CHALMERS
UNIVERSITY OF TECHNOLOGY

Fault Tracing Bot in Electric Drive Software Using Machine Learning

Master's Thesis in Electrical Engineering

Hariharan Gopinath, Soundarya Jai Prakash

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2023
www.chalmers.se

Master's thesis 2023

Fault Tracing Bot in Electric Drive Software Using Machine Learning

Hariharan Gopinath, Soundarya Jai Prakash

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden 2023

Fault Tracing Bot in Electric Drive Software Using Machine Learning
Hariharan Gopinath, Soundarya Jai Prakash

Supervisor: Mathews Peter, Volvo Car Corporation.

Advisor: Mohammad Farsi, Department of Electrical engineering ,Chalmers University of technology.

Examiner: Thomas Erikson, Department of Electrical engineering ,Chalmers University of technology

Master's Thesis 2023
Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2023

Abstract

The Electric Drive Software Continuous Deployment (EDCD) team at Volvo Car Corporation is working towards figuring out the automation of the analysis of the build log data obtained as a result of the continuous software integration for Electronic Control Unit (ECUs) using Machine Learning. Machine Learning is expected to be used in building a fault tracing bot, whose main goal is to analyse the provided log data and find the underlying meaning or information through it with minimum or zero human interaction. As this is an ongoing process and the creation of such log data will never really end, this project could make it easier for the analysis of big projects in getting instant data. At first, we are working on the data cleaning and the preprocessing of the big unstructured log data, followed by the labeling of the data by the clustering process. Lastly, the machine learning algorithm is used to predict the classification of the newly fed data. Many different machine learning algorithms have been implied to compare and get the best accuracy predicting algorithm among them.

Acknowledgements

We would like to thank our supervisor, Mathews Peter (Volvo Cars) for his assistance in interpreting the logs and are also very thankful to our Advisor at Chalmers, Mohammad Farsi, for helping us by giving some great feedback during our meetings and for guiding us forward through this process. We would like to take this opportunity to thank Thomas Eriksson (Examiner) for the academic guidance he has provided. We are very grateful for all the academic support and opportunity that has been provided to us.
Also, we would like to express our gratitude to our managers at Volvo Cars, Johan Zackrisson and Anna Dahl for being pliable with our requirements in this project and helping us throughout the process. This definitely would have not been the same without all the guidance and support we have been provided with.

Hariharan Gopinath & Soundarya Jai Prakash, Gothenburg, May 2022

Contents

1
Introduction

In today's world, the capacity to store and produce data is vastly more than what the human brain can decipher quickly. There has been an ample amount of research done in the fields of Artificial Intelligence (AI), Machine Learning (ML), and Data Mining. These investigations are alluring now more than ever before, as computational power has reached a level where such methods can finally be applicable in increasing numbers in real-world scenarios. Modern software applications are growing faster than ever with the fast pace of new software implementation methods and changes in infrastructures. During software and hardware development, logs are the prime input for understanding and maintaining the entire system. As the log data serves as software-intensive information for troubleshooting the system, the manual error identification process in log data becomes more expensive as the amount of data that needs to be analyzed increases. Fault tracing activities connected to software integration at Electric Drive software are conducted manually to a greater extent, based on experience regarding the types of faults and the main objective. However, since a major amount of data regarding the electric drive software is already available, in terms of reports, configuration, etc, if an automated classification is to be performed initially based on these reports, the amount of manual labor required to analyze the vast amount of fault report could be greatly reduced in the future.

1.1 Purpose

The purpose of this project is to create a bot for analyzing the log data using machine learning, which automatically characterises common faults and suggest solutions that facilitate quicker debugging and support. This would also help the team categorize the errors that have constantly been occurring in multiple jobs which are run on a daily process. Along with that, it also gives the frequency of their occurrence, which would indeed help the team in knowing the importance of a few particular errors and simplify the maintenance of such jobs which increases the productivity of the project.

1.2 Goal

This master thesis aims to develop a ML model that analyzes and automatically finds patterns in the errors and warnings in the log data, which aids in fault tracing, analysis, and visualization of the log data. This seemingly simple classification model

provides huge potential in improving planning, pre-discovery and pre-correction of errors before they are evident. In addition, this master thesis also involves the conversion of the log data into a structured format which is cardinal for training the ML model. Further, this master thesis aims to identify the multiple kinds of faults that occur during the software integration process, which can be used to train the ML model in classifying faults based on the incoming data. These strategies also help reduce manual interventions, improve automation and pave the way for DevOps implementation in automotive software.

1.3 Literature Study

A great deal of researches have been done on log data anomaly detection, but most of the papers are focused on open datasets that are already preprocessed into binary classifications. Till now there seems to be none other available work on multi-class classification for the complex log data that need to be preprocessed with the help of Natural Language Processing (NLP) techniques.

In the literature-survey [1], efficient word embedding methods are developed and used for anomaly detection of the log events during the software development process where most of the log data is unstable. The open source unstable log data is used to experiment with efficient embedding techniques and the binary classification is performed on the same to classify the data into normal data and anomalies. Hereby it demonstrates the robustness of the word embedding, that can be extracted from the preprocessed log data.

In [2], the authors provide suggestions on detecting failure possibilities to monitor log data from the computer system for predictive maintenance. Here the pros and cons of different ML approaches has been discussed instead of the actual approach itself for detecting the failure of the log data from the system. It also provides good instructions for the labeling of the supervised methods and also the testing of the models with the real test data once the model has been trained.

Here in [3], the authors have used various ML algorithms for anomaly detection in the dataset of logs generated by Hadoop Data File System (HDFS) servers which consist of more than 10 million log lines. Various word embedding methods that are suitable for unsupervised techniques are in practise here, for example the clustering of the log data to differentiate it into anomaly and normal data. Also, the performance results of ML methods have been compared using metrics i.e precision, recall, accuracy and F1 score[1] etc, which are also used for identifying the best algorithm by comparison.

In [4], an alarm log data has been used to perform clustering methods for better visualization purpose and to also identify and classify correlated alarms. The word embedding technique, a novel clustering scheme, and a multi-dimensional scaling method have been performed and evaluated. Instead of the troubleshooting log

[1]F1 score is "F1 measure" also known as "F1 statistic". It is a machine learning evaluation metric used to measure a test's accuracy, calculated as the weighted average of precision and recall.

data, other word2vec [5] technique has been performed to convert the alarm events to real numbers which can be used for better clustering and to provide the final output which gives insights to manage alarm flooding events.

In [6], different types of sampling methods have been performed on the ML training to determine the method with better performance, accuracy and Receiver Operating Characteristic (ROC) curve. Oversampling statical methods have been performed on methods i.e Synthetic Minority Oversampling Technique (SMOTE), Tomek[2] , and Edited Nearest Neighbor (ENN). This test is performed on a small dataset that has both majority and minority samples. Here, different sampling methods are discussed and implemented based on the dataset count and visualization.

1.4 Scope and Limitations

The scope of the project includes

- Analyze and prepare the data for annotation. Develop a strategy for annotation of the data and ensure that the data is ready to train a model.
- Select which features to use as model inputs.
- Implement and train a categorical regression model to identify certain faults in software integration.
- Evaluate the performance of the model and analyse the limitations of the model implementation.
- Suggest a root cause analysis for the faults.

The limitations at hand that is to analyse the log data and categorize them and find the root cause of the anomalies usually follow the same high level steps like:
1) The key features need to be extracted from the log data as the log data is imbalanced with large number of healthy or good logs.
2) There are many outliers or unwanted logs within the log data. Such logs need to be cleared or removed while preprocessing the data.
3) ML model has to be applied to the labeled data to find the root cause, or any other kind of analysis that has to be done on the dataset.

1.5 Problem Definition

The Electric Drive Software Continuous Deployment (CD) team, which works with the continuous integration of software components, faces many interface issues/errors which is stored in the form of build logs. There are different kinds of errors in the build logs like the configurations, compiler, linker, eXtensible Markup Language (XML) validations, etc. This thesis focuses on developing a ML model that analyzes and automatically find patterns in the errors and warnings from the log data. This is done along with fault tracing, analysis, and visualization of the fault in the log data. This task wasn't easy and comes with many challenges as the dataset provided was very unstructured and imbalanced. Undersampling and oversampling methods

[2]Tomek is an acronym that stands for "Tomek links". Tomek links refer to a type of pairs of nearest neighbors in a dataset that are of opposite classes and are located close to each other.

like SMOTE mentioned in chapter 3.9.2, were used for balancing the data. The unstructured data was converted into a structured manner by using intensive data preprocessing methods as mentioned in Chapter 3.4.2. The main goal, to find the ML algorithm for the automatic analysis was then carried out by implementing three different algorithms i.e the Gaussian Naive Bayes, Linear Support Vector and the Logistic regression. After choosing the ML algorithm that gave us the best results, the future prediction must be possible on any unseen datasets.

1.6 Research Questions

1. How can data pre-processing be carried out on the extremely unstructured data-set, such that the data can be used in the further processes for applying ML algorithms on it without any hindrance?
2. As the data-set is also unlabelled, how is clustering performed?
3. Can the above clustered data be used for accurate predictions using ML algorithms?

1.7 Report Outline

The organization of the report is as follows: the thesis started with conducting research on the topic by referring to papers that have previously been written on Anomaly Detection issues with log data. The main methods followed here are mentioned in the theory part on Chapter 2. Following this is the Methodology, which describes the steps taken and the algorithms used in this thesis. The Conclusion has the output results and the accuracy of the previously mentioned ML algorithms utilized. It ends with a discussion about the future work which can be done keeping the above topic in mind, which was beyond the topic of this project.

1.8 Background

This Project will be conducted with the EDCD team at Volvo Car Corporation which is working with the integration of software (SW) components in an AUTomotive Open System ARchitecture (AUTOSAR) [7] environment for electric propulsion ECUs. Electronic Control Unit (ECU) software is divided into 3 parts such as base software (BSW), run time environment (RTE), and application software (ASW). RTE is the data and functional connections established between the ASW and BSW. This project will mainly focus on building log data which is a result of software integration for the ECUs. Within the ECU, there are many software modules available, like motor control software, inverter control software and battery monitoring software, etc. They are all connected and data transfers from one to another software. The CD team is mainly responsible for the integration of all the software components and also providing working ECU software. External suppliers provide some of the software modules, while some are engineered within the Volvo cars. Changes in software composition from both the suppliers and Volvo

cars are becoming frequent as it involves multiple people, teams, and organizations. Because of the changes in software development and various tools, many interface issues arise when integrating different application software with the base software. All the interface issues/errors are stored in built log data, and this log data contains all the information about the software integration and the errors in different steps. These errors are of various types such as the configurations, compiler, linker, XML validations, etc. but there is no run time error. Along with errors and warnings, various labels including, date and time on a project timeline, project name, project variants, and more information and metadata, is associated with it. To solve this problem, an experienced software engineer can examine the log data to understand the status of the software systems, detect the anomaly or error log and do fault analysis for the problems that are occurring in the software integration at particular states. Further, when this is done manually it takes time and effort as some of the errors are easy to identify and solve, while some are not, as they tend to be complicated at some point as there are thousands of log data entries and plenty of log data being recorded. This may be a bottleneck for software development. The future should be in the direction of including machine intelligence and automation in improving and implementing these tasks. This led to rising of automated fault pattern identification systems using ML techniques that would help the developers to give an idea about the type of errors or warnings that occurs in a log file and to provide a fault tracing analysis of the errors. This reduces the time to solve with less manual intervention and reduces the effort to search which offers a faster solution. There have been many such work done on the predictive maintenance and anomaly detection, but this thesis mainly focuses on fault prediction for continuous software deployment.

2
Theory

2.1 AI-based Anomaly Detection

In the world of software engineering, producing the log data for troubleshooting purposes is very important in software-intensive systems. These log data monitor the condition of the software development or integration and record the run-time information by printing console logs in the log file. Sometimes, many log files are stored during the software development process. For example, about 30-50 gigabytes of tracing logs are stored per hour in the cloud computing systems of Alibaba Inc [8]. During the process of troubleshooting, the engineers used to analyze the run-time information to find the exact fault or error in the log files. But, manually searching or finding the fault log information is very hard, and will consume a lot of time and resources to go through it. There comes the importance of log analysis.

Log analysis is the process of understanding, and extracting knowledge from the log files for any purpose. This log analysis provides a clear picture of what has happened in the system behavior so that troubleshooting or undesirable behavior in a system can be easily found. Further, manual log analysis is a very tedious process and very time-consuming. Nowadays, there are lots of new techniques that have been found to do automatic log analysis. One such method for detecting the fault in the log data is Log Anomaly Detection.

Log Anomaly Detection is the process of automatically detecting or identifying undesirable behaviors or patterns which are indicating the error in the log file. This Log-based Anomaly Detection can be performed by training a machine learning model using the dataset where the outcome would represent a binary label, either an abnormal or normal pattern in the log file. There are different types of machine learning models like the supervised model [9] and the unsupervised model [10], based on the available labeled data. Further, these machine learning models can be either binary classification or multi-class classification based on the purpose or error types.

2.2 Unsupervised Learning

ML techniques are mainly classified into four types: supervised, unsupervised, reinforcement and semi-supervised learning. Unsupervised learning, is a type of machine learning in which the machines are trained on data that is neither labeled nor classified. This method uses machine learning methods to find, analyze and cluster unlabeled datasets. Based on this, hidden patterns are discovered and different kinds of

data is grouped without the need for human intervention. Unknown patterns in the data can be found by this method. The most commonly used unsupervised learning algorithm is Clustering. In the Clustering method, large amount of input data without any label has been fed into the algorithm and a set of data points which are similar to each other based on their relationship to the surrounding data points are grouped together on its own based on a mathematical model [11]. Another main idea for using the clustering method is to automatically label the dataset based on the similarity group so that all the input data can be analyzed and labeled with the help of developers or learners. The number of clusters also can be modified based on the observations. This method is mostly used to label large datasets which reduces manual labeling. Further, there are different clustering methods, such as K-means clustering (where K is the number of clusters found from data), Hierarchical clustering, Gaussian Mixture Model (GMM), Density-Based Spatial Clustering of Applications with Noise (DBSCAN). In this thesis work, we explore K-means and hierarchical clustering techniques. Other purposes of using the clustering method is to find the outliers in the dataset and to get a good visualization of the well spread data. Further, if any outliers are present in the dataset, they can be easily detected and neglected based on the observation of the outliers. Furthermore, using clustering, the underlying structure or insight of the data can be gained to generate hypotheses and identify the salient features. It also helps to organize the data and summarise it through cluster prototypes [12].

2.2.1 K-means Clustering

This is a popular method of unsupervised machine learning in which a predetermined k-value/cluster value has to be given. Depending on that number, the data set provided is divided into groups [13]. Here the large group of data can be clustered or divided, such that the labeling of the clusters can be carried out. There is no training required for the clustering algorithm. The K-means algorithm mainly depends on the centroid values for each cluster. The K-means is a distance-based or centroid-based algorithm, where the distances to assign a point to a cluster can be calculated. Every cluster has a centroid and the algorithms aims to minimize the sum of distances between the data sets and their corresponding clusters

The predetermined k value is given and the algorithm takes the unlabelled data set as the input and then categorizes them into different clusters based on the k value and the centroid value which is provided in the following steps. Firstly, the k number of clusters is considered and the data points belonging to the closest clusters are marked in the corresponding cluster. In the iterative steps, the centroid of the cluster is found every time and the data points very close to it falls under the same cluster. This process is carried out until the centroid position stops changing.

The solution for the K-means algorithm is obtained by the following steps. Let us consider a dataset X_n, where n is the number of data points and K-means assigns the n data points into the k clusters, in which each data point is grouped to a cluster with the nearest mean. The number of cluster or centroid values should be given in

prior as the input. This algorithm tries to minimize the squared error function or the total intra-cluster variance as follows.

$$J = \sum_{j=1}^{k} \sum_{j=1}^{n} ||x_i^{(j)} - c_j||^2$$

Where,

- J- Objective function
- $\sum_{j=1}^{k}$ - Summation over the clusters
- $\sum_{i=1}^{n}$ - Summation over the data points
- $x_i^{(j)}$ - data point
- c_j - Centroid for clusters j
- $||x_i^{(j)} - c_j||^2$ - Distance function

Initially, the data points are clustered into K groups, where the no of K points is pre-determined in input. Then, select random K points as the cluster centers and according to the Euclidean distance function assign each data points that are closest to the cluster centers. Once all data points have been grouped into a cluster, using the mean value of all data points, cluster centroids are recalculated in each cluster and the same cluster allocation method will be iterated until a convergence criterion.

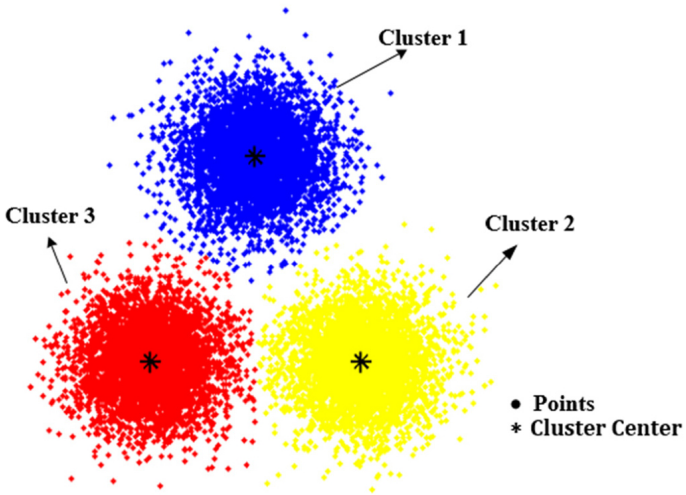


Figure 2.1: Example of K-means Clustering with 3 clusters and centriods in black point [14].

K- means is one of the simplest and most efficient methods for unsupervised machine learning. One of the limitation of this method is determining the number of clusters for the data point as the input. The optimal clusters[1] k lead to the greatest separation (distance). The elbow method is a useful methodology for selecting an optimal number of clusters.

[1]The term "optimal" generally refers to the number of clusters that results in the best clustering performance or the greatest degree of separation between the clusters.

2.2.1.1 Elbow Method

This method is used to run on the dataset for a wide range of k values and for every k value, an average score for all clusters is calculated. By default, the distortion score is computed, which is the sum of square distances from each point to its assigned center. This method helps us in finding the k value without having to guess and try a wide set of values [15]. In the elbow method, the Sum of Squared Errors (SSE) for each data point and the centroid clusters are calculated, and the values are plotted against the k values, where the graph looks like an elbow. As the number of cluster points increases, the SSE value decreases, and the optimal number of cluster value is found at the "elbow" point (i.e,) the point after which the SSE starts decreasing in a linear fashion, as seen in figure 2.2.

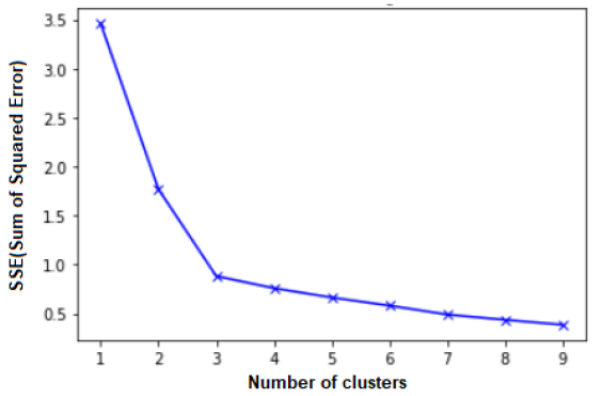
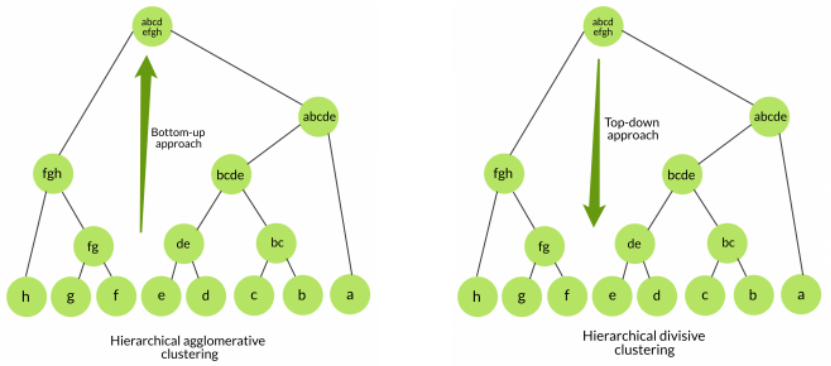


Figure 2.2: Example of elbow method to find optimal number of clusters [15]

2.2.2 Hierarchical Clustering

Unsupervised machine learning refers to the method in which the dataset provided does not contain classified or labelled data points. AI algorithms are used to find patterns in the data points provided. Hierarchical clustering is an unsupervised machine learning method in which a hierarchy of clusters is formed by grouping similar data in the form of trees, which is also called a dendrogram [16]. In this method, the total number of clusters is not pre-determined unlike K-means clustering. The end of the Hierarchical tree is a set of one cluster, and every cluster differs from each other. Hierarchical clustering can be carried out in two ways : Agglomerative and Divisive clustering. In the former method, first, all the data points are considered as single entities or clusters and successively agglomerates all those pairs of clusters until all clusters have merged, and a single cluster is formed at the end, as seen in figure 2.3. This follows a bottom-up approach from each data points to one single cluster.

The Divisive method is opposite of the agglomerative method where it follows a top-down approach. Initially, a single big cluster containing the whole data-set is created, which is split recursively until each data point is clustered into a single cluster.



(a) Agglomerative clustering **(b)** Divisive clustering

Figure 2.3: Agglomerative and Divisive clustering hierarchy

Consider, N data points. Initially the clustering is performed with a distance matrix or cosine similarity matrix which measures the similarity between vectors by calculating the cosine angle between two or more vectors. This distance matrix defines how each data point differs from the other. Once the similarity matrix is created, the shortest linkage criterion between each cluster is formed. The linkage of clusters is done using the Euclidean distance of each data point of every cluster. After some iterations, all the data points are clustered into a single cluster and the process is completed. The final hierarchy for each data point to all the clusters is formed by a dendrogram which is a tree-like structures that shows the similarity between each data, as shown in figure, 2.4.

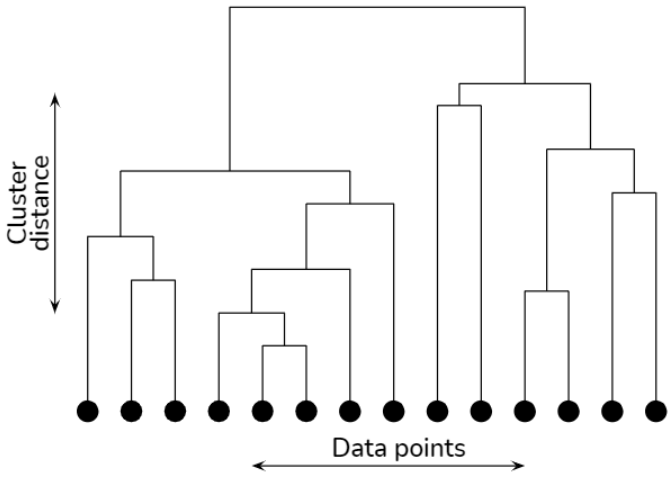


Figure 2.4: Hierarchical clustering dendrogram

The vertical axis represents the distance score of the clusters and the horizontal axis represents the clusters. The vertical lines indicate the dendrogram which represents the clusters and we can see the number of clusters increases as we move from the top to the bottom. The user can determine the clusters by drawing a horizontal cross-tolerance line across the dendrogram's longest vertical line. In this thesis, we use Agglomerative hierarchical clustering. Hierarchical clustering may be a better

approach compared to K means, as the other approach is dependent on us giving the predetermined number of clusters by better observation of the dendrogram.

2.3 Supervised Learning

It is a type of machine learning in which the machines are trained on labeled data i.e the input is mapped to the output, to make the desired predictions. In Supervised Learning, the training of the model is carried out until it is capable of finding the hidden underlying patterns between the input data along with its associated label, such that it is now capable of yielding the right output when it is provided with data which has not been previously used for training purposes. Here we have to find the mapping function which maps the input variable with the output variable. Supervised algorithms are mainly classified into two categories.
1. Classification: Based on the labeled input data that it has been trained upon, the classification algorithm is capable of sorting the inputs into different categories or classes.
2. Regression: The regression algorithm or the model is quite different from the classification model, as they are utilized for creating a numerical or real value relationship between the input and the output [17].

2.3.1 Support Vector Machine(SVM)

A support vector machine is a type of supervised machine learning algorithm which is mainly used for regression and classification purposes by finding the best optimum boundary between the possible outputs. This is dependent on the kernel function which is selected for data transformations, based on which the boundaries between the data points, which are based on the labels or classes mentioned previously are maximized.
Consider the picture below, here the points x, y, and z are the support vectors. The line $x1$ is the separation hyperplane. The SVM algorithm works on finding the line or hyperplane which separates two or more data sets provided, which is of 2 dimensions, by maximizing the separation. The support vectors here are the data points that are at very close distance to the hyperplane.
The finding of the hyperplane or line that separates two or more sets of data points, mostly depends on the kernel function which tells us about the smoothness and the efficiency of the class separation. There are many types of such functions, like the Linear, Polynomial, Gaussian, and Radial Basis functions. Here we can see that the Linear kernel function gave us good results/ accuracy.
SVM is mainly used for binary classification; this can also be utilized in the multi-class classification process, which is nothing but the binary classification with multiple binary classifications.
This has two approaches: the one-to-one approach, which breaks down the multi-class situation into various multiple binary classification problems which is a binary classification for each pair of classes. The one-to-rest approach is setting up a binary classifier for each class. Each SVM would predict membership in one of the distinct classes classes.

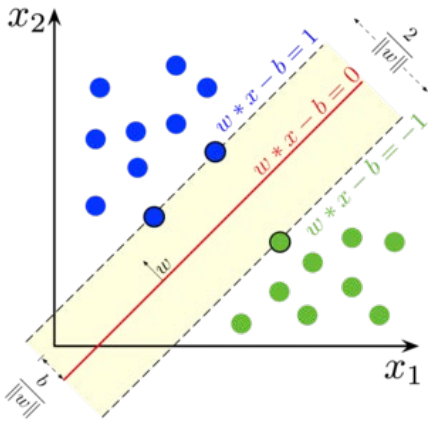


Figure 2.5: Example of Support Vector [18]

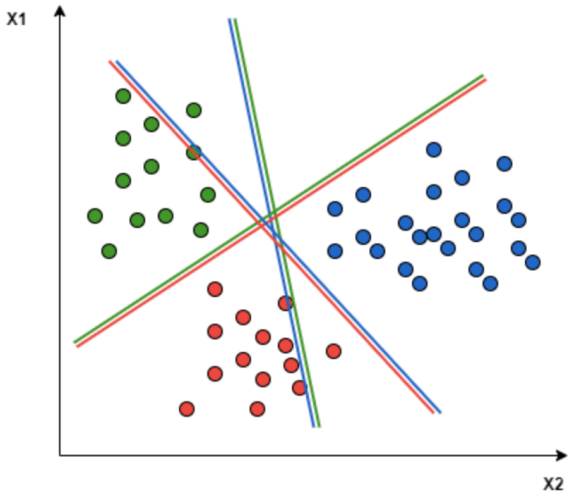


Figure 2.6: Example of One-to-One approach [18]

Let us try to understand the above two approaches a little better.

Consider the picture below with three classes of data points blue, red and green. While using the one-to-one approach, the hyperplane here distinguishes the points between any two classes first without considering the third. Here the green-blue line only concentrates on increasing the distance between the data points of the data groups green and blue and has nothing to do with red.

As per the one-to-rest approach, consider the picture given below, here the hyperplane or the line is considered to separate a class from all the other classes at the same time. for example here the red line separates the red data points and maximizes their distance from all other data points.

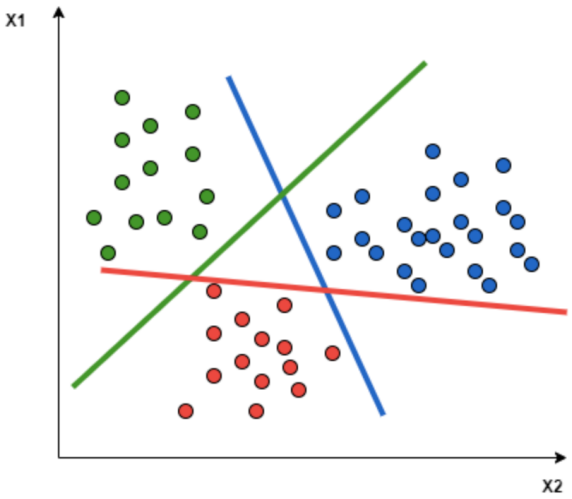


Figure 2.7: Example of One-to-Rest approach [18]

2.3.2 Logistic Regression

When the dependent variable is categorical and can be predicted using a given set of independent variables, the supervised machine learning technique called the logistic regression can be used. In the case of binary classification, the output is either categorical or discrete. i.e. yes or no, 0 or 1, true or false, etc. It gives a probabilistic value between 0 and 1. It is very similar to that of Linear regression, the only difference being that linear regression is used for solving the problems dealing with regression, and Logistic Regression is used for solving the problems dealing with Classification.

An 'S-shaped logistic function is fit instead of a regression line here, which indicates the likelihood of something occurring. It is capable of giving the probabilities and also classifying the required data by making use of continuous and discrete data-sets.

The multi-class classification using Logistic regression is the same as binary classification, instead of a yes or a no, here the input is a set of different features and characteristics, which predicts the label. This can be done by splitting up the multi-class classification problem into multiple binary classification problems by fitting a standard logistic regression model on every sub-problem. Two types of processes can be used to do the same, the one vs all model and the one vs one model.

In the one vs all method, the class that we are trying to predict is given by 1 and all the other classes are given by 0. When we move on and work on the next label we name that to be 1 and the rest of the other labels to be 0 again.

Logistic Regression can be used to classify the observations using different types of data and can easily determine the most effective variables used for the classification [19]. The figure 2.8 is showing the logistic function:

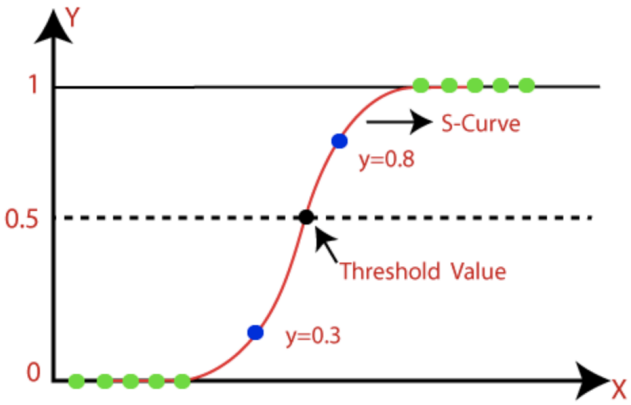


Figure 2.8: Logistic function [19]

2.3.3 Gaussian Naive Bayes

It is the extension of Naive Bayes used for the calculation of the mean and the standard deviation of the training data, which is based on Bayes theorem for classification purposes. In the Naive Bayes algorithm, changing the value of one of the features does not change the value of the others.
According to bayes rule,

$$p(y_i|x_1,\ldots,x_n) = \frac{1}{Z}p(y_i)\prod_{j=1}^{n} p(x_j|y_i) \tag{2.1}$$

where,

- $p(y_i|x_1,\ldots,x_n)$ is the probability of class y_i given input features $x_1,\ldots,x_n$.
- $p(y_i)$ is the prior probability of class y_i.
- $p(x_j|y_i)$ is the probability of feature x_j given class y_i, which is assumed to be normally distributed.
- Z is a normalization constant.

The probability of feature x_j given class y_i is modeled as a Gaussian distribution with mean μ_{ij} and variance σ_{ij}^2:

$$p(x_j|y_i) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}}\exp\left(-\frac{(x_j-\mu_{ij})^2}{2\sigma_{ij}^2}\right) \tag{2.2}$$

where,

- μ_{ij} is the mean of feature x_j for samples in class y_i.
- σ_{ij}^2 is the variance of feature x_j for samples in class y_i.

In the above Gaussian Naive Bayes equation 2.2, for every class of the input values the probability has been calculated through a frequency. The mean and the standard deviation has to be found for the X value of each class for the whole distribution.

$$\mu(x) = (\frac{1}{n})\sum(x)$$

where:

n: number of instances
x: value of the input variable
.

$$\sigma(x)^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu(x))^2$$

The Gaussian probability density function can be used to make predictions by substituting the parameters with the new input value of the variable and as a result, the Gaussian function will give an estimate of the new input value's probability.
The Naive Bayes classifier assumes that the value of one feature is independent of the value of any other feature. Naive Bayes classifiers need training data to estimate the parameters required for classification. Due to their simple design and application, Naive Bayes classifiers can be suitable in many real-life scenarios.

2.4 Semi-supervised Learning

It is a type of machine learning in which the algorithm is trained on a combination of both labeled, which accounts for a very small amount of data and unlabelled data, which consists of most parts of the data. In the first place, the Unsupervised learning algorithm is used to cluster similar data. Later the clustered data is used to label the rest of the data. There are certain assumptions to be made about the data for the semi-supervised approach like

- Continuity Assumption: Points situated closer are said to have the same label.
- Cluster assumption: division of data into distinct clusters is initially made and the points in the same cluster tend to have the same label.
- Manifold Assumption: The data lie approximately on a manifold of a much lower dimension than the input space. This assumption allows the use of distances and densities which are defined on a manifold [20].

2.5 Natural Language Processing

NLP is a branch of AI that deals with bridging the machine's understanding of humans in their Natural Language. Natural Language can be in form of text or sound data, which is used by humans to communicate with each other. NLP can enable humans to communicate with machines in a natural way [21]. NLP in software engineering has received more attention as most of the software logs consist of text data, and NLP with the use of deep learning techniques has been able to create a better representation of data for the classification of faults or error analysis.

2.5.1 Term Frequency and Inverse Document Frequency

Term Frequency-Inverse Document Frequency (TF-IDF) is a statistical weight measure method that is often used in information retrieval and text mining. This method is used to evaluate how relevant words appear in the document in the collection of

the whole document or corpus. TF-IDF is also called the weighted aggregation method as it is done by multiplying TF and IDF metrics.
This TF-IDF weight increases proportionally to the number of times a word appears in the document, along with the frequency of that word in the document. The weighted aggregation method is used here and although the number of words in every log data might differ, it is converted into useful data such that all such words in every log file is converted into a constant dimension. The weight of the TF-IDF is used to predict the significance of certain words in a sentence. If certain words appear too many times in a file, it predicts that the significance of such a word is higher and this can be obtained by using the Term frequency,

- t — term (word)
- d — document (set of words)
- *corpus* — the total document set

$$TF(\text{word repeated}) = \frac{\text{count of } t \text{ in } d}{\text{number of words in } d}$$

If a particular word seems to be appearing in every other log file, then such words seems to hold very low significant importance and therefore it's weight can be reduced along with its importance. The weight determining procedure can be obtained by calculating the Inverse Document Frequency (IDF),

$$IDF(t) = log\frac{N}{df(t)+1}$$

- N = count of corpus or the total number of log files taken into consideration
- $df(t)$ = occurrence of t in N documents

The TF-IDF gives the weight for every word in corpus which can be described beifly in section 3.5.1

2.5.2 Word2Vec

This is a machine learning process which is similar to other methods such as classification, regression, and prediction. Word2Vec is a machine learning method that provides Word Embedding given a particular word. It provides a numerical vector for the word using a neural network developed by Tomas Mikolov [5]. These numerical vectors can be easily used for mathematical operations like semantic similarity, relation with other words, etc.
Word2Vec is an unsupervised machine learning method consisting of 2 shallow layers of neural network which is obtained by two Neural Networks (NN) methods: Continuous Bag Of Words (CBOW) or the Skip-gram models which can be described below for the creation of the word embedding. Another objective for creating a Word2Vec is to create a word embedding where the words that have similar meaning or context to other words should be closer in the WE space than the different words.

Word2Vec is one of the game changers in the field of NLP for word embedding although it has a few limitations. Word2Vec cannot handle/embed the words that are not seen during the training of the ML model. In other words, Word2Vec can't treat Out of Vocabulary (OOV) Words i.e., unknown words that are present in the corpus but not in the vocabulary.

2.5.3 N-Grams

N-Grams are used in the natural language processing and text mining tasks. N-Grams are continuous sequences of n items or words from a text or corpus. Technically, N-Grams can be defined as continuous neighbor adjacent sequences of words in a sentence. The dimension of N-Grams increases exponentially as the n(Number of words) value increases. This method is used to get a better meaning or context of the words in a corpus. A simple example of the word 2 grams are

$$\text{My name is Markus} \rightarrow < My, name >, < name, is >, < is, Markus >$$

The N-Grams are also applicable for the character-based sequences in the word. An example for 2-grams character based sequence is

$$\text{example} \rightarrow < ex >, < xa >, < am >, < mp >, < pl >, < le >$$

2.5.4 FastText

FastText is a popular machine learning based word embedding and it is created by FaceBook as an open source tool for commercial use [22]. FastText is used for various applications, like the text classification, word embedding, and text representation. This FastText method seems to give better results for words that are Out of Vocabulary (OOV) with better numerical representation. To solve the problem, FastText uses the character n-grams instead of the word n-grams. FastText extracts all character n-grams with $3 \leq n \leq 6$ as seen in figure 2.9.

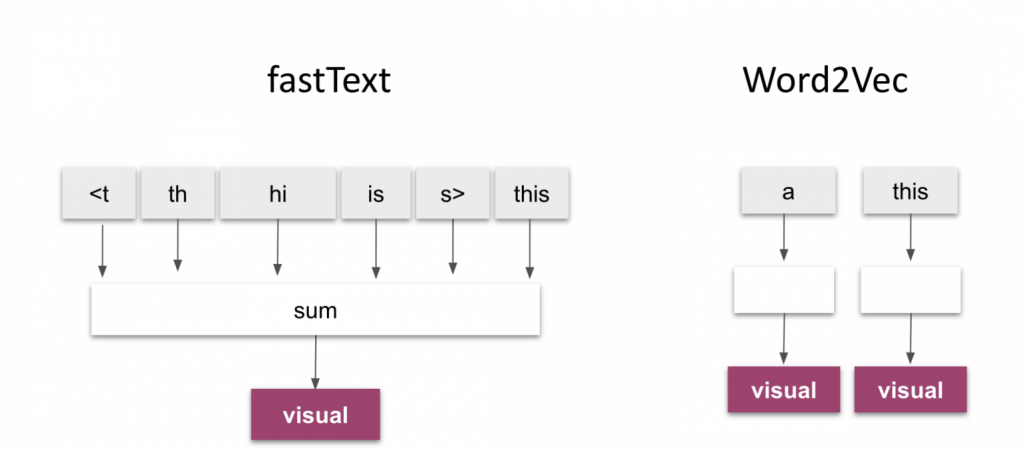


Figure 2.9: FastText vs. Word2vec

Word2Vec uses either the Continuous Bag of Words (CBOW) or Skip-Gram model for the word representation, but the FastText model uses both the models for the better numerical representation of each word. This makes similar words to be closer in the numerical space for better representation of their meaning and context.
Since FastText is an open source model, there are a lot of pre-trained models available for research purpose, which is trained on 157 different languages and also on different corpus [22]. Each word has been embedded for 300 dimensions and the number of words that has been used for the training differs for different languages. The output dimensions from the FastText model can be reduced to lower dimensions using the dimensions reducing techniques which can be discussed further.

2.5.5 Continuous Bag of Words (CBOW)

Continuous Bag of Words (CBOW) is an unsupervised machine learning method used for the word embedding of the given corpus or data-set. CBOW is used to predict the vector for the current word or target word, based on the surrounding words of its context as seen in session 2.11. Therefore, the output vector of the word depends on the surrounding words. The CBOW architecture looks the same as a deep learning model architecture, as it contains the input layer, hidden layer, and the output layer. The dimensions of the hidden and the output layer remains the same, but the input layer differs based on the surrounding words that the model has utilised for training [23].

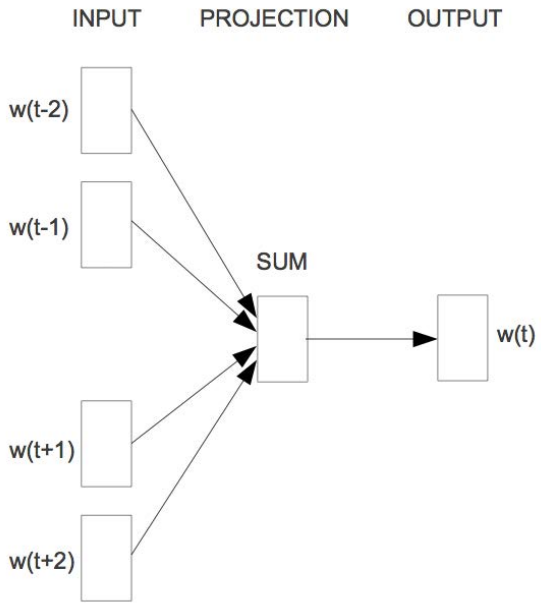


Figure 2.10: CBOW Block Diagram [23]

For example, if the context for one target word is 4 words, then the input layer will have the dimension of $1 \times V$ where V is the number of input words, which is 4 here. For each input word, a vector is predicted and the output from the hidden layer is $V \times E$, where E is the dimension of the output vector. Finally, the average of the 4 $1 \times E$ is taken and one output vector is received from the final layer by

using softmax function as seen in figure 2.10. Generally, for any machine learning model, the label has to be provided for training, unlike in this where the CBOW is trained based on words in the surrounding, by considering the context words as label. CBOW model trains faster than all other word2Vec models and it also gives a better representation of the rare words based on its surrounding words with better accuracy. The application of the CBOW can be seen on social media apps, where while typing a certain word, the next word is predicted based on the previous words used.

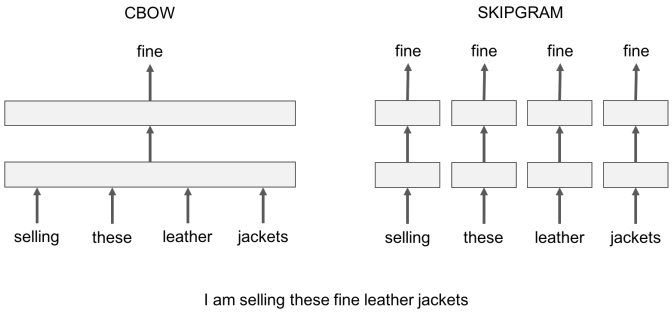


Figure 2.11: An example of the input and output of CBOW and Skipgram and the rectangular box represents NN layers

2.5.6 Skip-Gram

Skip-Gram is another word embedding method, which is the inverse of the CBOW model. It predicts the context when a word is given [23]. In the CBOW model, based on the surrounding words, the model tries to predict the vector for the target word. Whereas in the Skip-Gram model, given a word or middle word, it tries to predict the surrounding or context words as seen in figure 2.12. By this method, the model requires lesser training data for the word embedding purpose and it usually works well for rarely used words.

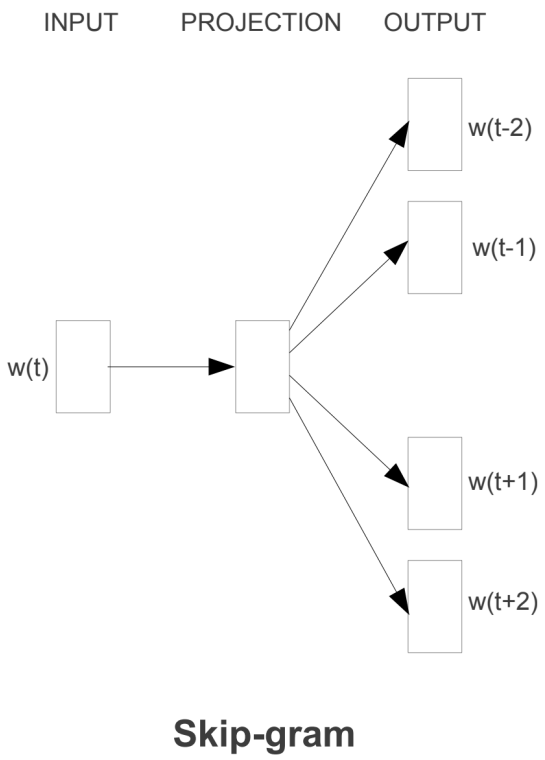


Figure 2.12: Skip-gram Block Diagram [23]

As seen in the CBOW model, the input to the input layer in the Skip-Gram model will be $1 \times V$ where V is the number of words in the vocabulary or context words. Next, the hidden layer dimension will be $V \times E$ where E, is the dimension of the vector. The output from the single hidden layer will be $1 \times E$ and the final output from the final layer which consist of softmax function will have dimension $1 \times V$, where each value in the vector is the probability score of being the target word at that position.

2.6 Reducing Dimension

2.6.1 PCA

Principal Component Analysis (PCA) is a linear algebra technique used in machine learning to reduce the dimensions such that the data is prepared to create a projection of the dataset into a new subspace before it fits into a model. This reduces the input variables which in turn makes the model has a better performance in the future for predictions on new data.
In the figure 2.13, the principal components PC1 and PC2 are the orthogonal axes in the direction of maximum variance given that the new feature axes are orthogonal to each other. Here the original feature axes were $x1$ and $x2$.
For dimensionality reduction, a $d \times k$–dimensional transformation matrix W has been constructed that maps a sample vector x onto a new k–dimensional feature

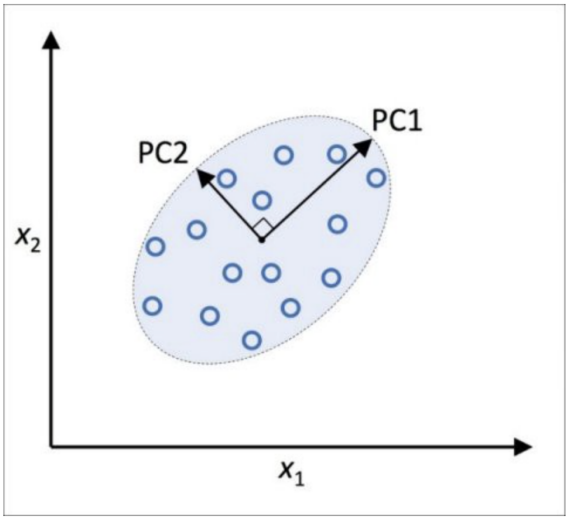


Figure 2.13: PCA [24]

subspace with lesser dimensions than the previous d–dimensional feature space.

$$\mathbf{x} = [x_1, x_2, ..., x_d], \mathbf{x} \in \mathbb{R}^d \tag{2.3}$$

$$\mathbf{z} = [z_1, z_2, ..., z_k], \mathbf{z} \in \mathbb{R}^k \tag{2.4}$$

In PCA for dimensionality reduction equation in 2.3 and 2.4, as k is smaller when compared to d, the largest possible variance is taken by the first principal component. If the other remaining principal components are uncorrelated, then they too will have the same variance.
If the features were initially measured on a different scale or they were not given equal importance, needs to be changed and standardized before PCA, as the PCA directions are sensitive to data scaling [24]. t-distributed stochastic neighbor embedding, is a nonlinear dimensionality reduction technique that is applied to the data produced by the PCA method to reduce it to 2 dimensions for data visualization purposes. This machine learning algorithm ends up producing a slightly different output every time, as it focuses on retaining the structure of the neighboring points. There are two main steps that describes the working of t-SNE:
1. For the pairs with higher dimensions, t-SNE constructs a probability distribution, due to which a higher probability is given to the objects which are similar and a lower probability is assigned to the ones which are not.
2. Step 1 is then again applied on the lower dimensions iteratively till it minimizes the Kullback-Leibler divergence citetSNE.

2.7 Evaluation

After the machine learning models have been applied, a metric check needs to be done to verify if the algorithm that we have applied is the right one and is giving good accuracy. For this purpose evaluation metrics like the F1-score, which is a machine, and learning metrics are applied to the classification model.

Previously up-sampling was performed to make sure all the classes in the data have the same amount of data. For finding the accuracy of this data, given we have classes with the same amount of data now, the following steps are performed. Accuracy is given by the total number of correct predictions for the total number of predictions that are done, which shows that if among the total number of predictions made, if 80 percent of them are correct, then the accuracy is 80%.
Even without up-sampling of data, when the data is still imbalanced, the F1 score could be used to see how accurate the classification model is. The F1 score is mainly dependent on the precision and the recall scores by combining both of them into a single metric.

$$Precision = \frac{TP}{TP + FP} \tag{2.5}$$

where:

TP is total number of true positives
FP is total number of false positives

The precision metric shows the percentage of actual positive or correct output among all the ones that have been declared as true or positive.
If a model is not precise, it may classify many values as positive even when they are not, as their selection method is very noisy. And on the other hand if a model is precise, anything that is classified as positive by this model is always likely to be positive.

$$Recall = \frac{TP}{TP + FN} \tag{2.6}$$

where:

TP is total number of true positives
FN is total number of false negatives

The Recall metric implies that, within everything that is positive, how many did the model succeed in finding. A high recall value of a model shows that the model has succeeded in finding almost all the positive data, and while doing this, some of the negative data or cases may also be wrongly classified into the positive case category. But with a model with a low recall value, many of the positive case data may not be identified.

$$F1score = \frac{2(Precision * Recall)}{Precision + Recall} \tag{2.7}$$

F1-score is the harmonic mean of the Precision from equation 2.5 and Recall from equation 2.6, combines them both to give a single metric. The F1 score lies between 0 and 1. The higher the value or closer it is to 1, the model is said to have a higher accuracy [25].

3
Methods

3.1 Research Environment

Windows 10 64-bit operating system with Intel(R) Core(TM) i5-8365U CPU @ 1.60GHz 1.90 GHz has been used for the purpose of this project. Several tools such as Jupyter notebook for running applications in python programming language, along with Microsoft Excel and Power Bi for the data visualization has helped us complete the process successfully. Within python, we have imported several python moduli like regular expressions, Numpy, Pandas, Scikit-learn, matplotlib, seaborn, and Keras. For word embedding, python libraries such as Gensim, nltk, and spacy has been used to import the pre-trained models.

3.2 Data Collection

The EDCD team at Volvo Car Corporation which is working with the integration of software components in an AUTOSAR environment, has provided log data which is obtained by the website called 'Victoria'. This website is used by the CD team, along with many other teams, to keep a check on the jobs that have been running along with their statuses and error logs. The error logs provided mostly deals with errors and warnings generated while configuring, integrating, compiling, and generating ECU software. The log data provided contains unique information such as the project name, project title, time, and fault or warning/ error messages and also hundreds to thousands of lines of data leading to this error along with the ones which has nothing to do with the error itself. Since there are various kinds of errors such as configurations, compiler, linker, calling frequency mismatch, ECU memory overflow, etc, the initial idea is Data understanding i.e., to focus on finding the most common types of errors that are difficult to discover and solve as they have bigger recorded logs compared to the other errors. Other than the healthy logs and error data, the data-set also records the working conditions of the systems while it is up and running, such as system environment status, paths of the file directory, etc.

3.3 Data Understanding and Preparation

As mentioned previously, the log data provided is a result of the continuous deployment and the changes in software development and various other tools being used, which gives rise to multiple interface issues while integrating different software ap-

plication layers with the base software. The errors are stored in the form of build log data which contains all the information about software integration in various steps and also about the cause and the multiple kinds of errors. As the error log data also contains a lot of unwanted data which is not required for the initial processes, we have to filter it out and pick just the lines with errors in it. Every failed log data file seems to have a few lines which tells us exactly how the log file happened to fail. To pick out this kind of information which is important, preprocessing the log data which is mentioned in the further steps by using certain keywords can be performed, such that it picks the information bearing necessary data.

Information like the machine log values, date and time, URLs, etc are the kind of data that will not help in the recognition of errors or the clustering process. Therefore such data can be dropped in the earlier processes itself. After the initial process, it has to be made sure that the data distribution is normal and the outliers with information which has no meaning to it, has been removed. To perform data preprocessing, the Python module regular expression that helps in matching or finding the special sequence of characters or sets of strings has been used.

As this is the primary process which is also very tedious, for further use, data pre-processing and clustering of data has to be performed, which analyzes the data-set before diving into the model for the machine learning purpose. The data preprocessing contributes to almost 70 percent of the total work done until now.

3.4 Data Pipeline

3.4.1 Data Cleaning

In the earlier stages, the first step to be followed before starting anything else is the Data cleaning process. By cleaning the data we decide what's important and what's not, which gives us a good idea about the data in this stage. From the hundreds and thousands of files and lines of data, here we make a call as to which needs to be used in the future for the actual machine learning purpose. As the data is vast with a lot of irrelevant information, the most tedious process which also seems to be an important step, is to understand the kind of data we are dealing with and pick the kind of lines in the data-set which gives us an insight into the errors.

The log data provided seems to contain a lot of unwanted information, which is also very unstructured and would hinder the analysis of the failed data in the future steps. Therefore it needs to be parsed to extract the log events only, which are helpful for the future analysis. The filtered log data seems to give good accuracy and efficiency in the future. For example, here the entire data set is converted into lower case and the words that add no meaning to the context of the data is removed. Links and punctuation are removed also with the unwanted numbers.

3.4.2 Data Pre-processing

Raw or unstructured data needs to be formatted to a structure that can be easily understood by the machine for further analysis. This method is called Data pre-processing.
Basic operations like removing multiple spaces and converting them to a single space, removing characters like punctuation, expressions, etc are also done here. By carrying out the above-mentioned processes, the data-set and the model are made robust for future use. In the data-set provided, the actual places of occurrence of the errors is initially picked by filtering out the lines which consists of negative words such as $'fatal'$, $'cannot'$, $'not'$, $'error'$, etc. Sometimes in the log data, the error which was supposed to be a single line between brackets, has been split into multiple lines. And as per the algorithm if such lines are picked from the data-set with the use of negative words, then the line which was supposed to be picked as a single line, may be divided into multiple lines or some lines may even be ignored, therefore reducing the accuracy and efficiency. Therefore it has to be confirmed that the data which is supposed to be in a single line is taken as a single line [1].

Algorithm 1 Data preprocess filtering

Ensure: $important = []$
Ensure: $important2 = []$
Require: keep phrases = [”*error*”, ”*cannot*”, ”*errors*”]
Require: delete phrases = ”*warning*”

```
for line in sentence do
    for phrase in keep phrases do
        if delete phrases not in line then
            if phrase in line then
                important.append(line)
            end if
        end if
    end for
end for
```

Here the non character tokens i.e delimiters, operators, punctuation and numbers are removed from the log data given, as it does not really help with the analysis or robustness of the model. The log data given is said to have been comprised of many sentences or lines. This process can be done by using the Python regular expression module [26].

3.5 Word Embedding

The Word Embeddingsf for cleaned and pre-processed log data can be created by using the FastText model that has been discussed in the section 2.5.4. One of the advantages of using the FastText model is that, it provides even OOV word embedding vectors. Since the cleaned log data contains various rare words or words

which are related to the software abbreviations and cannot be understood easily, needs to be embedded for such words using the FastText model. Therefore, the FastText model needs to be trained with the corpus containing the above words. The pre-trained FastText is downloaded from the python gensim module [27] and this pre-trained model is trained with a large corpus of words that is present in the log data. Further, the model is trained with FastText's CBOW model as it represents words which are rare when compared to the Skip-gram models.

Parameter	Value
vector size	300
window	100
min count	5
sample	1e-2
workers	4.00
sg	0
epochs	200
cbow mean	1

Table 3.1: Fast text model training parameter

Where, word_tokens represent the corpus to train the model, vector size is the dimensionality of the word vectors, and Window size is the maximum window or window size between current and predicted words within that sentence. Further, the model has been trained for 200 epochs. The total time taken to train this model is around 30 minutes. Once the model is trained, it can be stored in the Word2Vec format, which can be used for semantic vectorization purpose.

3.5.1 Semantic Vectorization

All the pre-processed log lines are converted into a python list format, which is further split into separate tokens. Once the tokens are separated, the Semantic Vectorization can be performed on each log line. Semantic Vectorization involves extracting semantic information from the log lines and converting them to the semantic vector of fixed dimensions. Here, the semantic information represents the lines that have similar meaning with few changes when it comes to the words which have identical or almost similar word embedding vector with less cosine similarity. This is done by Aggregation based on the TF-IDF method with FastText model as referred in paper [1].
To capture the semantic information of the log event, the Semantic Vectorization steps seen in the figure 3.1 can be followed. Initially, all the words in the corpus is aggregated in a dictionary, and for every word, the weight aggregation based on the TF-IDF in section 3.5.1 can be obtained [28], which is used for the information retrieval. This information retrieval using TF-IDF determines how important each word is in a sentence along with the whole corpus.

For example, the word *Fail* can appears several times in the log sentence, which signifies that the word needs more attention than all other words, as it indicates a

certain event in the log sentence which might have been an error. Therefore, the TF weight for this word can be calculated using

$$TF(word) = \frac{\#word}{\#total} \tag{3.1}$$

where:

$\#word$ indicates the total number of times the words appears in the log sentence
$\#total$ indicates the total number of words in the log sentence

In the similar manner, the IDF score will be calculated to determine the importance of the word when all the log sentences are considered, along with the occurrence of the same word in every sentences. This can be obtained by using the logarithm of

$$IDF(word) = log(\frac{\#N}{\#N_{word}}) \tag{3.2}$$

where:
N indicates the total number of log sentence
N_{word} indicates the number of a sentences containing a particular word

The TF-IDF score for all the words in a sentence will be calculated in the above mentioned manner. The word embedding vector for each word is calculated using the FastText model which has a dimension of 300.

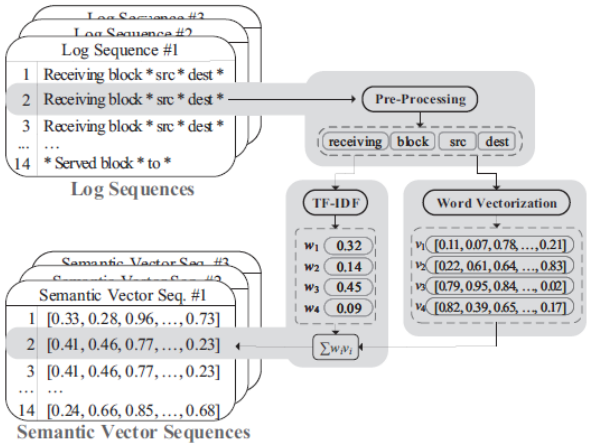


Figure 3.1: Robust semantic vectorization [1]

Finally, the semantic vector for each log event can be obtained by summing up all the word vectors in the log sentence and multiplying TF-IDF with the FastText vector as shown in the equation

$$V = \frac{1}{N}\sum_{n=1}^{N} \omega_n \cdot v_n \tag{3.3}$$

Where,

ω_n indicates word vector for word embedding
v_n indicates vector of TF-IDF

The final vector V represent the semantic vector for a log sentence, which is able to semantically identify a similar log sentence based on the cosine similarity. This method is able to handle instability in the log data.

3.6 Feature Analysis Using PCA

Once the word embedding vector is obtained from FastText and TF-IDF, the output of the word embedding will be a 300 dimension vector, where each vector represents certain features of the sentence. Visualization of the log data using a 300 dimension vector is difficult to analyze and plot. Therefore, the PCA method discussed in the section 2.6.1 can be followed. The matrix dimension of the PCA is going to be the same as the input dimension, for example, in our case, the dimension of the PCA matrix will be 300, where it will be described as 300 principle Components (PCs). But, all the 300 PCs are not informative. So, the number of PCs should be reduced for a useful examination. Further, the number of PCs that is required for a certain data-set can be determined using the methods mentioned

- Eigenvalue criterion
- Proportion of variance explained criterion
- Scree plot criterion

In this thesis, the Scree plot criterion will be followed to determine the number of PCs. The Scree plot criterion is the graphical representations for determining the number of PCs to be retained. This plot shows the eigenvalues for each PC. The plot looks like a downwards curve as seen in figure 3.2, The vertical axis represents the eigenvalues, and the horizontal axis represents the number of factors or components. The number of components can be selected by looking at a point where the plot highly flattens at one point from the beginning. In the example plot 3.2 at factor 2, the line gets flattened and that determines the number of PCs which needs to be retained.

3.7 Cosine Similarity

Cosine Similarity is a mathematical metric used to measure the similarity between 2 or more vectors of the same size. Mathematically, it measures the cosine of the angle between two or more vectors in a multi-dimensional space as seen in the figure 3.3. Cosine Similarity is obtained for the outputs using the dimensionality reduction method. In the PCA method, cosine similarity can be obtained for 2 or more PCs. The log lines which are quite similar seems to have an almost similar vector and the cosine similarity value for such lines is 1. The lines which are not similar seem to have a cosine similarity value of 0.
Further mathematically, the cosine angle of two vectors can be obtained using the equation 3.4.

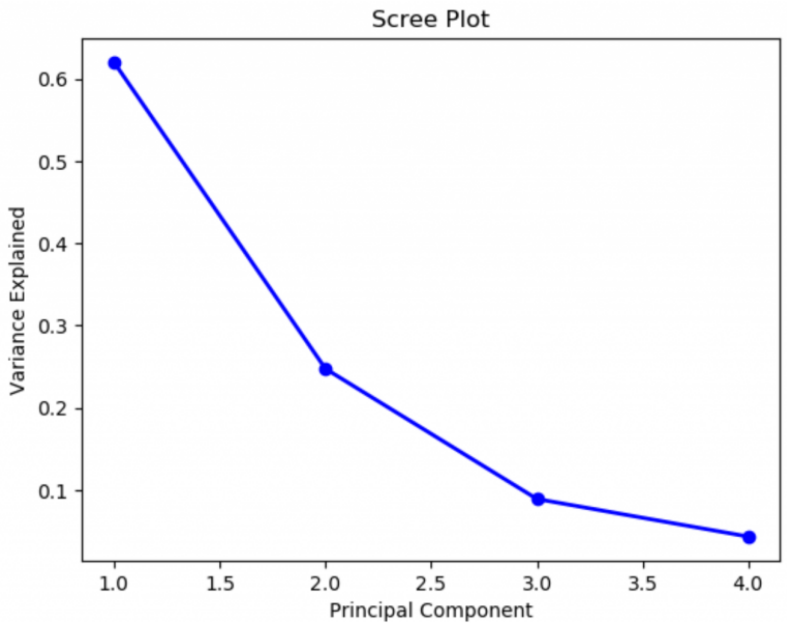


Figure 3.2: Scree Plot for the PCA [29]

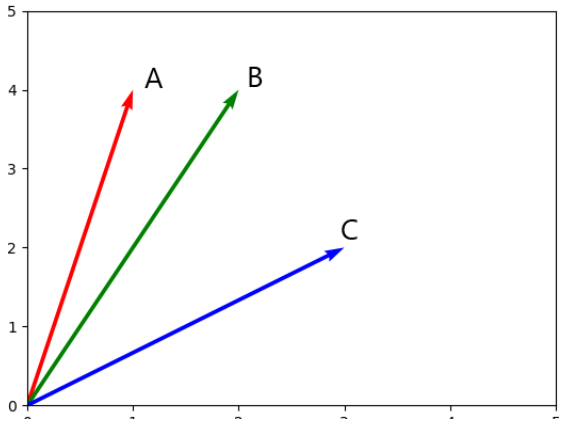


Figure 3.3: Cosine Similarity between vectors

$$cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||} \tag{3.4}$$

x and y represent two different text lines
The output from the cosine similarity is used for clustering and data labeling.

3.8 Data Labeling

Data labeling is an important part of this thesis, especially for the classification of error and normal data. Well labeled data must be provided as input to the machine learning models. Since we are dealing with the unlabelled data from log files, labeling every line in the log is a time consuming process. The number of categories of the labels will differ from the number of clusters formed using the clustering methods

like K-means and hierarchical clustering, as discussed in the section 2.2. Based on the results and observations from the clustering methods, the number of clusters can be determined for the given data-set from the scree plot as discussed in the section . We can use both K-means and hierarchical model to train the clustering methods on the data-set. The prediction method can be used to label the entire data-set. Once the labels have been determined, the number of mislabels can be compared manually with the log lines. Here, we merge the labels from both clustering and log data and check to see the one which produced accurate labeling, by sorting out the labels with log lines. This is discussed briefly in the results section.

3.9 Data Sampling

Once all the data-sets have been labeled using the clustering methods, if the number of data samples in each category of the label varies, it leads to an unbalanced data-set. An unbalanced data-set is the one in which the target variable have a bigger data count than any other classes of variables. Training a machine learning model with an unbalanced data-set will lead to a miserable effect on the results, which makes the algorithm prone to overfit or underfit to the majority class. For example, if there are 3 target class that has to be trained with the proportion of 70% for class 1 and 20% for class 2 and 10% for class 3. In this condition, the model will be overfitting, as it has been trained with more number of class 1 samples than the other 2 classes, and the output is likely to be class 1. Which means that the model is biased towards class 1. To solve this problem, several methods can be followed to make the class distribution equal.

3.9.1 Undersampling and Oversampling

The idea of sampling is to make equal distribution of data samples. In undersampling, the concept is to reduce the ratio of the majority class to the count of the minority class as seen in figure . By doing this, there is a chance of reducing or losing the information of the data-set. For example, if there are n samples in class 1 and m samples in class 2, using the undersampling technique, the total count of class 1 can be reduced to m to match with the count of minority class 2.

Figure 3.4: Undersampling vs Oversampling

Oversampling is the inverse of undersampling. It increases the ratio of the minority class to the count of the majority class as seen in figure . By doing this, synthetic data samples can be created using the existing data samples. This helps in producing more amount of data for the purpose of training the machine learning models. The more the amount of data samples, the more the accuracy in the prediction of unseen samples. By following this method, the information in the dataset cannot be lost. There are several methods to do upsampling and few of the algorithms include Synthetic Minority Over-sampling Technique (SMOTE) or Modified Synthetic Minority Over-sampling Technique (MSMOTE), Variational Autoencoders (VAE). We will be using the SMOTE algorithm in this case.

3.9.2 SMOTE

SMOTE (Synthetic Minority Over-sampling Technique) works by creating synthetic data samples for the minority class samples to match up with the majority class sample's weight. By doing so, random duplicated data samples can be created from the existing samples, although no useful information is added in the oversampling method itself. New synthetic data samples can be created using the existing samples rather than duplicating them. This kind of data augmentation for the minority class can be done in SMOTE.

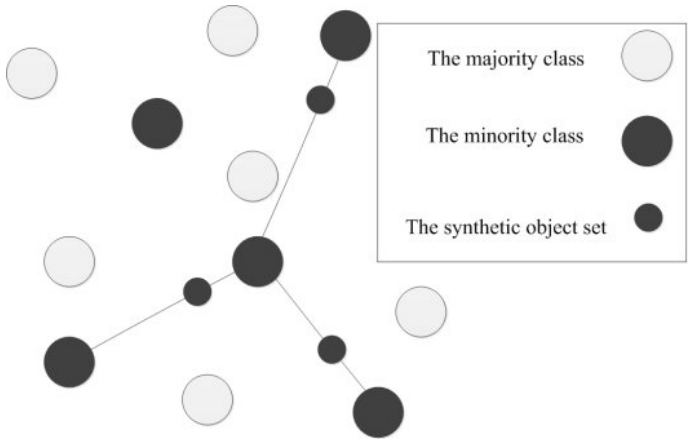


Figure 3.5: The diagram of SMOTE algorithm [30]

SMOTE works by drawing the minority samples in feature space and selecting some random minority samples that is close in feature space. A line connecting the random samples are drawn and we see that there are multiple lines connecting all the points. A new sample or point is drawn in-between every line to synthesize multiple number of samples as seen in the figure 3.5 which depends on the requirements. This selection of points is done by performing k nearest neighbors methods for new points. This is how SMOTE generate new points to the level of majority points.

3.10 Data Normalization

Normalization is the method used for making sure the values stay within the range of 0 and 1. From the data available, normalization can be carried out only when the minimum and the maximum values can be somehow obtained. Normalization value is obtained by subtracting it by minimum value and dividing by the value that is obtained by subtracting the minimum value from the maximum value.
A value is normalized as follows:

$$y = \frac{(x - x_{min})}{(x_{max} - x_{min})} \tag{3.5}$$

where,

- x_{min} and x_{max} are the minimum and maximum values pertain to the value x being normalized.

Before carrying out normalization, the value x has to be made sure to stay within the range of the minimum and the maximum value obtained. Otherwise, the normalized value obtained is not going to be within the range that is required. To make sure such mishaps do not happen, we can ignore such values from the data-set.
MinMaxScaler is a predefined object in scikit-learn which can be used to find the Normalization value by importing the libraries.
Certain practices which has to be followed while using the MinMaxScaler:

1. By using the fit function, the min and the max value can be obtained from the training data available.
2. Use the transform function to train the model with the data that has been normalized.
3. The Normalized scale can be used on any data in the future such that any data-set on which the predictions has to be performed falls under this data scale.

Instead of 0 and 1 as the range for the MinMaxScaler, the range can be mentioned in the argument 'feature range', which helps in giving the desired values.

3.11 Model Selection and Training

Finally, the data that has been pre-processed and converted into word embeddings, are labelled using the clustering methods. After clustering, the labelled data is used for the training of themachine learning models. Further, the labelled data is over-sampled and normalized using the SMOTE and Min-Max scalar methods as discussed in sections 3.9.2 and 3.10. Now, the pre-processed data can be used for the machine learning purposes. In this thesis, we have considered different machine learning models, like the Support vector machine, Logistic regressions and Gaussian Naives Bayes as discussed in the section 2.3.
Further, to improve the accuracy of the machine learning model for better classification, few modifications need to be perform with respect to the hyper parameters of the algorithm. For example, in Logistic regression model

Parameter	Value
njobs	1
C	1e6

Table 3.2: Logistic Regression model training parameter

Once the parameters has been defined, the model can trained with the following code

```
from sklearn.linear_model import LogisticRegression
LR_model = LogisticRegression(njobs, C)
LR_model.fit(x_train,y_train)
```

here, C is known as a Regularization hyper-parameter and it will penalize the extreme parameter for the training data. High C values shows that the model gives more weight to the training data such that it captures the real world information and thus the training data is very important. Further, the *njob* indicates the number of CPU cores used while training. The number of iterations can also be increased while training, unless the accuracy does not converge to an optimal one.

4

Results

4.1 Data Pre-processing

Initially, we loaded the failed logs from one of the above-mentioned processes in the methodology which consists of 57 thousand + lines among the 200 such files. After cleaning the data by picking only the particular lines with the negative words again, as mentioned in the pre-processing section of the methodology, 963 lines are captured. These 963 lines are mostly assumed to be the lines which gives the actual reason the error occurred in the uploaded set of failed log data. Among these lines, the lines which consists of only warnings are ignored.

4.2 PCA

As mentioned in the section 3.6 of methodology, PCA is a method used here to reduce the number of dimensions which in this case was 300 after the word embedding was carried out, on the data which was previously pre-processed.

The PCs can be obtained by analyzing the Scree plot given below. The downward curve in figure 4.1 can be seen to be flattening between 3 and 4 on the X axis, which represents the value of the components.

4.3 Clustering

As the data-set is very huge and may consist of thousands of files with hundreds and thousands of lines which leads to the error, manually labeling such data is a very tedious process and is also impossible as it requires hours of manual work. Therefore in this project, as mentioned in section 2.2, we make use of two methods of clustering which helps us in labeling the errors automatically.

1. In the Hierarchical clustering, when performed on the data which had previously undergone PCA to reduce the dimensions, we can see that at least two clusters are formed at the top. Further deeper, we can see there are more clusters formed. To confirm this and also get a better understanding of the different types of clusters, we make use of another clustering algorithm like the k means in this case.

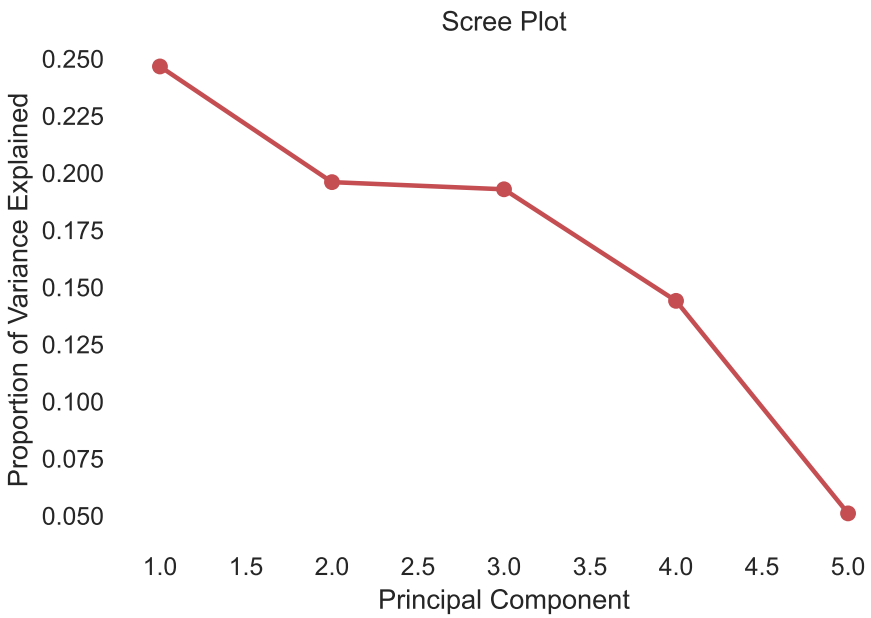


Figure 4.1: Scree plot for Principal component analysis

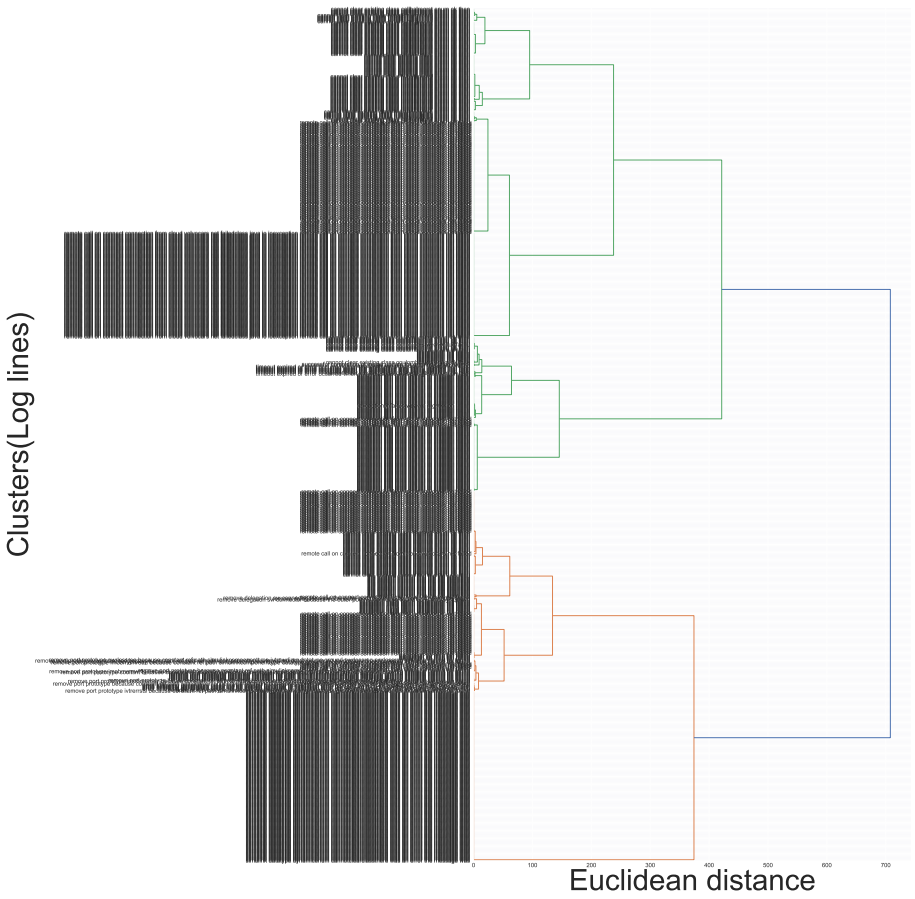


Figure 4.2: Hierarchical clustering for log lines

2. K-means clustering algorithm, when performed on the same PCA data, as seen

in the figure 4.3, shows that the number of clusters seems to vary from 9 to 11, as the curve flattens at these numbers. From the figure 4.4,4.5 and 4.6, we can see the 3 different K-means clusters formed at 9,10 and 11. After getting some manual help from the team to check if the clusters formed were right, we decided that there were 9 clusters in total for now.

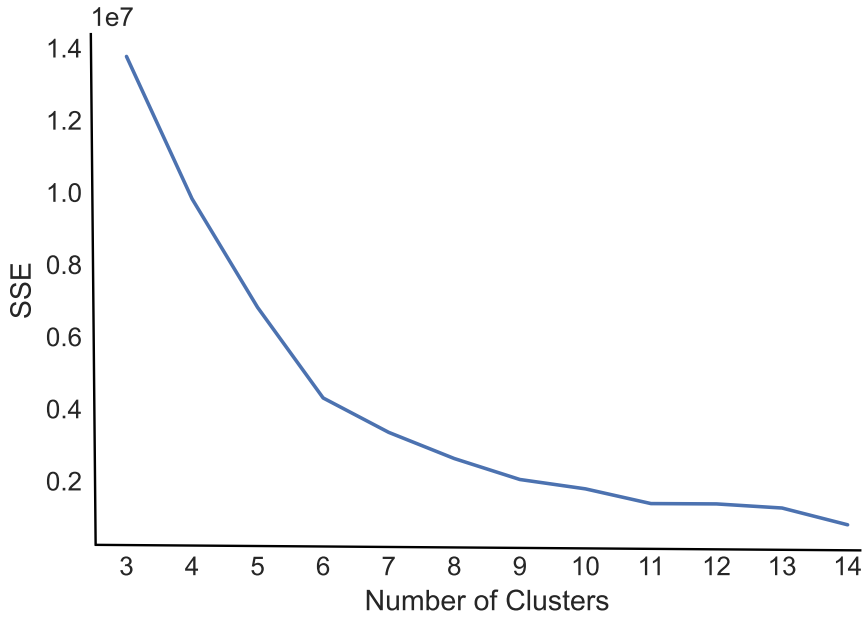


Figure 4.3: K-means Clustering

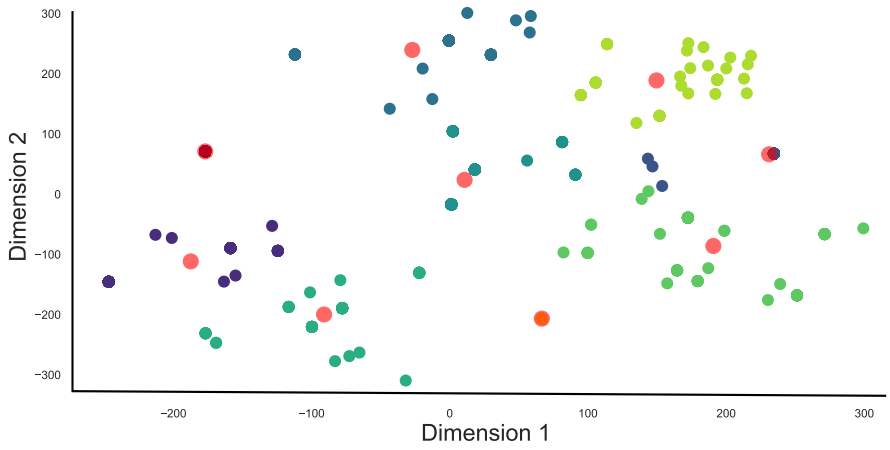


Figure 4.4: K-means with No.of clusters-9

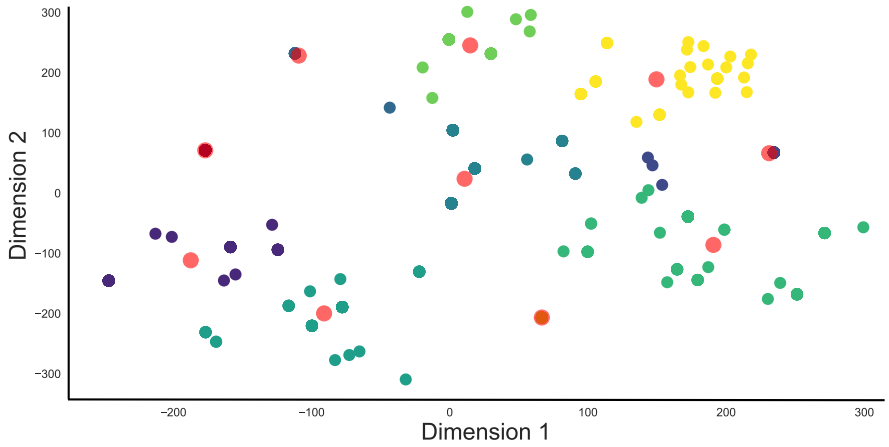


Figure 4.5: K-means with No.of clusters-10

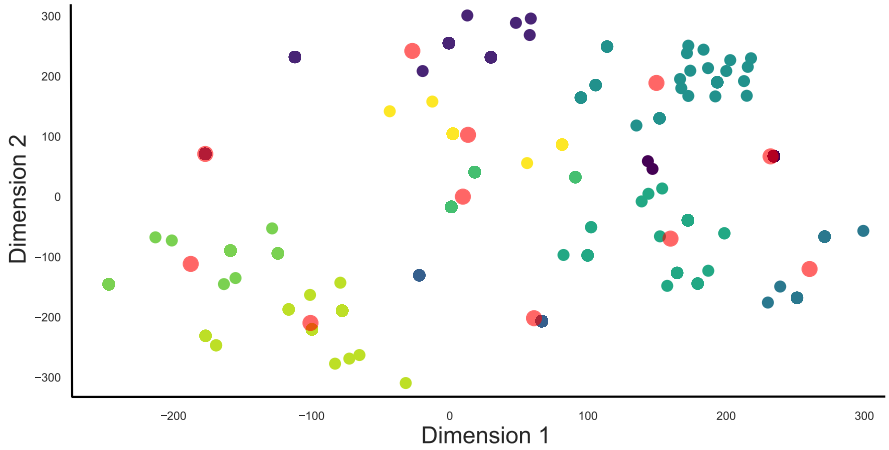


Figure 4.6: K-means with No.of clusters-11

In the above pictures 4.4, 4.5,4.6, X-axis and Y-axis represent the two dimensions difference of Euclidean distance between different log lines.

4.4 Data Labeling

From the clusters obtained by the previous k-means and hierarchical clustering processes, we need to label the errors which fall under every cluster category to identify the kind of errors being denoted by it. For example, all the errors occurring at the compile time can be put under one category and the ones occurring at run time in another one.
Along with the 9 clusters that we had obtained before, we also need to consider the log files which have passed without any errors. When the lines of data from such passed log files are fed into the machine learning algorithm, it has to classify them into the cluster which informs the user that the data seems to be good and not responsible for any kind of error. A -1 label is given to the cluster with the good data.

These clusters were named from -1 to 8, where -1 is the cluster that consists of data that does not lead to any error. The initial process considered this to be the good data.
In the below table you can see the clusters obtained from the hierarchical clustering and also the k means clustering, along with the final clustering label we decided on, after consulting the team and getting some manual help.
This method of clustering seems to be the right one giving us good results. But sometimes when the data labeled as an error is compared with the good data, gives rise to some issues. Which can be dealt with by a manual relabelling process.

Message	Hier label	K-means label	final label
Error 1	3	8	-1
Error 1	3	8	-1
Error 2	1	9	-1
Error 2	1	9	-1
Error 2	1	9	-1
Error 1	3	8	-1
Error 1	3	8	-1
Error 3	8	0	-1
Error 4	4	3	-1
Error 4	4	3	-1

Table 4.1: Data labelling

4.5 Re-labeling after Analysis

After clustering and labeling the entire failed error log data, there seems to be 9 clusters of error that have been analyzed with the help of keyword filtering as seen in the section 3.4.2 that consist of around 900 such data. Those 9 clusters indicate 9 log events that cause the problem. The keyword filtering is based on the assumption that the error lines contains one of those words from the negative words mentioned, that indicate a fail log event. Therefore, there has to be a manual verification process that needs to be performed with the help of pass log data files. We have thus collected pass log data files, that are higher in number than the fail log files and are also pre-processed in the same way as the fail data mentioned in the data pipeline section 3.4.

Once all the pass data has been pre-processed, there are 921 log data found from the pass log file filtering. Now, those log lines are exported into an excel sheet for comparison with the fail log data that are clustered into 9 classes. After careful examination with the help of experienced software developers, 9 classes of error clusters are reduced to 3 classes of error log events, because there are some common log events present in the passed log filtered data that do not indicate or cause any errors in the fail logs but just contains error keywords. Finally, 9 error log events are reduced into 3 log events, which are labeled as classes 1, 2, and 3. Moreover, the normal/ good log lines that are present in the past log data files are labeled as

-1, which includes all kinds of log events that are general in log files and are not resposible for the cause of any error.

Log events	Class labels
error 1	1
error 2	2
error 3	3

Table 4.2: Types of errors

4.6 Data Sampling

Once the data samples are labelled and refined, they are stored as pandas data-series frame and the count for each class can be determined with the help of Python function *.value_counts*() which gives the count of each categorical classes in the data.

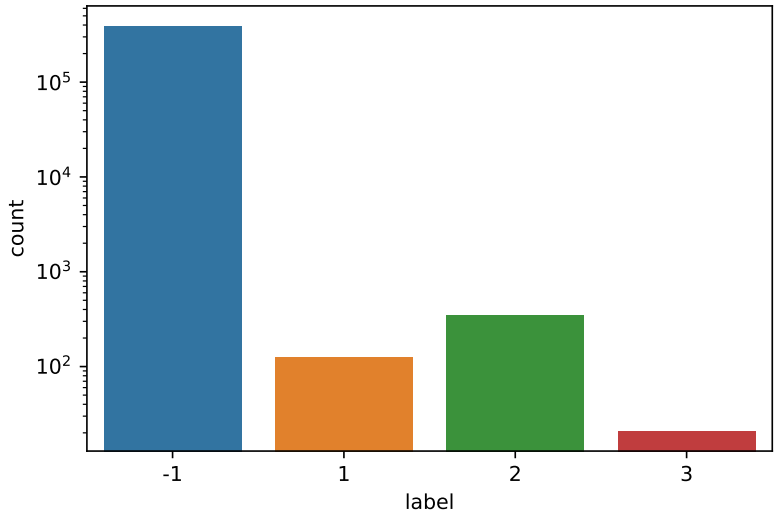


Figure 4.7: Count chart of classes

Figure 4.7 indicates the No of counts bar for all the classes.

Label	Count
-1	389273
1	352
2	125
3	21

Table 4.3: Count of class labels

From the table 4.3 and figure 4.7, we can see that the data-set is highly unbalanced as the count of the normal data -1 represents almost 389273 lines of data and all the other error log lines are in the hundreds range. This indicates the data-set is highly imbalanced with large variations in the class counts. To make the data-set a

balanced one, we have over-sampled it using the SMOTE method, which has been discussed in the section 3.9.2. One of the reasons to do oversampling is to create synthetic samples on class 1,2,3 so that no more information is lost. After applying

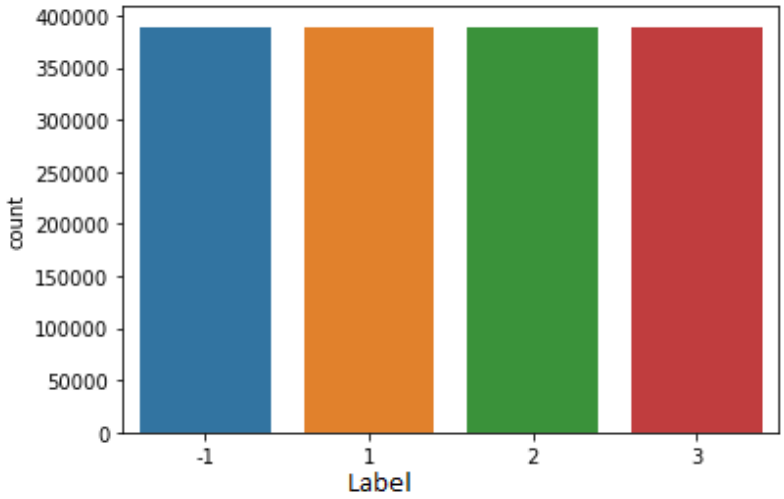


Figure 4.8: Counts after oversampling technique

the SMOTE method, from seeing the figure 4.8, all classes are oversampled to count 389273, which is the count of class -1 before sampling. In this way, the SMOTE method has been applied for oversampling technique.

4.7 Machine Learning Evaluation

Classification models discussed in the section 2.3 is trained with the data-set that is labeled with the clustering method and oversampled. The findings of the ML model's results were presented in that section. The classification model has been trained on the normal logs and the other three types of error class models which we have found in this project. The performances of SVM, Logistic regression, and Gaussian Naive Bayes are presented. The classifiers are evaluated using accuracy, precision, F1 score, and recall scores. After considering all the other metrics, Precision and recall are deemed to be the necessary metrics scores that can be used to evaluate, when the labels are multi-class and also when the data-set which is imbalanced is considered. Although we have balanced the data-set using the SMOTE method, there seems to be more of such duplicate data which can be classified wrongly. Therefore, Precision and recall are considered to be the important metrics.
The precision determines the number of data samples which has been classified as the correct prediction by the classifiers, out of all the positively predicted data samples. The recall, tells us the number of data samples that has been classified to be the correct prediction by the classifiers actual positive values. The precision is higher when the false positive (predicted negative but actually positive) is lower and the recall is higher when the false negative (Predicted positive but actually negative) is lower. Therefore the classifier needs to have both, a high precision value and recall value.

4.7.1 Confusion Matrix Evaluation

The confusion matrix showing the results for the three classifiers is seen in the figure 4.9, 4.10 and 4.11. The color bar in the figure, indicates that the count of the data samples classified on each label.

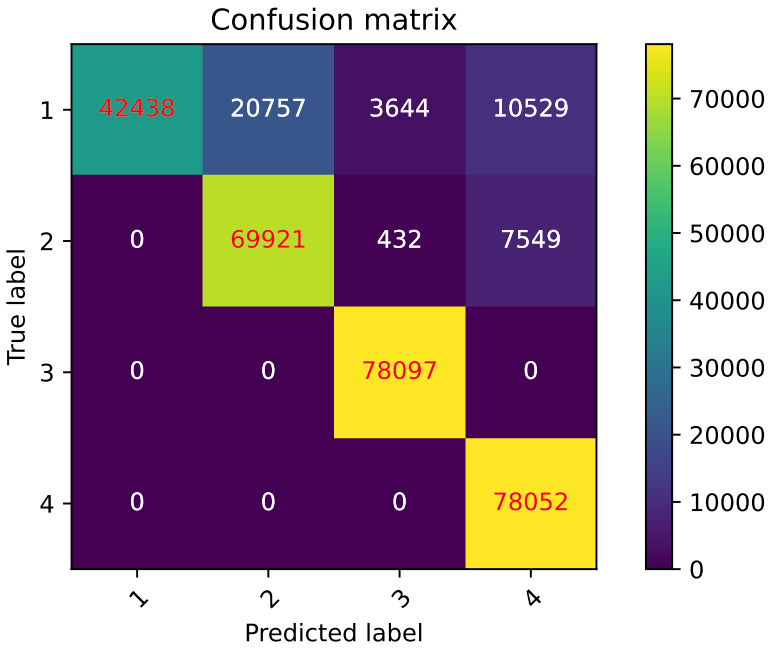


Figure 4.9: Confusion matrix for Logistic regression

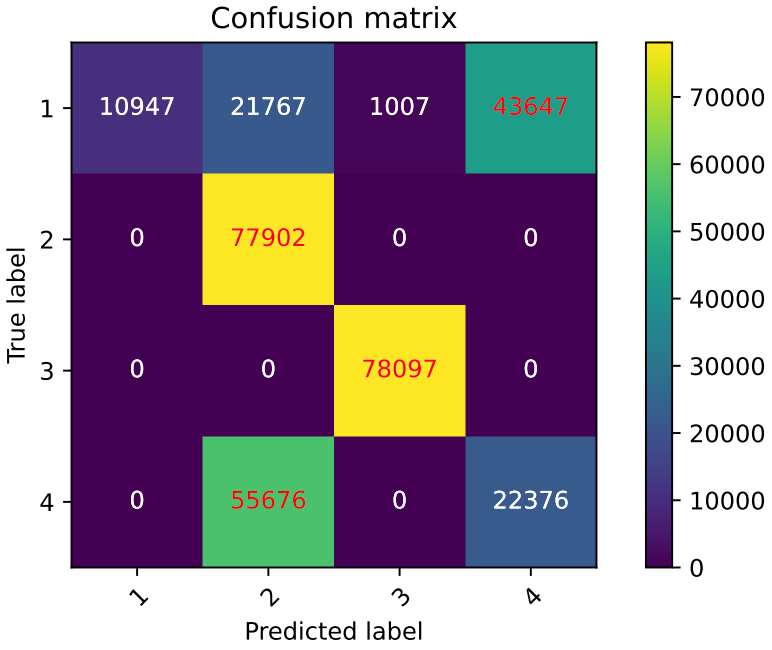


Figure 4.10: Confusion matrix for Gaussian Naive Bayes

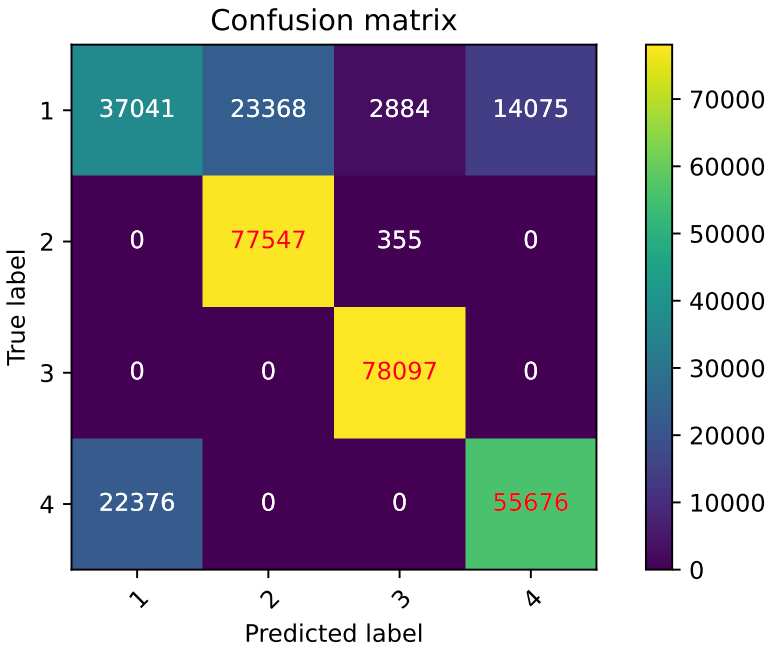


Figure 4.11: Confusion matrix for Linear support vector

Black color shows that 0 data samples are classified and white shows a higher number of data samples being classified. In the confusion matrix, the left side of the diagonal indicates the false positives and the right diagonal indicates the false negatives. When comparing with SVM and Gaussian Naive Bayes, the Logistic regression method performs better, as it has zero false positives and there is a high number of black boxes in this, which indicates that the classifiers have less probability of classifying the classes wrongly. Further, when analyzing the other classifiers, there are no data samples present diagonally although it shows a higher number of false positives and false negatives.

4.7.2 Classification Report

The Classification report tells us the precision, recall, f1 score, and the overall accuracy of the classifiers for each class as seen in the Tables 4.4, 4.5 and 4.6.

class	Precision	Recall	f1-score
-1	1.00	0.55	0.71
1	0.77	0.90	0.83
2	0.95	1.00	0.97
3	0.81	1.00	0.90

Table 4.4: classification report for Logistic regression

The overall accuracy for Logistic regression is 86.2%

class	Precision	Recall	f1-score
-1	1.00	0.14	0.25
1	0.50	1.00	0.67
2	0.99	1.00	0.99
3	0.34	0.29	0.31

Table 4.5: classification report for Gaussian Naive Bayes

The overall accuracy for Gaussian Naive Bayes is 61%

class	Precision	Recall	f1-score
-1	0.62	0.48	0.54
1	0.77	1.00	0.87
2	0.96	1.00	0.98
3	0.80	0.71	0.75

Table 4.6: classification report for Linear support vector

The overall accuracy for Linear support vector is 80%

Since we are performing multi-class classification and the synthetic data, the overall objective is to reduce the false positives and false negatives of the classifiers and to increase the precision and recall scores. Looking at the report, we can see that logistic regression performs well when compared with other classifiers as it gives an accuracy of 86%, whereas SVM and Gaussian Naive Bayes gives 60% and 79% respectively. Considering the recall score, the Gaussian Naive Bayes performs less in classifying the class -1 correctly, as it has a 14% recall score when compared to the other classes. This indicates a high number of false negatives in it.

4.8 Prediction Analysis

Finally, the machine learning model is trained with enough labels and we can see that the logistic regression gives good accuracy, with high precision and recall scores. It is now time to use the machine learning model for predicting error log events in the test log data file which hasn't been used in the training of the model. The figure 4.12, shows the imaginary pipeline which can pre-process a new log data and convert it to the numeric format such that it can be used for predicting the error log events.

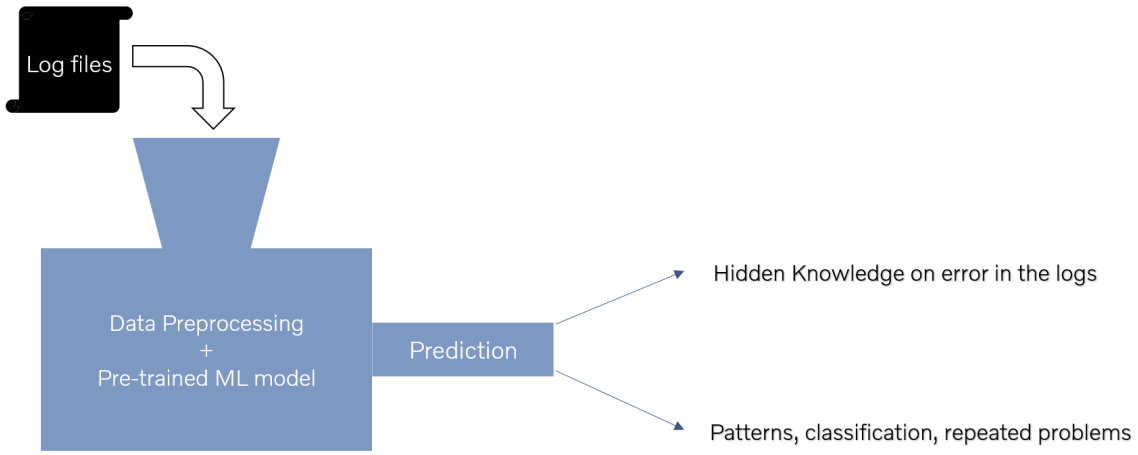


Figure 4.12: Fault Tracing using log data

Initially, we took a simple failed log file for testing the classifier and found that the log file has 214 log lines after pre-processing. All the 214 lines are then converted into a numeric format using the word embedding methods discussed in section 3.5. Once the word embedding is done, the trained machine learning model can be loaded, which makes the predication easy. After the prediction, the predicted labels are -1, 1, and 3 with their counts being 198, 15, and 1 respectively. This indicates that after carrying out the prediction, 198 log lines are normal, 15 lines are error 1, and 1 log line is error 3. On further investigating the error lines manually, out of the 16 error lines, 2 were actual errors that are causing trouble during the software integration.

5
Conclusion

5.1 Summary

Throughout the project, we have used various machine learning algorithms for clustering of the failed log data with its reason of failure and also prediction of failure in the future log data fed into the system. The log data which was used in the project was quite unstructured and the pre-processing of such log data consists of more than 50 percent of our initial work. Classification of data is done using the Hierarchical and the K-means method. The classes obtained from both these classification methods are compared, to give us the final labels for the failed log data. After this process we obtained around 9 to 10 different classes of failed data categories. During this process we also made use of data from the passed logs and compared the set of lines which appears to be in both these sets of log data i.e., passed logs and failed logs. If the data is in both the sets and still seems to be categorised as one of the reasons of failure i.e., one of the label, then we substitute such data to belonging to another label -1, which consists of data that did not lead to the actual failure of the files. While the classification of the failed log data is now done, this is not the only thing which is going to hold value for this algorithm or the project to be used in the future. For this purpose machine learning algorithms like, Support Vector Machine, Logistic Regression and Gaussian Naive Bayes are used for the failure prediction in log data files. Among the three machine learning algorithms, we see that the Logistic regression method gives us the best accuracy. This was even justified when checked with new failed log data and the predicted lines of failure was manually checked by the people well-versed in the field. In the future, if there are hundreds of files with thousands of such lines, instead of going through them one by one and manually finding the lines of data which could be reason why the log data failed in the first place, this algorithm could do all the work for us and just pick out the failure lines for further analysis.

5.2 Future work

The log data which was provided did not have a structured format which was followed which hindered our machine learning process initially. As a lot of time had to be spent on cleaning the data and preprocessing it. Even during the preprocessing of the log data, because of no particular structure, figuring out the pattern was a tedious process. Log data with time frame attached to every line along with the formatting would help the machine learning process to final even predict the time

around which the error seemed to have been occurring after running the jobs. Which would give us a better understanding too. Although the prediction accuracy is good, it is not in its best shape as the data is not in its best potential. So if the data is in a good format or structure, then the accuracy could be even better.
For the purpose of the thesis, we have now worked on only one of the jobs and its analysis. In the future all the other jobs can be analysed in the same way by using the same functions or the algorithms that has been used in the thesis, which helps in finding the other kinds of errors and its frequency of occurrence throughout the project.
As the pre-processing of data was the major part of the project, the same algorithm or pipeline as mentioned in 3.4 used here can be utilised for enhancing the analysis done by trying to find the hidden root cause of the error, which could be the next phase of the project.

Bibliography

[1] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 807–817, 2019.

[2] A. Tolstikov, "Machine learning approaches for failure type detection and predictive maintenance," 2015.

[3] S. S. M. Kommineni and Dindi, "Automating log analysis (dissertation)," 2021.

[4] S. Cai, L. Zhang, A. Palazoglu, and J. Hu, "Clustering analysis of process alarms using word embedding," *Journal of Process Control*, vol. 83, pp. 11–19, 2019.

[5] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013.

[6] G. E. A. P. A. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *SIGKDD Explor. Newsl.*, vol. 6, p. 20–29, jun 2004.

[7] "Introduction to AUTOSAR, wherepublished = `https://medium.com/mathworks/introduction-to-autosar-b5d383638c7`."

[8] H. Mi, H. Wang, Y. Zhou, M. R.-T. Lyu, and H. Cai, "Toward fine-grained, unsupervised, scalable performance diagnosis for production cloud computing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 6, pp. 1245–1255, 2013.

[9] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," (New York, NY, USA), Association for Computing Machinery, 2017.

[10] C. Zhang, J. Liu, W. Chen, J. Shi, M. Yao, X. Yan, N. Xu, and D. Chen, "Unsupervised anomaly detection based on deep autoencoding and clustering," *Security and Communication Networks*, vol. 2021, 2021.

[11] "UnSupervised Machine Learning, wherepublished = `https://www.geeksforgeeks.org/supervised-unsupervised-learning/`."

[12] S. Baek, D. Kwon, S. C. Suh, H. Kim, I. Kim, and J. Kim, "Clustering-based label estimation for network anomaly detection," *Digital Communications and Networks*, vol. 7, no. 1, pp. 37–44, 2021.

[13] M. Ahmed, R. Seraj, and S. M. S. Islam, "The k-means algorithm: A comprehensive survey and performance evaluation," *Electronics*, vol. 9, no. 8, 2020.

[14] J. Zhang, W. Chen, M. Gao, and G. Shen, "K-means-clustering-based fiber nonlinearity equalization techniques for 64-qam coherent optical communication system," *Opt. Express*, vol. 25, pp. 27570–27580, Oct 2017.

[15] E. Umargono, J. Suseno, and S. Gunawan, "K-means clustering optimization using the elbow method and early centroid determination based on mean and median formula," 01 2020.

[16] F. Murtagh and P. Contreras, "Methods of hierarchical clustering," *Computing Research Repository - CORR*, 04 2011.

[17] "Supervised Machine Learning, wherepublished = `https://www.javatpoint.com/supervised-machine-learning`."

[18] "Support Vector Machine." `https://medium.com/@hakobavjyan/overview-of-supervised-learning-model-svm-support-vector-\machines-20b683a4eaf`.

[19] "Logistic Regression, wherepublished = `https://www.javatpoint.com/logistic-regression-in-machine-learning`."

[20] "Semi Supervised Machine Learning, wherepublished = `https://www.geeksforgeeks.org/ml-semi-supervised-learning/`."

[21] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman, "Natural language processing: an introduction," *Journal of the American Medical Informatics Association*, vol. 18, pp. 544–551, 09 2011.

[22] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018)*, 2018.

[23] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[24] "Principal Component Analysis for Dimensionality Reduction, wherepublished = `https://towardsdatascience.com/principal-component-analysis-for-dimensionality-reduction-\115a3d157bad`."

[25] "TF - IDF, wherepublished = `https://towardsdatascience.com/essential-things-you-need-to-know-about-f1-score-dbd973bf1a3`."

[26] "Python - Regular Expressions, wherepublished = `https://www.tutorialspoint.com/python/python_reg_expressions.htm#:~:text=a%20regular%20expression%20is%20a,like%20regular%20expressions%20in%20python.`."

[27] "Fasttext model from gensim." `https://radimrehurek.com/gensim/models/fasttext.html`.

[28] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Information Processing Management*, vol. 24, no. 5, pp. 513–523, 1988.

[29] A.-J. Franco-Mariscal, J. Oliva, and M. L. A. Gil, "Students' perceptions about the use of educational games as a tool for teaching the periodic table of elements at the high school level," *Journal of Chemical Education*, vol. 92, pp. 278–285, 02 2015.

[30] Demidova, Liliya and Klyueva, Irina, "Improving the classification quality of the svm classifier for the imbalanced datasets on the base of ideas the smote algorithm," *ITM Web Conf.*, vol. 10, p. 02002, 2017.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY