SupportAssist: Auto-Triaging Network Issues and Identifying Hidden Anomalies using Machine Learning

Project Report

Mohina Ahmadi

Abstract

When defects or unexpected behavior are seen in any enterprise system, users collect diagnostic information from the system and share them for further debugging/root-causing. For Aruba’s wired switching, this information is collected as “support-files”. Support-files contain information about the diagnostic state of the switch at the time of the issue. This includes event logs, in-memory logs of daemons, data in common files/memory shared by daemons etc. These support-files are huge in size ( ~1GB) and hence, it takes time for analyzing this data manually. It is often desired to optimize the workflow by speeding up the analysis of support-files. Machine Learning has been extensively used in recent days to analyze the patterns in the data and to make successful predictions. This work aims to improve the diagnostic workflow by efficient triaging, assigning right issues to the right team at the earliest, identifying hidden issues and assisting in debugging with the advent of Machine Learning Techniques.

Contents

Abstract

[1.Objective 4](#_Toc75784900)

[2. Proposed Solution 4](#_Toc75784901)

[3.Background 6](#_Toc75784902)

[3.1 K means Clustering Algorithm 6](#_Toc75784903)

[3.2 K-Nearest Neighbors Algorithm 7](#_Toc75784904)

[3.3 Ways to perform Multi Label Classification 7](#_Toc75784905)

[3.3.1 Problem Transformation 7](#_Toc75784906)

[3.3.2 Algorithm Adaptation Methods 9](#_Toc75784907)

[3.4 Measures to select a classifier 11](#_Toc75784908)

[3.5 Box Plot 12](#_Toc75784909)

[3.6 Isolation Forest Algorithm 12](#_Toc75784910)

[3.7 Local Outlier Factor Algorithm 12](#_Toc75784911)

[4. Implementation 14](#_Toc75784912)

[4.1 Case-1 14](#_Toc75784913)

[4.2 Case-2 18](#_Toc75784914)

[4.3 Case-3 19](#_Toc75784915)

[5.Result 21](#_Toc75784916)

[6.Conclusion and Future Work 21](#_Toc75784917)

List of Figures

Figure 2.1:Block Diagram 5

Figure 3.1:Binary Relevance 7

Figure 3.2:Classifier Chains 8

Figure 3.3:Label Powerset 9

Figure 3.4:Box plot 12

Figure 4.2: Elbow method 15

Figure 4.3:Silhouette Method 15

Figure 4.4:Clusters 15

Figure 4.5:Normal data points and anomalies 16

Figure 4.6:Anomalies 17

Figure 4.7:Anomalies of daemons 17

Figure 4.8:Target labels 18

Figure 4.9:Box plot for mactables 20

Figure 4.10: Box plot for UBT 20

Figure 5.1: Input 22

Figure 5.2:Output 21

# 1.Objective

The major objective of this work is to accelerate the triaging and debugging network issues, for the three primary problems mentioned below:

1. Identifying the right owner to own and debug the issue: Manual triage is laborious, expensive and error prone. Multiple teams work on a product and hence, a certain issue could be relevant to one of the many teams. It is often time-taking to identify the right team to debug the issue which in turn delay the resolution time for the field and customer.
2. Identifying hidden issues: Usually the engineer scans the support-files focusing only on the reported issues, whereas there could be many other hidden issues across switch, belonging to different modules/protocols which are being overlooked, may be reported eventually by same customer.
3. Identifying focus areas prone to issues: Support-files contain lot of information. Hence, it is time-consuming to inspect the entire data in search of issues. The main problem in debugging is “where to look for and what to look for”.

It is important to solve these problems as it helps in resolving the issues quickly, enhances the productivity and the debugging efficiency.

# 2. Proposed Solution

To improve the network diagnostics, a tool “SupportAssist” is developed which helps in auto-triaging, identifies hidden issues present in support files and detects critical/focus areas where issues can be found.

1. In order to improve triaging, the tool classifies the support-files into various categories (i.e, features, modules, plug-ins) and assigns them to the right feature team based on the category. This reduces the unnecessary delays and helps in identifying the right feature team quickly.

This is implemented using K-Means Clustering algorithm to classify the CR, by analyzing various feature history/activities and classifying them into various clusters based on the activity. When a new support-file is fed to the tool, it returns the cluster to which this file can belong to using K-Nearest Neighbor algorithm.

1. To further identify other masked issues present in the support files, the tool maps the support files to multiple possible issues. It helps in identifying the potential problems faced by the customers in advance. It is a proactive approach to reduce the further downtime.

The mapping is implemented using Multi Label classification algorithms. When a new support-file is fed to the tool, it identifies and predicts all the possible issues present in that support-file.

1. As it is difficult to inspect all the files, the tool finds all the files which contain information relevant to the problem being analyzed as they have a high probability of finding issues. These files should be checked primarily. This increases the debugging efficiency and it draws the attention of the engineer to the critical areas where issues can be found.

The files which are prone to issues are found by detecting anomalous daemons using Isolation Forest algorithm. When a new support-file is fed to the model, it identifies all the anomalous daemons.

Figure 2.1 shows the block diagram for the proposed “SupportAssist”. A new CR (**Change Request**) number is given as the input. It fetches the respective support-file from the database (aruba/cr\_data), preprocesses and determines the data distribution by statistically computing different types of messages (event log, debug log, fast logs) per module, per severity. The data is cleaned (highly correlated and redundant data is removed), machine learning algorithms are applied on this data and the required output is obtained

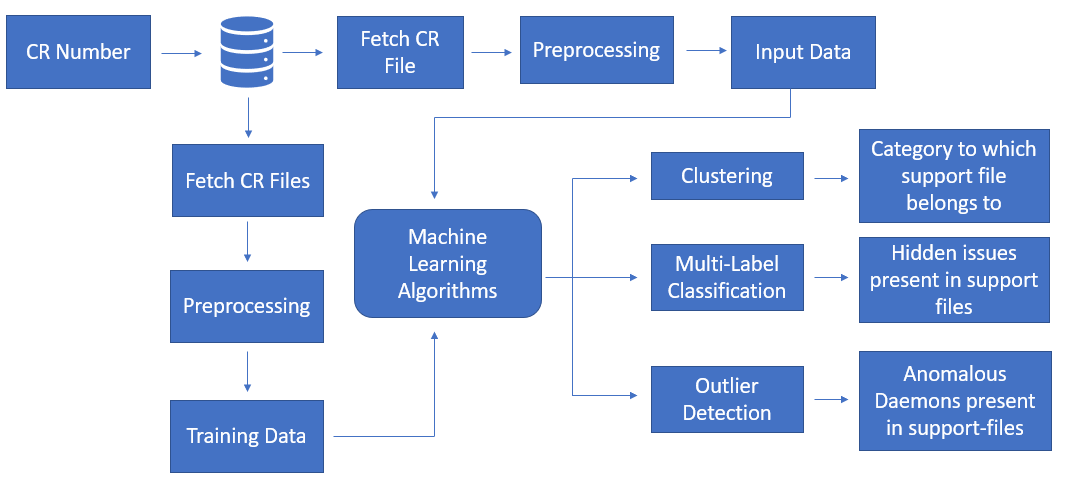


Figure 2.1: Block Diagram

# 3.Background

## 3.1 K means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm which groups the unlabeled dataset into different clusters. ‘K’ indicates the number of clusters which is decided using the Elbow method or Silhouette Method.

K means++:

* First centroid is selected randomly
* Next centroid is chosen such that its distance is maximum from the previously chosen centroid.
* Hence, centroids are far from each other, and data points lie in different clusters.

Elbow Method:

The sum of square distance from each data point to the nearest centroid is calculated for various K values. It consists of plotting the explained variation as a function of the number of clusters and picking the elbow of the curve as the number of clusters to use.

Silhouette Method:

The silhouette method computes silhouette coefficients of each point that measure how much a point is similar to its own cluster compared to other clusters. The Silhouette Coefficient is calculated using the mean intra-cluster distance (a) and the mean nearest-cluster distance (b) for each sample. The Silhouette Coefficient for a sample is (b - a) / max (a, b).

K value for which Silhouette coefficient is higher is considered as the optimal number of clusters.

Working of K Means:

Keep iterating until there is no change to the centroids. i.e, assignment of data points to clusters is not changing.

* Compute the sum of the squared distance between data points and all centroids.
* Assign each data point to the closest centroid.
* Compute the centroids for the clusters by taking the average of all the data points that belong to each cluster.

The approach K means follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the closest cluster. The M-step is computing the centroid of each cluster.

## 3.2 K-Nearest Neighbors Algorithm

A K-nearest-neighbor is a data classification algorithm that attempts to determine what group a data point is in by looking at the data points around it.

* The distance between new data and the all the data points in the training dataset is calculated
* The distances are sorted in ascending order.
* First K entries from the sorted collection are picked.
* The mode of the K labels is returned, that is the label which has occurred a greater number of times.

## 3.3 Ways to perform Multi Label Classification

### 3.3.1 Problem Transformation

In this approach, the multi-label classification problem is transformed to a single label classification problem (multi-class). There are three different techniques for this kind of transformation.

1.Binary Relevance:

The problem is broken into N binary classification problems with N being the total number of Labels. Each one of the binary classifiers predicts if the label belongs to the sample or not.

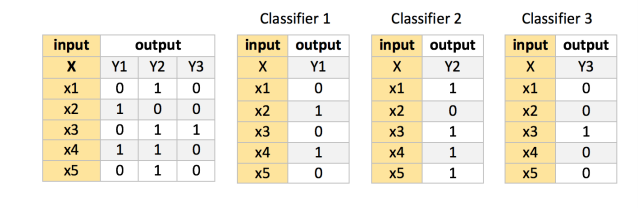


Figure 3.1: Binary Relevance

**Strong sides:**

* Estimates single-label classifiers
* Can generalize beyond available label combinations.

**Weak sides**:

* Not suitable for large number of labels
* Ignores label relations

2.Classifier Chains

The problem is broken into N binary classification problems, but the difference is that in this approach the prediction of the first classifier in the chain is considered as an extra feature attribute in the process of predicting of the other labels. This adjustment helps preserving the correlation between the labels.

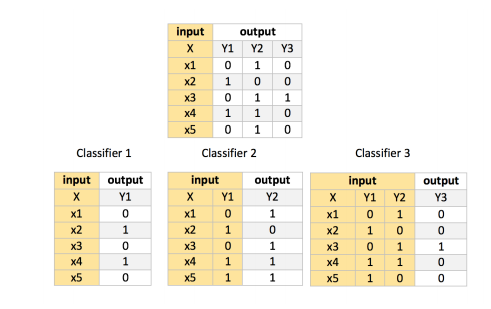


Figure 3.2: Classifier Chains

**Strong sides**:

* Estimates single-label classifiers
* Can generalize beyond available label combinations
* Takes label relations into account

**Weak sides**:

* Not suitable for large number of labels
* Quality strongly depends on the label ordering in chain.

3.Label Powerset

This approach transforms the problem into a multi-class classification problem by mapping all the unique combinations of labels that appears in the dataset to a single class. After the transformation one multi-class classifier is trained on the dataset to predict the label combination of the input sample.

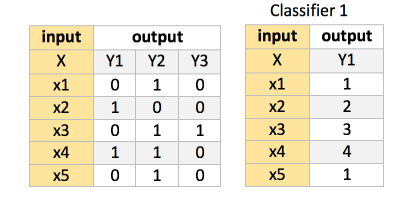


Figure 3.3: Label Powerset

**Strong sides**:

* Estimates label dependencies, with only one classifier.
* Often best solution for subset accuracy if training data contains all relevant label combinations.

**Weak sides**:

* Requires all label combinations predictable by the classifier to be present in the training data.
* Very prone to underfitting with large label spaces.

### 3.3.2 Algorithm Adaptation Methods

1.MLKNN:

It is a multi-label lazy learning, which is derived from the traditional K-nearest neighbor (KNN) algorithm. For each unseen instance, its K nearest neighbors in the training set are identified and based on statistical information gained from the label sets of these neighboring instances, the maximum a posteriori (MAP) principle is utilized to determine the label set for the unseen instance.

**Strong sides**:

* Estimates one multi-class sub classifier
* Works when distance between samples is a good predictor for label assignment
* Often used in biosciences.

**Weak sides**:

* Requires parameter estimation

2.BRKNN:

Binary Relevance multi-label classifier based on k-Nearest Neighbors method. This version of the classifier assigns the labels that are assigned to at least half of the neighbors.

**Strong sides**:

* Takes some label relations into account while estimating single-label classifers.
* Works when distance between samples is a good predictor for label assignment.

**Weak sides**:

* Trains a classifier per label
* Less suitable for large label space
* Requires parameter estimation.

3.RakelD: [Random label space partitioning with Label Powerset]

Divides the label space into equal partitions of size k, trains a Label Powerset classifier per partition and predicts by summing the result of all trained classifiers.

Strong sides:

* May use less classifiers than Binary Relevance and still generalize label relations while not underfitting like Label Powerset

Weak sides:

* Using random approach is not very probable to draw an optimal label space division.

4.Majority Voting Ensemble Classifier

Divides the label space using provided clusterer class, trains a provided base classifier type classifier for each subset and assign a label to an instance if more than half of all classifiers (majority) from clusters that contain the label assigned the label to the instance.

Strong sides:

* accommodates to different types of problems
* infers when to divide into subproblems or not and decide when to use less classifiers than Binary Relevance
* scalable to data sets with large numbers of labels
* generalizes label relations well while not underfitting like Label Powerset
* does not require parameter estimation.

Weak sides:

* requires label relationships present in training data to be representable of the problem.

## 3.4 Measures to select a classifier

Hamming loss measures how well the classifier predicts each of the labels, averaged over samples, then over labels.

*Accuracy score* measures how well the classifier predicts label combinations, averaged over samples.

*Jaccard similarity* measures the proportion of predicted labels for a sample to its correct assignment, averaged over samples.

*Precision*measures how many samples with. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

*Recall*measures how many samples. The recall is intuitively the ability of the classifier to find all the positive samples.

*F1 score* measures a weighted average of precision and recall, where both have the same impact on the score.

## 3.5 Box Plot

A boxplot is a graph that gives a good indication of how the values in the data are spread out.

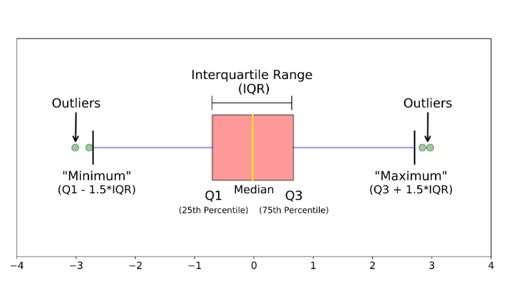


Figure 3.4:Box plot

## Isolation Forest Algorithm

Isolation forest works on the principle of recursion and it helps in getting the anomalies isolated faster.

* This algorithm recursively generates partitions on the datasets by randomly selecting a feature and then randomly selecting a split value for the feature.
* The anomalies need fewer random partitions to be isolated compared to the so defined normal data points in the dataset.
* Therefore, the anomalies will be the points that have a shorter path in the tree. Here, we assume the path length is the number of edges traversed from the root node.

## 3.7 Local Outlier Factor Algorithm

Local outlier factor (LOF) is an algorithm used for Unsupervised outlier detection. It produces an anomaly score that represents data points which are outliers in the data set.

It does this by measuring the local density deviation of a given data point with respect to the data points near it.

Working of LOF:

Local density is determined by estimating distances between data points that are neighbors (k-nearest neighbors). So, for each data point, local density can be calculated. By comparing these we can check which data points have similar densities and which have a lesser density than its neighbors.

K-distance: Distance between the point and its kth neighbor.

K-Neighbors: The set of points that lie in or on the circle whose radius is the kth distance.

Reachability Distance: The reachability distance of a point *a* to *b* is the maximum of K-distance of *a* and the distance between *a* and *b*. If the point *a* is within the circle, then reachability distance is equal to K-distance. Else, it is equal to the actual distance between *a* and *b.*

Local Reachability Density(lrd): To get the lrd for a point a, the reachability distance of a to all its k nearest neighbors is calculated and average of that number is taken. The lrd is the inverse of that average. More the average reachability distance, lesser will be the density of points.

Local Outlier Factor: The LOF is the average ratio of the lrds of the neighbors of a to the lrd of a. LOF of a point tells the density of this point compared to the density of its neighbors. If the density of a point is smaller than the densities of its neighbors then it is considered as an outlier.

3.7.1 Novelty Detection

The training data is not polluted by outliers, it contains only good data and we are interested in detecting whether a **new** observation is an outlier. In this context an outlier is also called a novelty.

# 4. Implementation

## 4.1 Case-1

1)Creating Dataset

* A python script is written which fetches the support-file data from /aruba/cr\_data and estimates the count of Fastlog messages and writes them into a csv file. Similarly, the count of log messages and percentage of memory data is also extracted.
* When the dataset contained only the count of fastlog messages number of clusters were 2. To improve the classification, count of logs mand memory data were added into the dataset.

2)Data Cleaning

* The dataset is extracted into a pandas data frame in order to perform further operations.
* The dataset contains count of messages of CRs as rows (67) and fast log messages of various daemons, logs, cpu percentage, memory percentage data as columns. (7316)
* The columns with low variance and the highly correlated columns are removed as they are not suitable for Machine Learning. [67 rows and 923 columns]

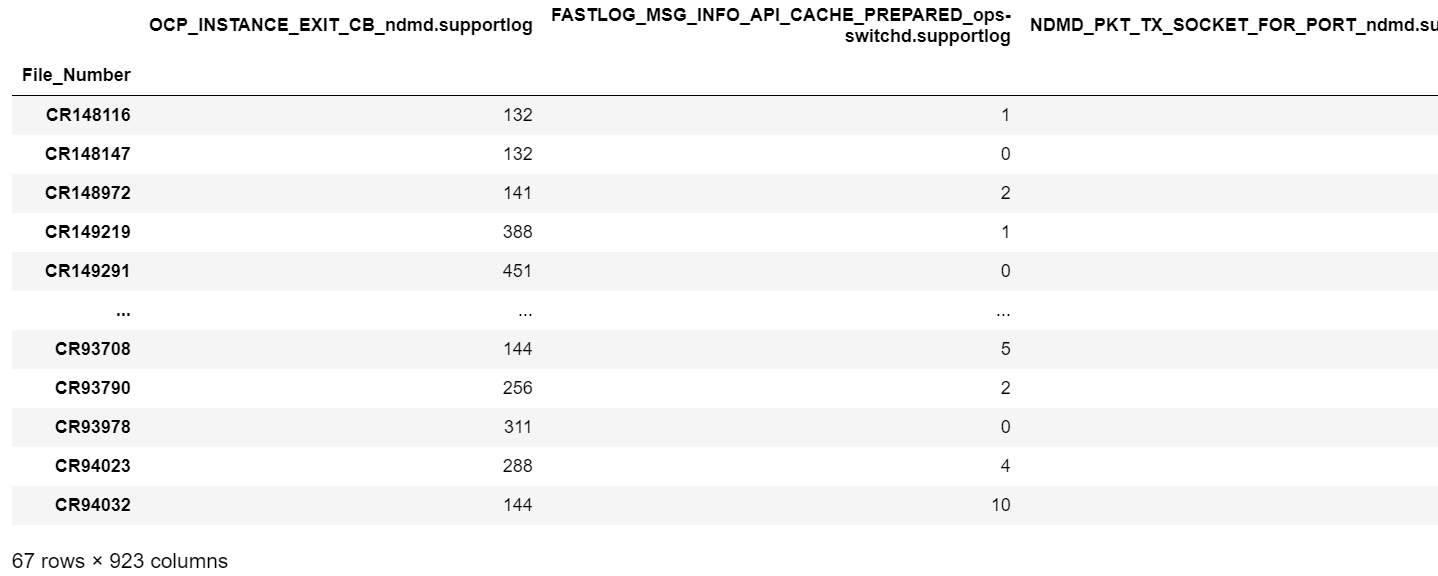


Figure 4.1: Dataset

3)Clustering

* Based on Silhouette method and Elbow method, K is estimated to be 4.
* K means Algorithm with K=4 is applied on this data and CRs are classified into 4 clusters.

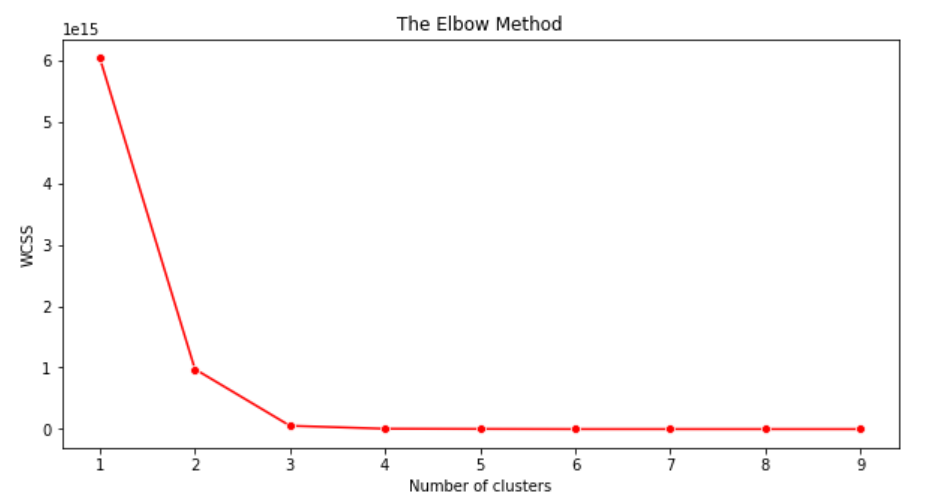
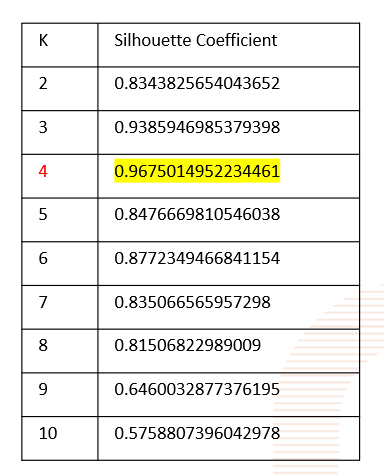
 

Figure 4.2: Elbow method Figure 4.3: Silhouette Method

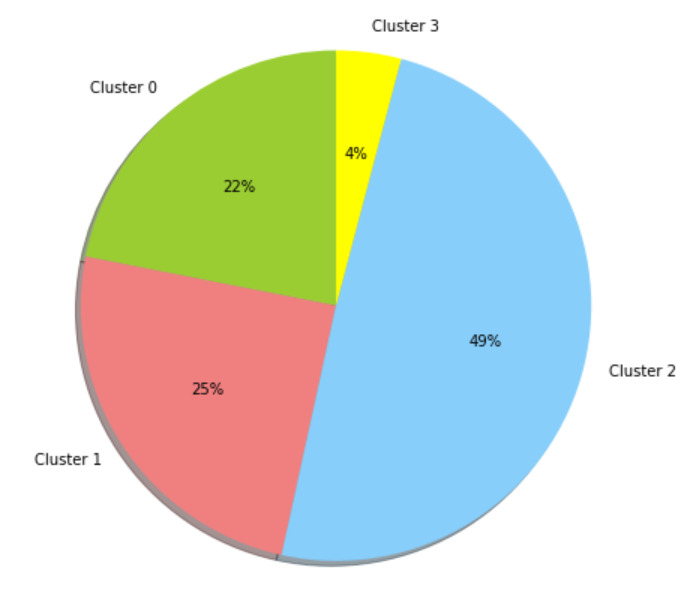


Figure 4.4: Clusters

4)The component of each CR is found using JIRA and it is observed that all CRs having the same component are not in one cluster.

5)KNN Algorithm is used to predict the cluster to which a new CR file belongs to.

**Observations**:

* No specific similarity is seen in the CRs which fell into same cluster.
* Using Fastlog messages of all the daemons together did not help in classification.
* A proper pattern is not found in the data points.
* CRs which have issues related to different components had similar data points. Hence, they couldn’t be differentiated.

Example

Consider the points in below representational 2-d graph (assuming each point represents fastlog counts of a certain daemon from a certain test/customer-setup instance). It has two kinds of data-points. The clusters cover data-points from those instances where issues where not seen are not anomalous. The data-points outside the clusters are anomalous which stand for the instances where an issue was seen and a CR was created.

Observation 1:Anomalous data-points would not fall in any cluster as shown in Figure 4.5. Hence, they do not conform to a statistic. i.e., Each cluster would have its mean and standard deviation values within which all its points lie. But these values are not relevant to the anomalies.

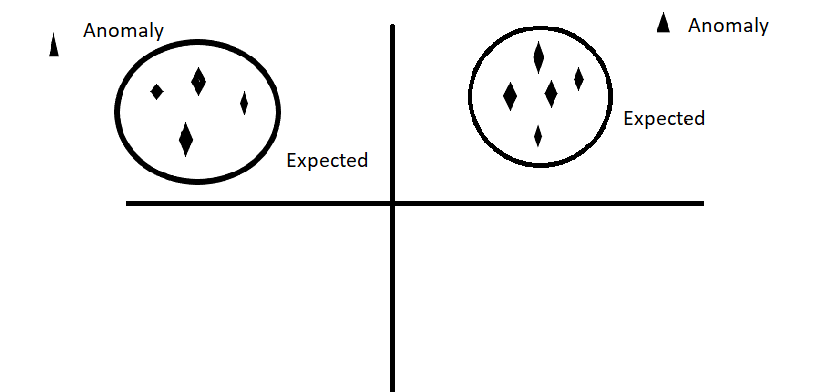


Figure 4.5: Normal data points and anomalies

Observation 2:

In this case, CR data-points were considered. Hence, only anomalies were taken. The picture of data-points is shown in Figure 4.6.

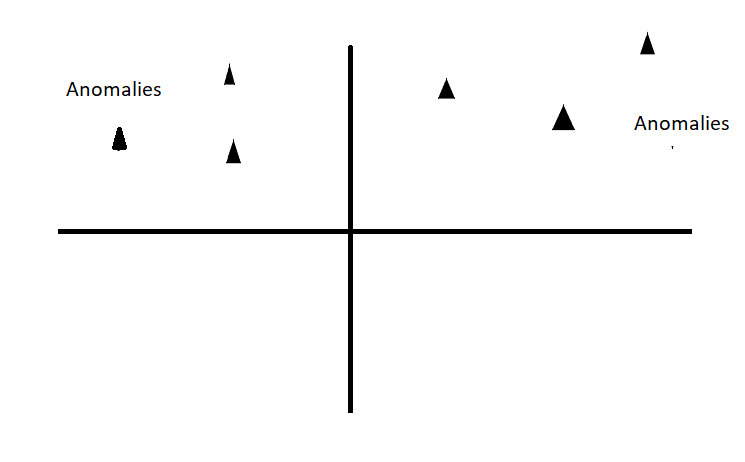


Figure 4.6: Anomalies Observation 3:

Not just one daemon but all the daemons were considered together as shown in Figure 4.7. It should be noted that the graph contains only anomalies and that too from multiple daemons.

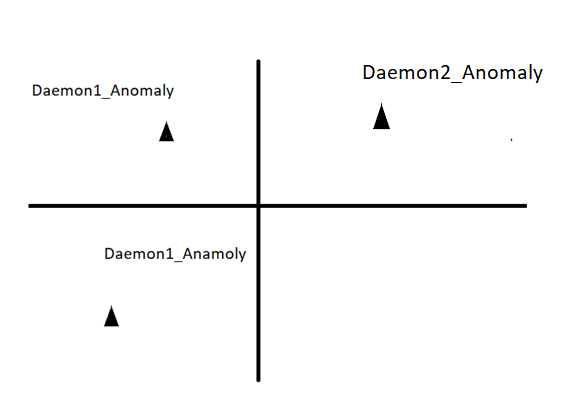


Figure 4.7: Anomalies of daemons

Observation 4:So, if the clustering algorithm is applied on these anomalies, the anomalous points of different daemons are clustered together. So, this defeats our aim of trying to distinguish between CRs.

**Conclusion**:

This approach does not seem to be effective as there is no certain pattern seen in the data points related to one component. Hence, it is not possible to segregate the CRs into various components for this data. Better observations could be made by selecting proper features and accurate data.

## 4.2 Case-2

1)The dataset created in case-1 is used and the issues present in each CR are taken from JIRA. These issues are used as labels as shown in the Figure 4.8.

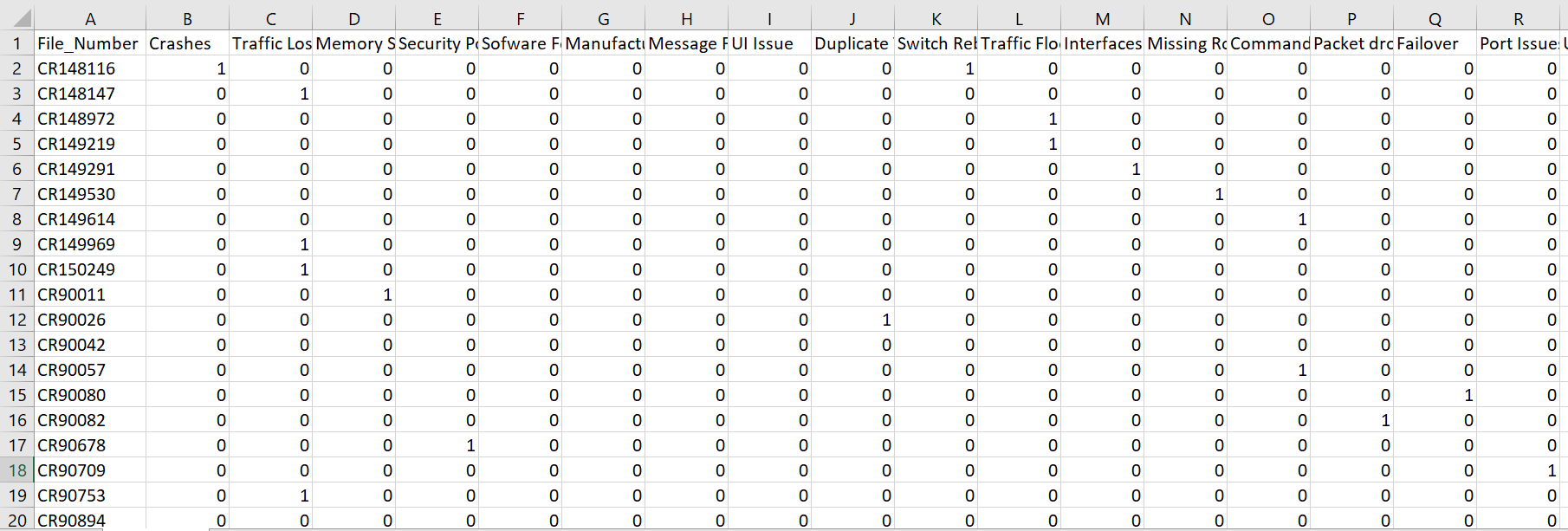


Figure 4.8: Target labels

2)Multi Label Classification is a predictive modeling task that involves predicting zero or more mutually non-exclusive class labels. It involves Problem Transformation methods and Algorithm Adaption methods as explained in the section 3.2.

3)These algorithms were applied, and the possible issues present in the CR are predicted based on the data.

4)Manually labelling data is a huge task. So, a semi-supervised learning approach is devised where a self- training classifier learns from labeled data and assigns the labels to unlabeled data.

**Observations:**

* The problems faced in case-1 hold true in this case too.
* The issues for each CR are not assigned accurately.

**Conclusion**:

Better observations could be made by choosing the right features and by figuring out all the issues present in CRs in order to train the model.

## 4.3 Case-3

1)Creating dataset:

* The count of fastlog messages for specific daemons like hpe-cardd and ops-switchd are collected in the dataset.
* The percentage contribution of each fastlog messages among total number of messages is calculated and dataset is created with this data.

2)The count of LOG ERR, LOG\_CRIT messages are taken from event.log file for the respective CRs.

3)Statistical Analysis

* Mean and standard deviation of all fast log messages is calculated to analyse the data.
* Z score outlier method (observations which are not in the range of mean +/- standard deviation as outliers) cannot be applied as the data is not normally distributed.
* Outlier detection based on IQR do not work well because the data points do not have a good spread.

5)Hence, Isolation Forest and LOF algorithms were applied to find the outlier CRs in the dataset.

6)The output was not satisfactory as CRs which do not have any LOG ERR percentage were also predicted as outliers.

Another approach

1)The daemons which have a good spread of fastlog distribution may be useful for analysis. Hence, five daemons tunnelednode (user based tunnel) and l2mac-mgrd (Mac tables), hpe-vsxd(VSX), radius-srv-trkd, ipsavd(DHCP Snooping) are chosen which seemed to have a good spread as shown in Figure 4.9 and 4.10.

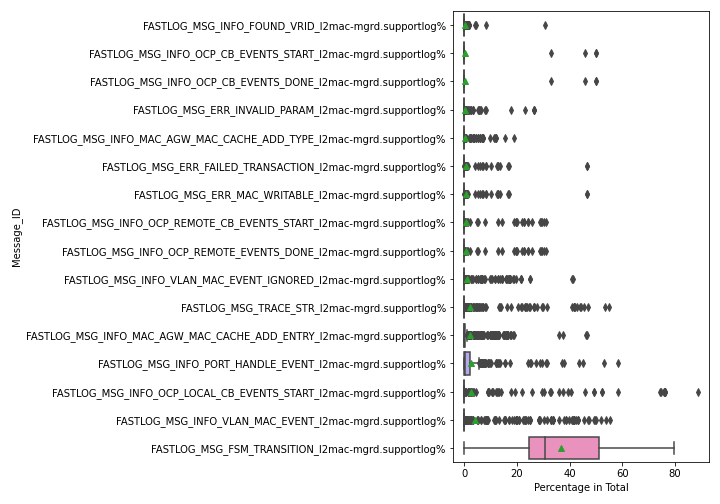
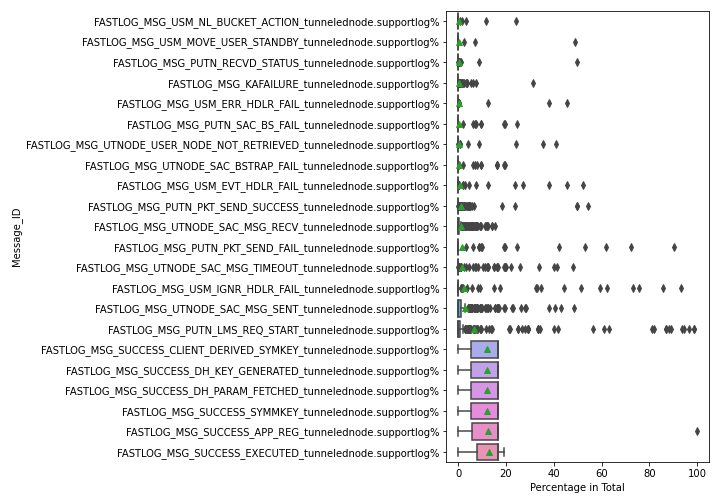
 

Figure 4.9: Box plot for mactables Figure 4.10: Box plot for UBT

2)From JIRA, CRs for these 5 daemons with fixVersion=10.08 from month of June (since old files may no longer be present in /aruba/cr\_data). However, only two daemons tunnelednode (12 CRs) and l2mac-mgrd (8 CRs), had good number of support-files available for analysis.

3)The assumption here is that fastlog counts of collected tunnelednode CRs are anomalies and the other CRs are expected to have normal fastlog count. Similarly, for mac tables.

4)Row cleaning: The mean is calculated for the existing CRs. If row's mean is low, it means the daemon had very little activity. So, the rows with less mean are removed.

5)Column cleaning: correlation method and PCA analysis were tried but stuck to correlation because its simpler and existing data is kept as is.

6)Finally the dataset contains

i)Dataset 1: 65 rows and 16 columns

ii)Dataset 2:99 rows and 14 columns

7)Novelty Local outlier factor algorithm is applied on this data with the non-anamolous data as training data and anomalous data as test data.

8)The algorithm predicted ‘0’ outliers for UBT and ‘4’ outliers for Mactables.

Alternative data:

18 CFDs were taken assuming that customer setup should have enough activity for ubt/l2mac-mgrd. But even CR having UBT as a component had very little fastlog count which deviates from our assumption that CRs having UBT as their component will have high fast log count for tunnelednode daemon

**Observations:**

* Outliers are not predicted accurately.
* The number of rows and columns are too less for data analysis.
* Data at normal scenario is needed (good data).

**Conclusion:**

The obtained results are inconclusive, because of inadequate and inaccurate data. If enough data is obtained from customers which have real use cases for ubt/Mac-tables (or other daemons of interest) (preferably from same release, so that fastlogs can be compared), then algorithms can be run on the right data and right conclusions can be derived on whether this approach is useful or not.

# 5.Result

In VM, a script is written which executes all the above three cases and gives the output. The UI is implemented using Tkinter.

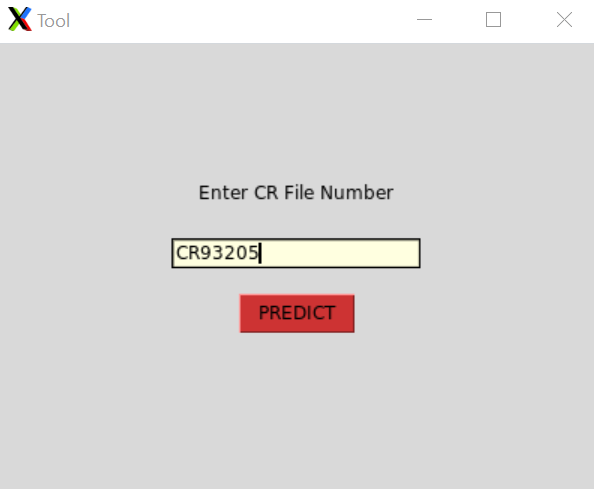
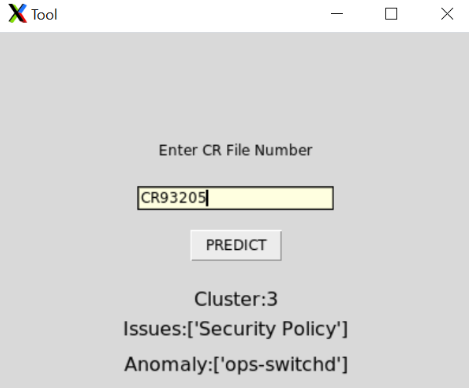
 

Figure 5.1: Input Figure 5.2: Output

# 6.Conclusion and Future Work

The proposed approach has been implemented but the results are indecisive. The reason behind this could be the features used and the less amount of data. Though the results obtained were not satisfactory, this work enabled deeper research of Machine Learning techniques. This work can be further improved by selecting targeted features, collecting more number of support-files and support-file data at normal scenario.