **p** is the number of variables. However, it is crucial that we use the very first PCA's to help in visualising data [5].

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 [3].

Nonetheless, Clustering is unsupervised problem, because it's applied to determine different clusters (can be thought of as 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 [17].

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, since datasets may have high dimensionality, and the results of clustering algorithms can be viewed in different ways [3].

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 [17].

However, since every clustering algorithm follows a different set of rules for determining the similarity measure between observations. The following section will discuss, 3 of the most popular algorithms used, which are, K-means, Hierarchal and Density-based Algorithms.

4.1 K-means Clustering Algorithm:

This algorithm uses a pre-defined value of K, which specifies the number of distinct clusters dataset should be divided into. Choosing the correct value of K is significant, since the algorithm assigns each observation to only one of K-clusters [3].

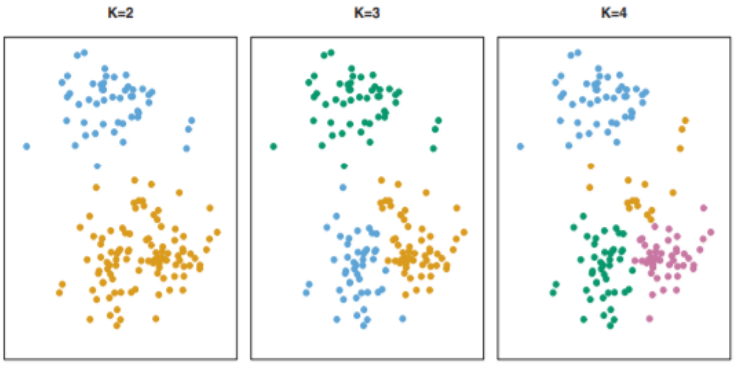

Figure 2. applying K-means clustering with different values of K

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.

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 distance … etc.

An abstract way to describe K-means clustering algorithms, used *from Introduction to statistical learning* book [3], is as follows:

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.

Centroid, is the mean of observations assigned to each cluster (see Figure 3).

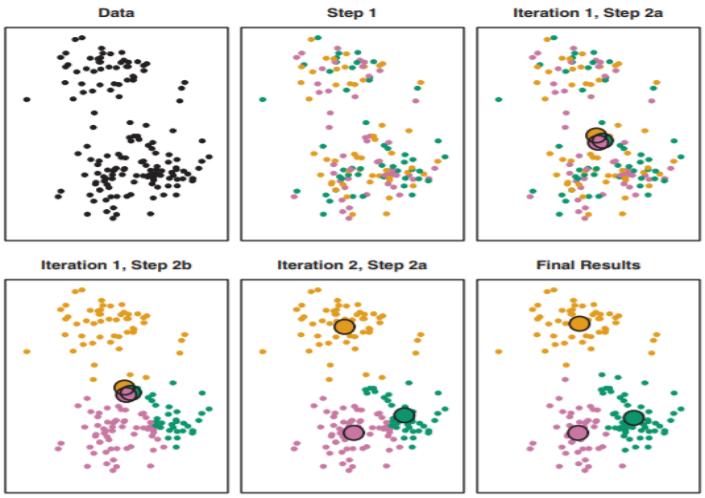

Figure 3. The progress of K-means Algorithm, K=3

4.1.1 Advantages:

Nevertheless, using K-means is found to be efficient with datasets of high dimensionality, it is also considered to have an easy implementation, and has high performance(fast) [9].

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 [3, 9].

4.2 Hierarchal Clustering Algorithm:

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 dendrogram. It has two types, Agglomerative, and Divisive [3].

Agglomerative, which is also known as "bottom-up, the way it works is that each observation initiates its own cluster, and as moving upwards (the top), pairs of clusters gets merged. Whereas, Divisive "top-down", all data are in the same cluster, and it will start splitting them up, while moving down the hierarchy. However, using Agglomerative is less problematic, since Divisive needs to consider all possible divisions [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 [3].

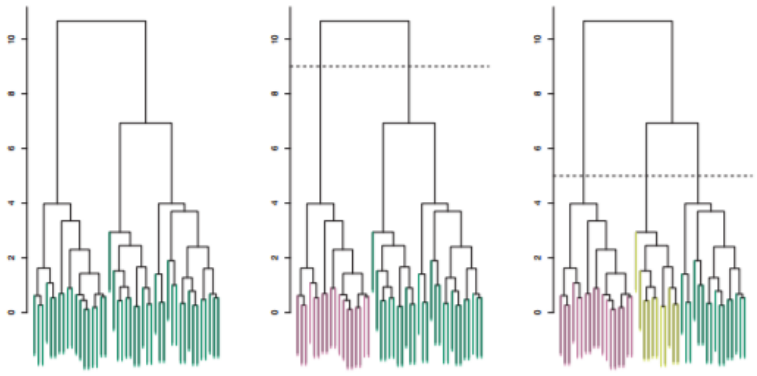

Figure 4. dendrograms obtained from hierarchical clustering

To discover number of clusters a dendrogram has, a horizontal cut at a relevant depth, and depending on where the horizontal line is, the sets (major branches) found below that line, represent the different clusters (see Figure 4) [3,8]. 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 to be found [8].

Different values of where the horizontal line cuts the dendrogram, would indicate a different number of clusters, i.e. in Figure 4, 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, clusters are set to 3 in that case.

4.2.1 Dissimilarity measure:

Which is calculated using variety of methods. i.e. Euclidean distance. Where it considers the distance between two data points. Computed 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 [8].

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, 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) [3].

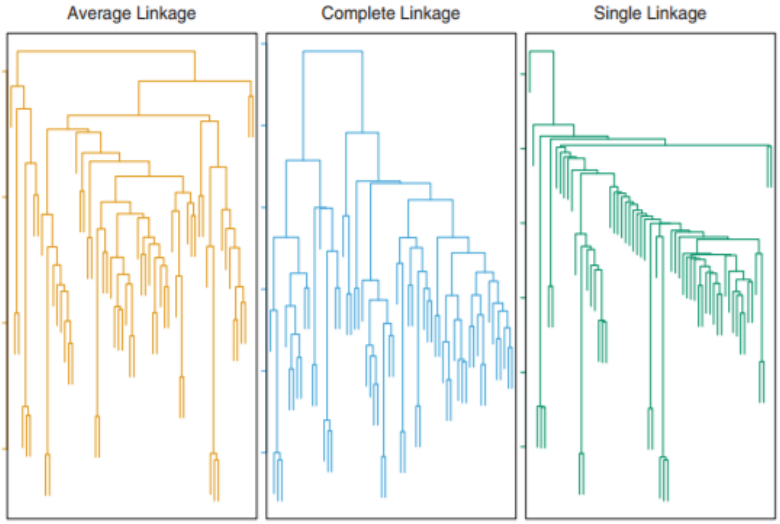

Figure 5. Shows different Linkage methods used

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

4.2.3 Advantages:

It does not require the user to specify the value of K before starting the algorithm. When using Agglomerative clustering, 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 has access to all data [9].

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 [4]. Moreover, determining where the horizontal line should be placed, counts as a disadvantage. As it's in many cases, may seem ambiguous. When using Divisive, several computational difficulties emerge particularly when splitting the clusters [9]. Using Centroid linkage, can cause an inversion, which will result in more difficulties, especially in visualising and analysing data [3].

However, hierarchal algorithms sometimes can lead to less accuracy when compared with K-means [3]. Depending on the distance metric used, the results would vary, and if the data grouping is incorrectly done in earlier stages, the results cannot be altered after [9].

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. Moreover, data points that exist in low-density areas, are considered either noise, or outliers [4].

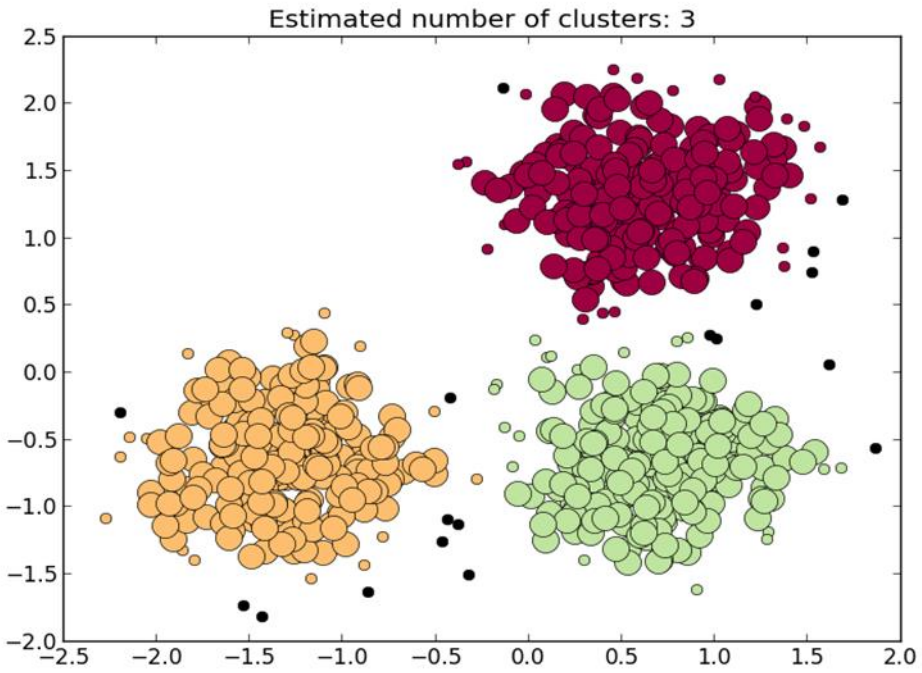

Figure 6. Plot of DBSCAN performed on dataset

The DBSCAN Algorithm needs 2 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 [8].

There are some significant definitions made by Shah, et al., 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. $|Eps(q)| \geq MinPts$, the number of points in q's neighbourhood is greater than or equal to MinPts(Core point condition).

This relation, is guaranteed to be transitive (see Figure 7) [4].

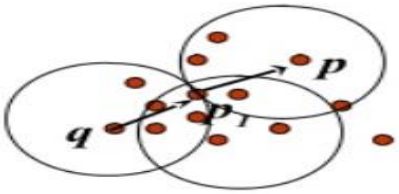

Figure 7. Density-reachability

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 8) [4].

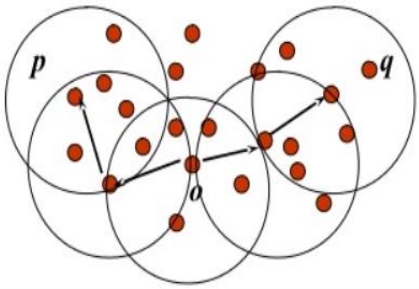

Figure 8. Density-Connectivity

An abstract approach to illustrate DBSCAN clustering algorithm [12]:

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 [4].

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. [4] 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 [12].

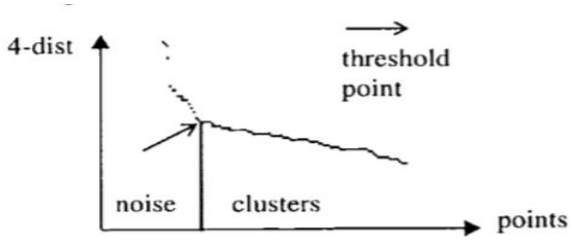

Figure 9. An example of k-dist. graph

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. [4] 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 9) [4].

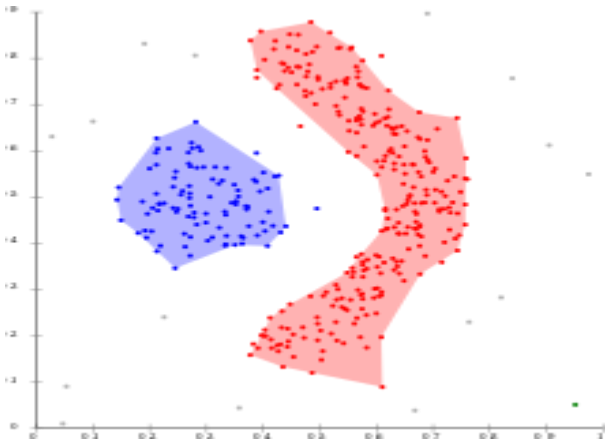

Figure 10. DBSCAN result, that has one cluster surrounded by another

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) [12].

The algorithm uses 2 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 [4]. It can even find a cluster completely surrounded by (but not connected to) a different cluster (see Figure 10).

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 [12]. 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 [12]. In the case of ambiguous dataset (that aren't understood in terms of data and scale used.), the process of computing Eps parameter can be found difficult [13].

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.

Timescale and Summary of Completed Work:

Following the Project plan, which was submitted at the beginning of Autumn term, setting tasks for each week of term, helped in keeping up working through out term.

Task	**Started/ Finished**	**Status**
Writing a report on background information, i.e. Android, Malware	5- 10 Oct	done
Writing a report on CopperDroid, including its architecture, and why it is helpful	11-15 Oct	done
Viewing some Machine Learning concepts, and figuring out how each mechanism should be used.	18-24 Oct	done
Learning how to program in Python, since it has many existing Machine learning implementations (Libraries).	20-26 Oct	done
Analysing samples provided, and going through .json files, which are produced by CopperDroid. To understand which information will be useful, when clustering.	27-29 Oct	done
Experimenting with .json files.	27-29 Oct	done
Writing a report on Clustering, what it means, and how different algorithms function.	7-10 Nov	done
Writing about the importance of scaling and Principal Components Analysis(PCA), since its filters data, and reduce high dimensionality.	12-15 Nov	done
Continue writing on K-means, hierarchical and DBSCAN algorithms, adding their advantages and disadvantages.	15-21 Nov	done
Started Parsing .json files of each malware family, then creating a Feature Vector, which represent each family's behaviour.	22-25 Nov	done
Implementing a frequency vector, as well as, bit-vector.	25-27 Nov	done
Implementing vectors, by using CountVectorizer. (Efficient way)	27-30 Nov	not
Adding information to the presentation.	30 Nov	done

Overall, early deliverables that are required throughout this term, are accomplished. Further work is still to be made regarding, the clustering algorithm (DBSCAN), visualising its result, comparing different features and evaluating the algorithm's accuracy.

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.

Bibliography

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

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

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

[4] 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.

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

[6] 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.

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

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

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

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

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

[12] 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.

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

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

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

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

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

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

Appendix A

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

GitHub Repository:

https://github.com/ghadashehri/Final-Year-Project17-18/tree/master