Fuzzer Test Log Analysis Using Machine Learning

Framework to analyze logs and provide feedback to guide the fuzzer

Jyoti Yadav
Abstract

In this modern world machine learning and deep learning have become popular choice for analysis and identifying various patterns on data in large volumes.

The focus of the thesis work has been on the design of the alternative strategies using machine learning to guide the fuzzer in selecting the most promising test cases. Thesis work mainly focuses on the analysis of the data by using machine learning techniques. A detailed analysis study and work is carried out in multiple phases. First phase is targeted to convert the data into suitable format (pre-processing) so that necessary features can be extracted and fed as input to the unsupervised machine learning algorithms. Machine learning algorithms accepts the input data in form of matrices which represents the dimensionality of the extracted features. Several experiments and run time benchmarks have been conducted to choose most efficient algorithm based on execution time and results accuracy. Finally, the best choice has been implanted to get the desired result. The second phase of the work deals with applying supervised learning over clustering results. The final phase describes how an incremental learning model is built to score the test case logs and return their score in near real time which can act as feedback to guide the fuzzer.

The thesis work has been carried out at Ericsson AB, Kista.

Keywords

Similarity Measures; Distance function; Machine learning; Clustering; Classification; Benchmarking
Abstract

I denna moderna värld har maskininlärning och djup inlärning blivit populärt val för analys och identifiering av olika mönster på data i stora volymer.

Uppsatsen har fokuserat på utformningen av de alternativa strategierna med maskininlärning för att styra fuzzer i valet av de mest lovande testfallen. Examensarbete fokuserar huvudsakligen på analys av data med hjälp av maskininlärningsteknik. En detaljerad analysstudie och arbete utförs i flera faser. Första fasen är inriktad på att konvertera data till lämpligt format (förbehandling) så att nödvändiga funktioner kan extraheras och matas som inmatning till de oövervakade maskininlärningsalgoritmerna. Maskininlärningsalgoritmer accepterar ingångsdata i form av matriser som representerar dimensionen av de extraherade funktionerna. Flera experiment och körtider har genomförts för att välja den mest effektiva algoritmen baserat på exekveringstid och resultatnoggrannhet. Slutligen har det bästa valet implanterats för att få önskat resultat. Den andra fasen av arbetet handlar om att tillämpa övervakat lärande över klusterresultat. Slutfasen beskriver hur en inkrementell inlärningsmodell är uppbyggd för att få poäng i testfallsloggarna och returnera poängen i nära realtid vilket kan fungera som feedback för att styra fuzzer.

Avhandlingsarbetet har utförts på Ericsson AB, Kista.

Nyckelord

Likhetsåtgärder Avståndsfunktion; Maskininlärning; klustering; Klassificering; benchmarking
Acknowledgement

I would like to express my sincere gratitude to my supervisor András Méhes at Ericsson Research. He has been my mentor and has been of immense help and support throughout the work done. His data science skills have proved to be very helpful and valuable for me to understand the data science concepts since I had very basic knowledge about data science and machine learning techniques when I started working on this project. András you have been an inspiration and I have always learned something new from you!

I would like to thank my manager Joakim Jardal for showing faith and provide the opportunity and necessary support to work on this project. I would like to thank my team at Ericsson for their help in understanding the current system architecture and help whenever needed.

I would like to thank my thesis examiner Prof. Mihhal Matskin for his support and valuable guidance during the project. Your support during the entire master’s programme has been very significant.

I would also like to thank my family and friends for their moral support during the period of my study.

Jyoti Yadav
November 2018
# Table of Contents

1 Introduction .................................................................................................................. 7
  1.1 Background .............................................................................................................. 8
  1.2 Problem ................................................................................................................... 8
  1.3 Purpose .................................................................................................................... 9
  1.4 Goal ........................................................................................................................ 9
    1.4.1 Benefits, Ethics and Sustainability ................................................................. 9
  1.5 Methodology / Methods ......................................................................................... 9
  1.6 Delimitations ......................................................................................................... 10
  1.7 Outline .................................................................................................................... 10

2 Theoretic Background ................................................................................................ 11
  2.1 Similarity Measures ............................................................................................... 11
  2.2 Machine Learning Techniques ............................................................................. 14
    2.2.1 Clustering: An Introduction ............................................................................ 14
    2.2.2 Types of Clustering ....................................................................................... 15
      2.2.2.1 Hard Clustering ..................................................................................... 15
      2.2.2.2 Soft Clustering ..................................................................................... 15
    2.2.3 Clustering Algorithms .................................................................................... 15
      2.2.3.1 DBSCAN ............................................................................................... 15
      2.2.3.2 HDBSCAN .......................................................................................... 17
  2.3 Cluster visualization using TSNE ........................................................................... 17
  2.4 Classification ......................................................................................................... 18
    2.4.1 classification algorithms ................................................................................. 19
      2.4.1.1 Random forest ...................................................................................... 19
      2.4.1.2 SVM (Support vector machine) ............................................................. 20

3 Web Framework Architecture Design ...................................................................... 21
  3.1 Workflow Design ................................................................................................... 23

4 Framework Development ......................................................................................... 24
  4.1 Milestone M1 ......................................................................................................... 24
  4.2 Step-wise process of development of learning model for Milestone M1: 25
    4.2.1 Data pre-processing ....................................................................................... 26
    4.2.2 Feature extraction ......................................................................................... 26
    4.2.3 Distance matrix ............................................................................................. 26
      4.2.3.1 Benchmarking various implementations of Levenshtein distance: ......... 26
      4.2.3.2 Distance function benchmarking conclusion: ......................................... 28
    4.2.4 Applying Clustering algorithms ................................................................. 29
      4.2.4.1 DBSCAN ............................................................................................... 30
      4.2.4.2 DBSCAN cluster Visualization using t-SNE ......................................... 32
      4.2.4.3 Challenges with DBSCAN ................................................................. 34
      4.2.4.4 HDBSCAN .......................................................................................... 34
  4.3 Evaluation of Milestone M1 .................................................................................. 35
  4.4 Milestone M2: Combining the results (another level of clustering) ................. 35
    4.4.1 Experiments with Classifiers: ....................................................................... 35
  4.5 Milestone M3: provide testcase feedback based on cluster size .................... 39

5 Evaluation ................................................................................................................... 42
5.1 Results

6 Conclusions and Future work

6.1 Conclusions

6.2 Future work

7 References

Appendix A

Appendix B
# Table of Figures

**Figure 1.** Levenshtein distance function.................................................................6
**Figure 2.** Clustering with DBSCAN .................................................................10
**Figure 3.** Steps involved in supervised ML..................................................13
**Figure 4.** High level application overview...................................................15
**Figure 5.** High level overview diagram of system..........................................17
**Figure 6.** Step-wise process of development of learning model..................19
**Figure 7.** Code example for benchmarking of distance functions ..........22
**Figure 8.** Runtimes for different distance functions........................................23
**Figure 9.** Code example of DBSCAN algorithm.............................................24
**Figure 10.** Sample output of DBSCAN Algorithm..........................................25
**Figure 11.** Sample code for visualization using t-sne.................................26
**Figure 12.** Cluster visualization with a range of perplexity.........................27
**Figure 13.** HDBSCAN example usage..............................................................30
**Figure 14.** Sample code for evaluating classifiers........................................31
**Figure 15.** Run time benchmarking of (SVC, RF) classifiers......................32
**Figure 16.** Sample code for embedding............................................................32
**Figure 17.** Sample code for Random Forest classifier for training and prediction .................................................................33
**Figure 18.** Sample code for scoring.................................................................35
**Figure 19.** Scoring results..................................................................................37
## Acronyms and Abbreviations used

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ML</td>
<td>Machine Learning</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>Density Based Spatial Clustering of Application with Noise</td>
</tr>
<tr>
<td>HDBSCAN</td>
<td>Hierarchical Density-Based Spatial Clustering of Applications with Noise</td>
</tr>
<tr>
<td>RF</td>
<td>Random Forest</td>
</tr>
<tr>
<td>SVC</td>
<td>Support Vector Classifier</td>
</tr>
<tr>
<td>NLP</td>
<td>Natural Language Processing</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
</tbody>
</table>
List of Tables

Table 1. Example of distance matrix optimization
Table 2. Example of distance matrix update
1 Introduction

With the increase in computing resources the systems are producing logs on a massive scale. This is very much applicable in telecom domain where logs up to scale of terabytes are produced every day. It is not possible to manually analyze the log data. The traditional techniques of analyzing the log data are becoming less relevant and machine learning is playing an important role from analysis to prediction [1]. The challenge with logs is that most part of the logs is not interesting and does not provides useful insight. It is only a fraction of the logs that are important.

Also, with the growing complexity of the infrastructure and systems, it is not easy to test and debug distributed systems in case of failures. Machine learning applies advanced statistical techniques where machines can learn on their own and make sense of the given data to identify patterns and make predictions. “Data is the new oil” [2] [3] and ML techniques are used to create the value out of the data [4]. The Machine Learning is subset of the Artificial Intelligence field, which aims to impersonate the power of human brain [5].

Machine Learning is broadly classified into two categories:

Unsupervised learning is used to uncover the concealed patterns inside the data. It is basically used when no prior facts(labels) about the data are available and you are trying to make sense of the data in some manner. It is used find the dataset underlying structure and find various groups. It is usually applied to understand complex, highly non-linear models with lots of parameters on unlabeled data [6]. Clustering is one of common used technique that falls under the unsupervised category [7]. In contrast to supervised learning it is not easy to evaluate the performance of unsupervised learning and is very domain specific [8].

Supervised Learning is applied in case where the data is already labeled. It focuses on building a mapping function from input variables to output which can be used to predict on new data set. On an abstract level it can be summarized as building a function f(X) which can map an input to output [9].

Fuzz testing plays a vital role in software security. Fuzz testing mainly falls under the category of black box testing [73]. In 1989 Barton Miller at the university of Wisconsin developed FUZZ testing [74]. This quality assurance test technique involves feeding random input data like invalid characters, unexpected data size and then check or monitor the behavior of the system under test like memory leaks, code assertion, program crash [73]. A software tool fuzzer is used to detect the memory leaks, built-in code assertions. The software tool fuzzer is used to analyze vulnerabilities that can be exploited by denial of service attack and buffer overflow. Fuzz testing mainly helps to discover the most significant defects, or we can say loopholes which can be dangerous for the security of the software [75].
To summarize, to make the meaning out of huge volume of data is not easy and ML techniques have proved very useful and growing as a very powerful methodology. It can be used together with Fuzz testing to improve the quality of testing.

1.1 Background

The thesis project deals with data analysis from test cases logs. To analyze log data both supervised and unsupervised Machine Learning techniques are used. Unsupervised Learning algorithms DBSCAN and HDBSCAN are used for cluster analysis [10]. Supervised Learning algorithms SVM [11] and RF [12] are used to provide more accurate data analysis and prediction [13].

Cluster analysis is very popular unsupervised learning technique used to group similar objects [14]. In other words, organizing unlabeled data into similarity groups is called clustering. The main motive of cluster analysis is to find the structure of the given unlabeled data set [15] and organize large quantity of unordered text documents into a small number of meaningful, coherent clusters. To perform cluster analysis a variety of distance functions and similarity measures can be used. Few examples of similarity measure are Levenshtein distance, Euclidean distance, cosine similarity [16].

In the thesis work Levenshtein distance function is used to calculate similarity score between items and later fed on various clustering algorithms for cluster analysis.

Text classification is one of the most prominent application of Machine Learning which is mainly used to automatically assign predefined labels to free text documents. The main motive behind the text classification is to give conceptual organization to a large collection of documents [17].

For the thesis project run time benchmarking with classifiers SVC and RF is carried out.

1.2 Problem

Fuzz testing is automation technique to induce invalid random data to software systems. It can cause the system to be in an invalid state from where it cannot recover and sometimes can even lead to system crashes [18]. In today’s cyber world security in software systems is a very important aspect. Fuzz testing is one of the most popular technique to test the security of the system in telecom domain. One of the challenges is one cannot always fully decipher the test results since there are many if and buts in a big and complex system. Also, the logs produced by fuzz testing are quite huge and results are large datasets which are unstructured. The logs are complex, and it is not possible to analyze the logs manually especially in large systems.
“How to analyze the fuzzer test case logs and based on the analysis how can we select the most promising test cases?

1.3 Purpose
The purpose of this thesis project is to analyze the test logs and design the alternative strategies to provide the feedback in form of score per test case so that they can be used to improve the existing radio protocol fuzzer.

1.4 Goal
The goal of this thesis includes both an element of theory and hands-on implementation to develop a system capable of doing logs analysis and returning a feedback by applying machine learning techniques. Theoretical focus is on the design of alternative strategies to guide the fuzzer in selecting the test cases based on feedback and experience. We apply various machine learning algorithms to analyze the test case logs data. Detailed literature study is carried out to choose the most suitable algorithm and apply that algorithm to test logs to create feedback in near real time.

1.4.1 Benefits, Ethics and Sustainability
The thesis work deals with log analysis using machine learning and based on that analysis can greatly improve the effectiveness of fuzz testing. Machine learning is about making sense of data pretty much in similar way as humans, the difference being the large scale of data which is not possible in traditional ways. It is a kind of artificial intelligence where algorithms determine patterns in data.

Machine learning has challenges in terms of dealing with high dimensional data, poor data quality(noise) and sometimes with limited information available there is lot of uncertainty. It lies somewhere between statistical theory and practically noise data. Any model developed usually needs lot of iterations, A/B testing so that results produced have the desired accuracy. Also, the data being analyzed may acquire new attributes which might lead to revisiting the model again. Identifying useful attributes in high dimensional data also needs good collaboration between domain and technical experts.

1.5 Methodology / Methods
Research for the thesis project deals with both qualitative and quantitative methods [19]. Qualitative analysis is used to investigate the behavior of the log data.
The research project includes both elementary studies and hands-on implementation. Literature study has been conducted to understand all the concepts related to data analysis and tools, which are used to design the proposed system.

All the tasks and milestones have been planned for the proposed system.

### 1.6 Delimitations

One of the most challenging aspect of development in Machine Learning is to finalize the most suitable learning algorithm for the given data set. The main aspect of this study is to do data analysis using machine learning techniques and design the alternative strategies to provide the feedback that can help to the fuzzer to choose the test cases.

This study mainly carries out experiments using different Machine Learning algorithms and chooses the most suitable algorithm for the given dataset to finally build a prototype. The study yearns to do some benchmarking to compare run time of different distance functions and some benchmarking to compare the run time of different classifiers also.

The study is conducted with the limited amount of static dataset.

### 1.7 Outline

The outline of the thesis project is described in the following sections:

- Section 2 covers literature study of similarity score using distance functions, various implementations of Levenshtein distance function, Machine Learning techniques including clustering algorithms, classifiers and related work.
- End to end application architecture, design and programming tools used in the thesis project have been described in section 3.
- All the experiments and research and methodologies used are described in section 4.
- Evaluation of the work have been described in section 5.
- Future work is described in section 6.
- Section 7 is allocated for conclusion.
2 Theoretic Background

This section of thesis report presents in-depth literature study and background of the Supervised and Unsupervised Machine Learning algorithms for e.g. SVC [11], RANDOM FOREST [11], DBSCAN [20] and HDBSCAN [10] with various implementations of Levenshtein distance function [24] like Python-Levenshtein [21], editdistance [22] and Leven [23].

Following are the key requirements for the system to be developed:

- Extract the input datasets of different types from the logs produced by the running fuzzer test cases. The log file contains results of each test case. Every test case result further contains output in the form of different data types such as errors, counters and alarms.
- Measure the similarities and dissimilarities between the different data types and transform it into distance matrix.
- Perform clustering for different data types.
- Calculate a score per testcase.
- Send the score as feedback to client.

Our research progresses with following milestones:

- Data preprocessing.
- Analyzing various similarity measures (distance function).
- Experimenting with unsupervised learning algorithms and choose most optimal clustering algorithm for the data set.
- Experimenting with supervised learning based on the output of unsupervised learning algorithm. This step can be considered as Semi Supervised learning.

2.1 Similarity Measures

This is the first step of the process. Similarity determines how similar are two objects. This similarity score is used in machine learning techniques such as clustering, recommendation engines, anomaly detection and classification. In Natural language processing we are interested in finding similarity between different sentences or different documents [26].

It is important to note that the value of similarity depends on the context and domain of the application. Following are widely used similarity measures for text data type:

1) Cosine Similarity: It is a geometrical distance and also called as cosine distance. It measures the cosine of the angle between two vectors [25]. If the two vectors are same, then \( \cos (0) = 1 \), which means two objects are similar. It is used to calculate similarity between text documents.
2) Jaccard distance: This is primarily used to quickly determine how similar text is by counting the frequencies of letters in a string and then counting the characters that are not the same across both [26].

\[
\text{Jaccard Similarity} = \frac{\text{Intersection of A and B}}{\text{Union of A and B}}
\]

3) Levenshtein distance: Another similarity measure that is commonly used in natural language processing is the Levenshtein distance. The value refers to the minimum number of actions (deletion, insertion and substitution) required to transform from one word to another word [27]. This distance algorithm is very commonly used in natural language processing.

For the thesis project, we focus on Levenshtein distance. Levenshtein distance is the most commonly used distance function to measure the similarity between two strings. Levenshtein distance supports substitution as well as deletion and insertion at character level in the string [27]. Our input dataset contains strings of varying size. For example, data type counter contains one string for the name and one string for the value.

Distance functions serve as necessary base for clustering and classification problems [24]. As part of our work, we have conducted run time benchmarks of different modules (implementations) of levenshtein distance function.

In mathematical terms, it can be described as mentioned in Wikipedia

Mathematically, the Levenshtein distance between two strings \(a, b\) (of length \(|a|\) and \(|b|\) respectively) is given by \(\text{lev}_{a,b}(|a|, |b|)\) where

\[
\text{lev}_{a,b}(i, j) = \begin{cases} 
\max(i, j) & \text{if } \min(i, j) = 0, \\
\min \left\{ \text{lev}_{a,b}(i - 1, j) + 1, \text{lev}_{a,b}(i, j - 1) + 1, \text{lev}_{a,b}(i - 1, j - 1) + 1_{(a_i \neq b_j)} \right\} & \text{otherwise.}
\end{cases}
\]

where \(1_{(a_i \neq b_j)}\) is the indicator function equal to 0 when \(a_i = b_j\) and equal to 1 otherwise, and \(\text{lev}_{a,b}(i, j)\) is the distance between the first \(i\) characters of \(a\) and the first \(j\) characters of \(b\).

Note that the first element in the minimum corresponds to deletion (from \(a\) to \(b\)), the second to insertion and the third to match or mismatch, depending on whether the respective symbols are the same.

**Figure 1. Levenshtein distance function (source: Wikipedia)**
https://wikimedia.org/api/rest_v1/media/math/render/svg/4520f5376b54613a5b0e6c6db46083989f601821
The formula can be applied, and computation steps can be demonstrated using matrix. For Example, we can find the distance between strings “JYOTI” and “JKOBI” as demonstrated in the following matrix.

<table>
<thead>
<tr>
<th></th>
<th>J</th>
<th>Y</th>
<th>O</th>
<th>T</th>
<th>I</th>
</tr>
</thead>
<tbody>
<tr>
<td>J</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>K</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>O</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>I</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

**Table 1. Levenshtein distance example**

The distance between two strings is the lower right corner of the matrix i.e. 2. This means that “JYOTI” can be transformed to “JKOBI” by substituting “K” for “O” and a substituting “B” for “T” (2 substitutions)

We mainly experimented with following Levenshtein distance implementations:

1) Python Levenshtein module: Levenshtein (Levenshtein.distance) [28]

   Python C extension module contains functions for fast computation of:
   
   - Levenshtein (edit) distance, and edit operations
   - string similarity
   - approximate median strings, and generally string averaging
   - string sequence and set similarity

   It supports both normal and Unicode strings.

2) Python editdistance module: editdistance (editdistance.eval) [29] is another python package that we will be used in our benchmarking process.

3) stringdist.levenshtein [30]: stringdist.levenshtein is another python package which is also used to calculate distance between two strings.
2.2 Machine Learning Techniques

Arthur Samuel coined the term machine learning as "Field of study that gives computers the ability to learn without being explicitly programmed" [32].

Machine learning involves algorithms that can learn from and make predictions on data. They differ from static program instructions and operate by first building a model from an existing training set of input observations and then making predictions for new datasets [32].

Machine learning techniques can be divided into three major categories:
- Supervised learning
- Unsupervised learning
- Reinforcement learning

Supervised learning involves fitting a data to a function or function approximation. It is used when existing dataset is labelled (called training dataset) and needs to be predicted for new datasets. A supervised learning algorithm analyzes the training data and produces an inferred function, which can be used for mapping new examples [33].

Unsupervised learning is about figuring out what is special with the data. It is used to infer patterns in the data set when there are no existing labels on the datasets. Unsupervised learning mainly focuses on the grouping of similar objects to produce a cluster. It is used for discovery of patterns in the data and uncover unknown patterns in the dataset. In unsupervised learning it is hard to determine accuracy of the result produced which makes supervised learning popular in real world problems. Common applications of unsupervised machine learning are following:
- Clustering: to automatically split the dataset into various groups based on similarity.
- Anomaly detection: to discover unusual data points in the dataset. Fraud detection is one such use example.

Reinforcement learning is an important type of Machine Learning where an agent learns how to behave in an environment by performing actions and seeing the results [34]. This is out of scope for this thesis project.

2.2.1 Clustering: An Introduction

Clustering can be thought of dividing the data points into several groups in such a way that data points which fall in the same group have similar characteristics
to other data points in the same group as compared to those in other groups. The efficiency of the cluster depends on how similar the objects are within a group and how different objects are across groups [35]. It is one of the most important analytical methods and used in data mining [36]. There are various clustering algorithms which differ from each other in terms of input parameters they accept, their interpretation of formed cluster and how efficiently they can create clusters. To get the desired results from clustering it is important to modify the data pre-processing and tune the various parameters. Cluster analysis does not provide out of the box solution and is not an easy task. It usually spans many iterations for knowledge discovery before a useful result can be achieved [37].

2.2.2 Types of Clustering

Clustering can be majorly divided into following subgroups:

2.2.2.1 Hard Clustering
Hard clustering is also known as Exclusive clustering and the clusters do not overlap. An element either belongs to a cluster or not [35].

2.2.2.2 Soft Clustering
In soft clustering clusters may overlap and it is possible that a single element falls into more than one group. There is a degree of belonging i.e. probability in the sense that how much an element belongs to different clusters [38].

2.2.3 Clustering Algorithms
A deep literature study of the clustering algorithms (like DBSCAN, HDBSCAN) and classifiers (like SVC, RF) used in the thesis project has been conducted to select the most promising algorithm for the datasets.

2.2.3.1 DBSCAN
DBSCAN is also called ‘Density-Based Spatial Clustering of Application with Noise’. Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu proposed DBSCAN in 1996 to detect clusters in spatial dataset which contain noise [39]. It groups the points which are close to each other based on a distance function and the minimum number of points required to form a group [41]. DBSCAN algorithm automatically creates a number of clusters. It does not produce complete clustering in the sense that the points which lie in the low-density regions are considered as noise points and thus omitted [42].

Parameter selection: DBSCAN algorithm mainly needs two parameters, epsilon and minPoints. epsilon specifies how close points should be to each other to be considered neighbors. minPoints specifies how many neighbors a point should have in order to be included in a cluster [43].
Figure 2. Clustering with DBSCAN
Source: https://cdn-images-1.medium.com/max/1250/1*zbm_3K647rvNDmgL6HWUNQ.png

On abstract level DBSCAN begins by choosing a random point in the given data set and determines how many other points are nearby to that random point based on eps value. It continues this process until it cannot find data points nearby, and then it will start building the next cluster. The non-core points which do not lie in any cluster are considered as noise points [40].

Advantages:
- Main advantage of DBSCAN algorithm is that it works well with outliers. Outliers are the points that exist in low density regions [41].
- DBSCAN does not require to specify the number of clusters in advance like K-means.
- DBSCAN can find arbitrary shaped and non-linear clusters.
- DBSCAN is quite good in separating high density regions from low density regions for a given dataset.

Disadvantages:
- It is quite hard to choose the appropriate combination of eps and minPoints parameter values.
- DBSCAN does not perform well on datasets with large variation in densities.
2.2.3.2 HDBSCAN
Hierarchical Density-Based Spatial Clustering of Applications with Noise. A hierarchical, density-based clustering algorithm that extends the previous density-based algorithms DBSCAN [44] and converts the DBSCAN into a hierarchical clustering algorithm. HDBSCAN starts off much in the same way as DBSCAN. It also first transforms the space according to density and performs single linkage clustering on the transformed space [45]. Advantage of HDBSCAN is that it performs DBSCAN over varying epsilon values and takes care of integrating the result to find a clustering that gives the best stability over epsilon. This makes HDBSCAN to identify clusters of varying densities and be more robust to parameter selection [46].

In the thesis project, HDBSCAN is finally chosen as a clustering algorithm of choice.

Parameter Selection: Although HDBSCAN has large number of parameters but following parameters cause notable impact on clustering:

- **min_cluster_size**: This determines the smallest grouping size which can be considered as cluster. It is used when splitting large clusters. Increasing the value of this parameter will reduce the number of clusters formed and decreasing the value will increase the number of clusters.

- **min_samples**: min_sample has a significant role on clustering. It determines minimum number of points needed to form a cluster and used for forming clusters. The greater value of min_samples will lead to more points being declared as noise points.

2.3 Cluster visualization using TSNE
Laurens van der Maatens and Hinton in 2008 [47] developed t-SNE which is basically a unsupervised non-linear technique which proves to be very useful in case of data exploration and visualization of high-dimensional dataset [48]. Visualization of high-dimensional data in a meaningful way is a critical problem in almost every domain and it becomes more crucial when we are dealing with varying dimensionality dataset [49]. t-SNE (t-distributed Stochastic Neighbor Embedding) technique is a way of transforming a high-dimensional data set into a matrix of pairwise similarities and visualize the results in a way that are easy to interpret by humans. t-SNE mainly start its working by calculating the similarity measure between pair of objects in the high dimensional region verses pair of objects in low dimensional region. After that t-SNE starts optimizing the similarity measures with the help of cost function [48]. Visualization of high-dimensional data is done by giving each object a location in a two or three-dimensional space [49].
The main objective is t-SNE is to take a set of instances in a high-dimensional space and assign a dedicated characterization to them in a lower-dimensional space which can be in 2D or 3D [49]. Some of the use cases of t-SNE that Laurens van der Maaten mentioned in different areas are like breast cancer research, climate research, bioinformatics, security area in IT, medicine.

Parameter Selection:

t-SNE contains a tunable parameter called perplexity which determines how to stabilize the observation between local and global characteristic of the given data. The parameter describes the approximation about the number of close neighbors each point has. The value of the parameter perplexity value has significant impact on the visualization of the resulting data.

The stability or we can say accuracy of the visible clusters identified by the t-SNE algorithm can be endorsed by analyzing the clusters by using a range of perplexities. Recommended value for perplexity range between 5-50 [50].

Dataset for the project is in the form of long strings that contains pairwise similarities between two strings instead of using high-dimensional vector representation of each object.

2.4 Classification

Classification falls under the supervised machine learning category in which a computer program learns from the data input given to it and then apply this knowledge to classify the new observations. We can relate classification as prediction for new observations. Classification is the process of assigning the classes or categories to the given data points. The goal to classify the dataset is to predict the class of the data points. Classification algorithms are about approximating a mapping function(f) from input variables(x) to discrete output variables(y) [51]. Classification algorithms usually requires lots of labelled data and designing good features is a tedious job, although they provide automatic method to discover hypothesis explaining the data [52].
Figure 3. Steps involved in supervised ML

2.4.1 classification algorithms

Following are the common classification algorithms [76]

1. Linear Classifiers (Logistic Regression, Naïve Bayes Classifier)
2. Support Vector Classification
3. Boosted Trees
4. Random forest
5. Neural Networks
6. Nearest Neighbor
7. Decision Trees

Every algorithm has its own advantages and disadvantages and to summarize no single algorithm works best for every problem. In the thesis project, the focus is on experimenting using Random Forest and Support Vector Classification algorithms.

2.4.1.1 Random forest
Random forest classifier is an ensemble algorithm. Ensemble learning involves the use of multiple learners and combining their results. A machine learning ensemble is a collection of models. The models in the ensemble can be based on different techniques, trained on different data sets, can be based different
features and different values of parameters. A random forest is an ensemble of decision trees. It belongs to the category of supervised learning algorithms. Random forest classifier creates a set of decision trees from randomly selected subsets of its training set. To decide the final class for the object it combines votes from various decision trees [53].

Random forest algorithm can be used for both classification and regression. Since it considers the average of all predictions thus cancelling biases, it overcomes the problem of overfitting. Due to the number of trees involved in the decision process it provides accurate and robust results [55]. At the same time due to the multiple decision trees involved, generating prediction values in random forest will take more time compared to a single decision tree.

2.4.1.2 SVM (Support vector machine)
SVM is also a supervised machine learning algorithm. SVM is used to build a binary classifier. It makes the classification decision based on the linear function of a points coordinates and does not require prior knowledge of probability distribution of points. SVM finds the (n-1) dimensional hyperplane in an n dimensional space to separate objects into two categories. A hyperplane can be described as a geometrical shape with (n-1) dimensions and zero thickness in one dimension in a vector space of n dimensions.

SVM differs from the other classification algorithms in a way that it chooses the decision boundary that maximizes the distance from the nearest data points of all the classes. SVM is known for finding the most optimal decision boundary called hyperplane. The best hyperplane is the one which maximizes the sum of distances of the nearest points on either side of the hyperplane. The nearest points from the decision boundary that maximize the distance between the decision boundary and the points are called support vectors [56].
3 Web Framework Architecture Design

This section describes the architecture of the complete web application (score-fuzz-feedback). One of the considerations when designing the application was to integrate the score-fuzz-feedback application with the output of existing fuzzer application so it is decided to use REST API for integration with the existing system.

REST APIs are quite flexible and have the following characteristics [57][58]:

- **Client-Server**: Client and Server are separate from each other and can evolve independently.
- **Stateless**: Each request contains all the necessary payload required by the server which means that the server doesn’t need to keep track of previous requests.
- **Uniform Interface**: means the client is decoupled from server application implementation. REST defines standards which are used to achieve contract between client and server. This helps independent evolution of the application business logic without being coupled to the API layer.

Rest APIs have the advantage over SOAP since they are not limited to XML but can support any format such as XML, JSON, YAML or any other format. Also, they have advantage over RPC that clients aren’t required to know the procedure names or parameters in specific order.

Figure 4. High level Application Overview
The above figure provides a high-level design overview of web application. The application is deployed in the flask server where the client interacts with it using HTTP request and response. Flask Framework [70] is chosen to develop the web application. There are dozens of tools and frameworks available for writing the web application, but flask was chosen due to following reasons:

- We have application’s clustering/machine learning module written in python, so we are looking at frameworks which could integrate and support python programming language, flask being one of them.
- Flask is a micro web framework written in python which doesn’t require particular tools and libraries [69].
- Flask is very lightweight framework which helps you in building working web application in a short amount of time from the very ground up. This suited us well since the focus of the application was more on clustering/machine learning module and we are looking to integrate clustering module in the web application as easily as possible. It is very easy to get started and does not require much time for the web application.

When coming to the application storage/persistence part we are looking for some document-based database as storage engine. The reason being the data that we want to store is in http request payload data (i.e. testcase log entry). Testcase data is of JSON format and it could be considered as one document. Although there are plenty of document-based database available, we decided to use MongoDB [71]. It is free and cross platform document-oriented database. Our main reasons to choose MongoDB are following:

- **Scaling**: MongoDB support horizontal partitioning and it is to scale.
- **Installation**: environment setup and installation are quite easy in case of mongoDB.
- **High Availability**: MongoDB provides high availability with replica sets of data.
- **Schema less**: One of the primary features of mongoDB is that it is schema free which makes it more flexible.
- **Deep Query ability**: MongoDB provides powerful dynamic query facility taking on documents using document-based query language [72].
3.1 Workflow Design

This section provides detailed understanding of the various steps involved in analyzing, modelling and development of the system.

![Workflow Diagram](image)

**Figure 5. High level overview diagram of system**

Above diagram shows a very high-level overview of the system. eNB is the radio base station used in Ericsson. GTE (Generic Test Environment) framework is an Ericsson internal test tool used for eNB. GTE is used to support testing of eNB. The diagram shows that the web application receives input data in form of test logs from the fuzzer application. It then calculates the score for the input data by applying HDSBCAN clustering algorithm and passing the cluster results to the scoring algorithm. The score is then returned as response to the fuzzer. The algorithm and scoring functionality are explained in detail in further sections.
4 Framework Development

This section describes various development tasks needed to develop the score fuzz feedback application. This section mainly focuses on steps involved from processing of input data to generating the final score for the input data.

4.1 Milestone M1

We set up following objectives to be achieved as part of our first milestone:

- Convert data to a format usable in python.
- For each of the data type extract and build samples for clustering from the converted data.
- Compute pairwise distance matrices for the data types and in doing so benchmark the 3 different python modules of Levenshtein distance.
- Experiment with string clustering using DBSCAN (using metric='precomputed' an above derived matrices).
- Data Visualization
4.2 Step-wise process of development of learning model for Milestone M1:

![Steps overview diagram]

Figure 6. Step-wise process of development of learning model

The above diagram describes the step-wise working process of the leaning model in details.
Following are the steps executed during the development of the learning model:

4.2.1 Data pre-processing
Logs appeared to be JSON objects but during processing we found that they were not valid JSON string type. We stringify it by using nodejs library. It might sound strange why we had to convert the text data to JSON since there are various for machine learning techniques which can work on text data. It was needed in our case since the log data itself could be sub grouped in different types and we needed to apply machine learning techniques to each specific type in test logs.

For e.g. it is not a good idea to apply clustering algorithm over two types of data A and data B on overall data together since each type also has its own specific attributes and values.

The system applies clustering algorithm to data type A to create various clusters where are separate from clusters created for data type B. To summarize we can say that data was already in subcategories where we had to apply ML technique to each subcategory separately since these subcategories were not related together. The output of pre-processing yields JSON parseable data which is fed to feature extraction stage.

4.2.2 Feature extraction
For each type in input data, system extracts respective specific features for each input type. The extracted features are used in building distance matrix for each type using various distance functions described above in chapter 2.

4.2.3 Distance matrix
For each type, the system creates a distance matrix based on features extracted above. The idea behind distance matrix is to cluster data based on how similar these strings are.
System applies different Levenshtein distance function implementations to different data types. Based on benchmark tests the most efficient implementation of Levenshtein distance function for that type is chosen.

4.2.3.1 Benchmarking various implementations of Levenshtein distance:
Evaluating the performance of various implementations of Levenshtein distance function was important since we wanted to choose the best performing implementation on our data. There are various modules available for Levenshtein distance metric like leven, editdistance and Levenshtein.
The final benchmarking is carried out with two implementations:

- First one is our own loop-based function (pw_dist) on each of the Levenshtein module mentioned above
- Second one is carried out using scipy library inbuilt function (pdist).

The benchmarking was carried out by running more than one execution for each distance function and then take average run time for benchmarking for each function and data type and then plotting bar charts for benchmarks.

Given the widely varying size of the different data types, and the resulting significant difference in run-times, we decided to keep the number of total items per data type constant. So, instead of cycling through each dataset the same number of times, we'll determine the number of repetitions taking into account the size of the dataset using

\[ R = \frac{C}{|D|} \]

where R is the number of repetitions
C is our fixed constant e.g. 10000
|D| represents the size of the dataset

The following code snippet shows how the benchmarking for various implementations of distance function was carried out. Both these functions accept input dataset (for which the distance matrix is supposed to be calculated) and the distance function implementation as arguments.
Figure 7. Code example for benchmarking various Levenshtein modules

4.2.3.2 Distance function benchmarking conclusion:

Following are the findings as shown in the Figure 8.

- For datatype errors editdistance package is performing better than all other alternatives leven, Levenshtein, stringdist.
- For counters leven is performing better.
- For alarms data type performance of Levenshtein module is better.

Data type “errors” contains more varied strings. It does appear that editdistance does work better on more varied strings but not so well on shorter or more homogenous strings in data type counters, sanity and alarms.
Figure 8. Runtimes for various Levenshtein distance function implementations

4.2.4 Applying Clustering algorithms

The system feeds the distance matrix into one of the clustering algorithms to create various clusters for each type. After a detailed literature study on various unsupervised machine learning algorithms it is concluded that DBSCAN and HDBSCAN are the most suitable algorithms for the sample dataset in our case. The sample dataset contains the data in form of variable strings.
4.2.4.1 DBSCAN

DBSCAN is density-based clustering algorithm. It is based on connected regions with sufficiently high density [59]. It is known to produce good results for non-linear structure based on density [60] and can discover clusters with arbitrary shape which is not possible with other clustering algorithms for e.g. K-means. For DBSCAN there is no need to define the number of clusters in advance but for other clustering algorithm we need to define number of clusters in advance.

```python
# construct a larger matrix for largest data set (i.e. errors)
import numpy as np

matrix = pw_dist(samples['errors'], impls['editdistance'])
tiled = np.tile(matrix, (10, 10))
tiled.shape

from sklearn.cluster import dbscan
deltas = []
steps = range(100, 4700, 100)
for m in steps:
    m_by_m = tiled[:m, :m]
    start = time.time()
    runs = 0
    while runs < 5:
        core_points, labels = dbscan(m_by_m, metric="precomputed", eps=7.5, min_samples=3)
        runs += 1
        deltas += [(time.time() - start)/runs]

plt.clf()
plt.plot(steps, deltas)
plt.show()
```

**Figure 9. code example of DBSCAN algorithm**

DBSCAN algorithm is accepting the following parameter as input [41]:

- `m_by_m` is the distance matrix which we created by using distance function.
- `metric` has value “precomputed” which is used when distance matrix is calculated in advance.
- `eps=7.5` indicates the maximum between two samples to be considered in the same cluster.
- **min_samples=3**: It indicates the minimum number of data points for a point to be considered as a core point.

**Table 1. Sample output of DBSCAN Algorithm**

```
<table>
<thead>
<tr>
<th>core_points, labels</th>
</tr>
</thead>
</table>
| ([ 0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 79, 80, 81, 82, 83, 84, 85, 86, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 99], dtype=int64),
| ([ 0, 0, 1, 0, 1, 0, 0, 2, 1, 2, 0, 3, 0, 0, 2, 0, 3, 0, 1, 2, 0, 1, 1, 0, 0, -1, 1, 0, 0, 1, 1, -1, 0, 3, 0, 0, 1, 0, 0, 1, 1, 0, 0, 2, 2, 1, 2, 2, 0, 0, 3, 0, 0, 0, -1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 2, 0, 1, 1, 0, 0, 2, 1, 3, 2, 0, 1, -1, 1, 0, 1, 1, 0, 1, -1, 0],
| dtype=int64))
```

**Figure 10. Sample output of DBSCAN Algorithm**

The above figure describes the sample output of the DBSCAN algorithm. DBSCAN produce the output in the form of core points and labels. The points which exist in the low-density region are known as outliers and have label -1. For e.g. data point 28 has label -1 which means it is a noise point(outlier) and doesn't belong to any cluster. [61].
4.2.4.2 DBSCAN cluster Visualization using t-SNE

t-SNE is a technique used to reduce the high dimensionality of the data into two dimensional or three dimensional and helps to visualize and interpret the high dimensional clustered data.

```python
# Plotting function (for db scan cluster results)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from matplotlib.ticker import NullFormatter
import time

def scatter_plot(ax, X, y, perplexity):
    ax.set_title(f'Perplexity-{perplexity} % perplexity')
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.get_cmap('tab20'))
    ax.xaxis.set_major_formatter(NullFormatter())
    ax.yaxis.set_major_formatter(NullFormatter())
    ax.axis('tight')

def plot_TSNE(X, y):
    perplexities = [5, 10, 30, 50]

    plt.clf()
    fig, subplots = plt.subplots(ncols=len(perplexities), figsize=(15, 8))

    for i, perplexity in enumerate(perplexities):
        t0 = time.time()
        tsne = TSNE(n_components=2, metric='precomputed', init='random', random_state=0, perplexity=perplexity)
        Y = tsne.fit_transform(X)
        t1 = time.time()
        scatter_plot(subplots[i], Y, y, perplexity)
        print(f'Perplexity-{perplexity} in %.2g sec' % (perplexity, t1 - t0))

    plt.show()
```

**Figure 11. Sample code for visualization using t-sne**

To plot the DBSCAN output (cluster data) we have chosen the following parameters [62]:

- n_components (chosen the default value). Parameter n_components describes the dimensionality of the embedded space.
**Metric** (chosen precomputed) T-sne takes the precomputed distance matrix as an input.

**Perplexity** (chosen range between 5 to 50). The feature perplexity plays a vital role in the visualization of the clustered data. It describes the number of close neighbours for a data point. According to the original paper “The performance of the SNE is fairly robust to changes in the perplexity”.

![Cluster visualization with different perplexity values](image)

**Figure 12. Cluster visualization with a range of perplexity**

The above figure clearly shows the different shapes of clusters with respect to the different values of perplexity. With the perplexity range 0 and 5 it is very complex, or we can say impossible to interpret the results. Perplexity value 30 shows the results in a better way. It shows the visibility of merging of clusters. But with the perplexity value 50 it is easy to see the well merged cluster.
4.2.4.3 Challenges with DBSCAN

The DBSCAN relies mainly on parameters eps and minPoints. Eps is minimum distance between two points to considered them as neighbors. minPoints is the minimum number of points to form a cluster. Finding the right value of above two parameters is a big challenge and it is important to have the good understanding of the dataset.

Eps: For a chosen very small eps value will create large number of outliers and data will not be clustered. This is due to the reason that minimum number of data points in the dense region(cluster) will not be satisfied. On the other hand, if the eps value is big, everything will fall into the same cluster. As a rule, small (not very small) eps values should be considered.

minPoints: minPoints value depend on data set dimensionality. To get the desired result it is very important to know the dimensionality of your data set. For data sets with noise it is recommended to have large value of minPoints. The general rule is to have minimum value as 3 [41].

Due to these challenges we decided to move to HDBSCAN which is extended form of extended form of the density-based algorithm DBSCAN. HDBSCAN does obviate the need for fiddling with 'eps'. Another thought to move to HDBSCAN was to use its 'fit_predict api' for prediction.

4.2.4.4 HDBSCAN

HDBSCAN is a hierarchical clustering algorithm. The main advantage of using HDBSCAN over DBSCAN is that it produces good results with data of varying density in form of cluster and performs better as compared to DBSCAN [45]. Most useful and tunable parameter which is used in HDBSCAN is min_cluster_size. In our case we have chosen minimum cluster size 5 which describes the minimum size grouping to be considered as a cluster.

Below is the example usage of HDBSCAN.

```python
if 0 < len(self.items):
    # update distance matrix
    dists = np.array(list(map(lambda item: self.metric(sample, item), self.items)))
    self.dists = np.insert(self.dists, 0, dists, axis=0)
    self.dists = np.insert(self.dists, 0, np.insert(dists, 0, 0), axis=1)

    # cluster samples
    cltrr = hdbscan.HDBSCAN(metric='precomputed', min_cluster_size=5).fit(self.dists)
```

**Figure 13.** HDBSCAN example code

As mentioned in the above code snippet we used HDBSCAN ‘fit’ api for clustering from distance matrix.
Initially the aim was to use ‘fit_predict api’ for prediction of new datasets. Unfortunately, we found that HDBSCAN ‘fit_predict api’ does not support prediction if the clusterer is trained with ‘metric=precomputed’ [63]. Since we pass distance metric as ‘precomputed’ it did not fit our needs.

4.3 Evaluation of Milestone M1

We found that DBSCAN doesn’t build a model and cannot be used to classify/predict new data sets [64]. It is because the new data sets can alter the cluster and can cause the clusters to merge. This means that for new data sets we will have to do clustering again for the entire dataset (old and new) again. Redoing clustering for every new data sets would be too expensive for our use case but the labels from the cluster could be used to build a classifier, since we are not interested in rediscovering the structure created by DBSCAN. HDBSCAN looked promising to build a model and predict new data sets but as mentioned above we cannot use its predict api for our case.

4.4 Milestone M2: Combining the results (another level of clustering)

As mentioned in the above section 4.3, redoing clustering every time would be too much expensive, but the labels could be used to build a classifier instead. Using unsupervised learning to build a cluster and using labels from the cluster as an extra feature or input to some supervised learning model is called semi-supervised learning [65]. The following tasks are carried out to combine results:

- Build classifier (SVC, Random forest)
- Benchmarking performance of classification algorithms

4.4.1 Experiments with Classifiers:

Benchmarking is a very crucial and important way to evaluate the algorithms based on their accuracy, performance on run time [66]. In the thesis project, we have performed run time benchmarking with the classifiers (SVC, RF) to check their performance, accuracy on run time with our dataset.

Random forest has some advantage over other classifiers which makes it more suitable for our dataset. For e.g. it does not overfit, fast, produce good results with large data sets [67]. SVC (support vector classifier) is another classifier which we have chosen for the run time benchmarking for our data set. SVC is a widely used algorithm for non-linear classification because of its high accuracy rate and its robustness [68].

Our experiment included the following:

- Try SVC and RF and compare the results
- Experiment with Ignoring and Including samples marked as noise (labels = -1)
Below is the code usage of the classifiers (SVM, RF). We ran comparison benchmarks for SVC and RF in terms of time taken for execution. We performed it by increasing the data size in steps for each classifier. For each classifier and each step (data size) we run it multiple number of times and then take the average.

```python
# benchmark run times for different(SVC and RF) classifiers
import numpy as np
import matplotlib.pyplot as plt
import time

colors = ['r', 'g']
clfs = ['SVC', 'RF']
tiled_features = np.tile(features, (10, 10))
tiled_labels = np.tile(labels, 10)

steps = range(100, 4700, 100)
total_runs = 3

data = 
for i, classifier in enumerate(clfs):
times = []
    for m in steps:
        feature = tiled_features[:,m]
        label = tiled_labels[:,m]
        start = time.time()

        runs = 0
        while runs < total_runs:
            classifiers[classifier].fit(feature, label)
            runs += 1

        delta = (time.time() - start)/runs
        times += [delta]
        print(classifier, m, delta)

    plt.plot(steps, times, colors[i], label=classifier)

plt.legend()
plt.show()
```

Figure 14. Sample code for evaluating classifiers
Figure 15. Run time benchmarking of (SVC, RF) classifiers

From the above graph we can see that RF is performing better than SVC. RF performs better when data size grows.

Above graph clearly shows the runtime performance of the classifiers (RF, SVM). The accuracy of both were nearly same.

Since RF had the best execution time, we used RF as a classifier to train our model and use it for prediction on new data sets. For this, we needed an embedding multidimensional scaling (MDS) to apply on our high dimensional distance matrix. To reduce the dimension, we used manifold.MDS embedding. The embedding results were fed to classifier to train the classifier and later used for prediction. The code snippet mentioned below shows the sample usage.

```python
def embed(self, distance_matrix):
    embedder = manifold.MDS(n_components=9, dissimilarity="precomputed")
    coords = embedder.fit_transform(distance_matrix)
    return coords
```

Figure 16. Sample code for embedding
Figure 17. Sample code for using a Random Forest classifier for training and prediction

Classifier Conclusion:
The above classifier approach did not go well since we realized that we will have to re-train the model periodically. We will also have to cluster first and only switch to classification after a certain size, if at all. The latter depended on the time efficiency of clustering vs embedding.
4.5 Milestone M3: provide testcase feedback based on cluster size

We decided not to proceed with classifier at all and thought if we could always use clustering instead of prediction, every time a new test case data is fed to the system.

The rationale behind this is if we could build an incremental version, we could just keep doing clustering, assuming that the incremental operation is sufficiently cheap and comparable to the effort involved in classification, instead of building a classifier.

The above can be achieved if we can have a strategy to retire old samples from the active set to enforce a maximum size and consequently run time. The reasonable constraint was how long are we willing to wait for a score to be computed for a new test case.

To build an incremental solution following points need to be addressed:

1. On receiving new test cases, the distance matrix should not be computed from scratch every time for the entire dataset i.e. existing test case data stored in the system and the data for the new test case for which prediction is to be done.
2. Define a scoring function to calculate score for new test case.
3. The distance matrix fed to the HDBSCAN algorithm cannot grow linearly every time a new case data arrives. This required a strategy to be defined to retire old data once the size of the data grows beyond a certain threshold value.

The above points are addressed in the following way

1) Distance matrix calculation optimization: consider that we have three existing items Item1, Item2, Item3 for which we have already calculated the distance matrix and it looks like the following:

<table>
<thead>
<tr>
<th></th>
<th>Item1</th>
<th>Item2</th>
<th>Item3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item1</td>
<td>0</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Item2</td>
<td>8</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Item3</td>
<td>2</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 1. Example of distance matrix optimization
Let’s assume that we receive a new item “Item4”. The new list will comprise of items Item4, Item1, Item2 and Item3. The matrix table will look like the following:

<table>
<thead>
<tr>
<th></th>
<th>Item4</th>
<th>Item1</th>
<th>Item2</th>
<th>Item3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item4</td>
<td>0</td>
<td>d1</td>
<td>d2</td>
<td>d3</td>
</tr>
<tr>
<td>Item1</td>
<td>d1</td>
<td>0</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>Item2</td>
<td>d2</td>
<td>8</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>Item3</td>
<td>d3</td>
<td>2</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2. Example of distance matrix update

As shown in the above matrix if we only calculate the distance of new item “item4” with corresponding existing items “item1”, “item2”, “item3” and prepend it to the existing matrix we can avoid the expensive operation of calculating the distance of entire matrix altogether and saves substantial amount of calculation time.

2) Scoring function: DBSCAN provides cluster probabilities which is used to determine the strength with each point is a member of the cluster. The score for each new item was calculated in the following manner:

```python
# cluster samples
clstr = hdbscan.HDBSCAN(metric='precomputed', min_cluster_size=5).fit(self.dists)
label = clstr.labels_[0]
labels = clstr.labels_[1:]
if not -1 == label: # discard noise points
    selector = labels == label
    prob_new_item = clstr.probabilities_[0]
    probs = clstr.probabilities_[1:][selector]
    score_new_item = prob_new_item * probs.dot(self.scores[selector]) / len(probs)
```

Figure 18. Sample code for scoring

Separate numeric weights were assigned to each type (errors, alarms and counters) in the data set. The score for every individual item in each type was multiplied with its respective weight factor and the final test case score was summation of score of each individual item in the test case.

3) The final issue of unbounded growth of data which is fed to the HDSBSCAN clustering was addressed by removing stale entries. The mechanism chosen was to remove the stale entries from the largest cluster with the lowest score.

After the above foundations for the system were laid down, they were put together to create an application which exposes a rest interface to accept new test cases and returns score for the test case as response. Mongo DB
was added for the persistence layer so that existing data is not lost during upgrade and application restarts.
5 Evaluation

This section describes the evaluation of the developed framework called fuzz-score-feedback. As mentioned above we performed benchmark test for each of the steps above, starting from distance functions evaluations, choosing the most suitable clustering algorithm and then combining the steps to develop the final application which is called score-fuzz-feedback.

Following steps were followed to test and evaluate the built prototype:

- A HTTP client was developed to POST test case logs to the application.
- The developed application was deployed in the flask web framework.
- The http client read the test case logs and sent it to the score-fuzz-feedback application.
- The application returned the feedback in the form of score as HTTP response.

5.1 Results

We can see from the following sample results that score value is varying on requests and response time is around one second.

```
score = 0.37252799999999997 ,time taken(seconds) = 1.5730900764465332
score = 0.114624 ,time taken(seconds) = 0.3350306034058135
score = 0.34935025026946476 ,time taken(seconds) = 1.192068338394165
score = 0.3183190379196776 ,time taken(seconds) = 0.5110292434652383
score = 0.114624 ,time taken(seconds) = 0.5050289630859893
score = 0.31293163980402194 ,time taken(seconds) = 0.5070286181304932
score = 0.312971100227017 ,time taken(seconds) = 0.5050289630859893
score = 0.37252799999999997 ,time taken(seconds) = 0.529030084609854
score = 0.3494251613269027 ,time taken(seconds) = 1.576090974273652
score = 0.114624 ,time taken(seconds) = 0.5060288506097412
score = 0.114624 ,time taken(seconds) = 0.5060292224884033
score = 0.37252799999999997 ,time taken(seconds) = 0.5010287761668232
score = 0.37252799999999997 ,time taken(seconds) = 0.523029042297363
score = 0.37252799999999997 ,time taken(seconds) = 0.5140295026666623
score = 0.29002875902642355 ,time taken(seconds) = 1.224069337554932
score = 0.29005963130465157 ,time taken(seconds) = 0.5060292224884033
score = 0.2899446857007794 ,time taken(seconds) = 0.5480313301064126
score = 0.37252799999999997 ,time taken(seconds) = 0.6090347766076221
score = 0.28993186171824057 ,time taken(seconds) = 1.2350707054138154
score = 0.289967995208145 ,time taken(seconds) = 0.5080292224884033
score = 0.34947451159315573 ,time taken(seconds) = 1.615925159434346
score = 0.28988050281650374 ,time taken(seconds) = 1.2360706329345703
score = 0.20965224563630177 ,time taken(seconds) = 0.5110292434682033
```

Figure 19. Scoring results
6 Conclusions and Future work

This chapter presents the conclusion of our analysis and research work carried out, limitation and future work which can be continued to further improve and refine the built prototype.

6.1 Conclusions

This thesis work presents a framework from data pre-processing step to until the final test case feedback. The in-depth study of the data and various machine learning techniques led to build a near real time test case feedback in the form of score. Based on various combination of mathematical techniques the solution incrementally builds and keeps learning. The developed solution offers a rest interface which can be easily integrated with the fuzzer.

The aim of the thesis work was to offer a solution which has mechanism to guide the fuzzer in picking the most promising test cases. At the end of the thesis work, the developed framework provides the feedback in the form of score to the fuzzer.

6.2 Future work

- One possibility for better score feedback will be to work with more dynamic and large data sets. Due to some limitations with the existing system, we had to work on the old data sets which were kind of static since they were limited in size.
- Machine learning algorithms require revisiting the models and tuning various parameters from time to time. It is an iterative and continuous process and there is no limit. The scoring function can be improved by experimenting with different weights assigned to each type.
- Experimenting with alternate deep learning techniques as they are proving very useful in NLP and text classification.
7 References

http://alex.smola.org/drafts/thebook.pdf


[3] [website], date of access : 30.10.2018. 
https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data.


[6] Machine learning wiki [website], date of access: 15.10.2018, 
https://en.wikipedia.org/wiki/Machine_learning


[10] How HDBSCAN works, [website], date of access: 10.10.2018, 


https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd


[22] Konstantinidis Stavros, Computing the edit distance of a regular language. “Department of Mathematics and Computing Science, Saint Mary’s University, Halifax, NS, Canada B3H 3C3 Received 19 July 2005; revised 17 March 2007”.

“Department of Computer Science the University of Waikato, Hamilton, New Zealand”.


[26] Polamuri Saimadhhu, April 11, 2015. [website], date of access: 2.09.2018. FIVE MOST POPULAR SIMILARITY MEASURES IMPLEMENTATION IN PYTHON.


[29] editdistance 0.5.2, Last Released: Sep 18, 2018. [website], https://pypi.org/project/editdistance/


[34] Simonini Thomas, Date of access: 01.09.2018. An introduction to Reinforcement Learning. https://medium.freecodecamp.org/@thomassimonini


[37] Cluster analysis, wiki, [website],
https://en.wikipedia.org/wiki/Cluster_analysis

[38] Raut A. B., Bamnote G. R., Software clustering: An overview.
“Special Issue of IJCCT Vol.1 Issue 2, 3, 4; 2010 for International Conference [ACCTA-2010], 3-5 August 2010”.


[40] Lutins Evan, Sep 6, 2017. DBSCAN: What is it? When to use it? How to use it?
https://medium.com/@elutins/dbscan-what-is-it-when-to-use-it-how-to-use-it-8bd506293818

[41] Prado Salton do Kelvin, April 1, 2017. How DBSCAN works and why should we use it?
https://towardsdatascience.com/@kelvin_sp


https://towardsdatascience.com/lightning-talk-clustering-with-hdbscan-d47b83d1b03a

[46] API Reference, [website],


[60] Veroustraete Frank [website].
https://www.researchgate.net/post/What_is_the_difference_between_K-MEAN_and_density_based_clustering_algorithm_DBSCAN


[62] Pathak Manish, September 13th, 2018. Introduction to t-SNE.

[63] API Reference, HDBSCAN [website].
https://hdbscan.readthedocs.io/en/latest/api.html#hdbscan.prediction.all_points_membership_vectors

[64] Use sklearn DBSCAN model to classify new entries, [website].

[65] Semi-Supervised Machine Learning [website].


[67] Breiman Leo, Cutler Adele. Random Forests Leo Breiman and Adele Cutler. [website].

[68] Support Vector Machines (SVM) Introductory Overview. [website].
http://www.statsoft.com/Textbook/Support-Vector-Machines#Classification

[69] Flask (web framework), wiki, [website].


[71] What is MongoDB, [website], date of access: 26.5.2018.
https://www.mongodb.com/what-is-mongodb

https://www.tutorialspoint.com/mongodb/mongodb_advantages.htm

[73] wiki, [website], date of access: 22.11.2018.
https://en.wikipedia.org/wiki/Fuzzing

[74] Rouse Margaret, [website], date of access: 26.11.2018.
https://searchsecurity.techtarget.com/definition/fuzz-testing
[75] [website], Date of access: November 20, 2018. https://www.guru99.com/fuzz-testing.html

Appendix A
Appendix B
To create a cover for the thesis, use the link:

http://intra.kth.se/kth-cover/